

Berpikir Objek:
Cara Efektif
Menguasai Java

Ginanjari Utama

ginanjar_utama@yahoo.com

Table of Contents

BAB I.YOUR SURVIVAL KIT IN MASTERING JAVA	4
A.SEKILAS MENGENAI TEKNOLOGI JAVA.....	4
1.Mengapa kita perlu belajar bahasa Java?.....	4
a)Visi Java.....	4
b)Karakteristik Java	5
c)Java di Indonesia dan Dunia	7
B.INSTALASI DAN PENGENALAN LINGKUNGAN JAVA.....	8
1.Struktur Direktori Java.....	8
2.Menjalankan Program Demo dan Melihat Dokumentasi API	9
a)Dokumentasi API.....	9
3.Package dan Classpath.....	11
C.LATIHAN:	17
BAB II.PENGENALAN IDE NETBEANS DAN ECLIPSE.....	18
A.KONSEP – KONSEP PENTING	20
1.Templates.....	21
2.Filesystems.....	21
3.Modules.....	22
4.Projects.....	23
B.BERKELILING DALAM IDE.....	24
1.Jendela Utama.....	25
a)Menus dan Toolbars.....	25
2.Workspaces.....	26
3.GUI Editing.....	27
4.Browsing.....	30
5.Running.....	31
6.Debugging.....	32
BAB III.CORE JAVA AT MINIMUM.....	34
A.DATA TEKS.....	34
1.Character.....	34
2.String.....	34
3.StringBuffer.....	35
a)Konversi dari String ke bilangan.....	38
B.BILANGAN DAN MATEMATIKA.....	39
1.Angka-angka.....	39
b)Mengkonversi Number dari dan ke String.....	39
c)Memformat angka-angka.....	40
2.Fungsi-fungsi matematis.....	42
a)Bilangan acak (Random Numbers).....	43
3.Bilangan Besar.....	43
BAB IV.CLASS DAN OBJECT	45
A.KONSEP OBJEK.....	45
B.MODEL SEBAGAI ABSTRAKSI DARI DUNIA NYATA.....	46
1.Hirarki Model.....	46
C.CONTOH ABSTRAKSI DARI RUMAH DAN SWITCH.....	47
D.REFERENSI.....	49
E.CLASS SEBAGAI CETAK BIRU DARI OBJEK.....	50
a)Konvensi penamaan Class:.....	51
1.Field.....	51
2.Method.....	51
3.Parameter.....	52
4.Letak data dalam memori.....	52
5.Array.....	53
6.Lingkup dari deklarasi variabel	53
7.Kata kunci Static	53

BAB V.INISIALISASI DAN CLEANUP.....	55
A.CONSTRUCTOR.....	55
B.METHOD OVERLOADING.....	55
C.OVERLOADING AND RETURN VALUE.....	55
D.OVERLOADING CONSTRUCTORS.....	56
E.CONSTRUCTOR DEFAULT	56
F.REFERENSI THIS	56
G.PEMANGGILAN THIS DALAM CONSTRUCTOR	57
H.INISIALISASI VARIABEL.....	57
I.URUTAN INISIALISASI OBJECT	57
J.URUTAN INISIALISASI CLASS	58
K.INISIALISASI ARRAY	58
L.FINALIZATION DAN CLEANUP.....	59
BAB VI.REUSING CLASSES.....	60
A.KOMPOSISI OBJEK.	60
B.COMPOSITION WITH FORWARDING.....	60
C.INHERITANSI ATAU PEWARISAN.....	62
D.KAPAN MENGGUNAKAN COMPOSITITON ATAU INHERITANCE.....	70
BAB VII.POLYMORPHISM DAN INTERFACE.....	71
BAB VIII.MERANCANG CLASS	75
A.PENTINGNYA BERFIKIR SECARA OBJEK	75
B.DEKOMPOSISI PROSEDURAL DAN DEKOMPOSISI STRUKTUR OBJEK.....	76
C.MERANCANG CLASS MONEY.....	77
BAB IX.BELAJAR COLLECTIONS.....	85
A.TIPE-TIPE INTERFACE DARI COLLECTIONS.....	85
B.SETS.....	86
C.LISTS.....	86
D.MAPS.....	87
1.Tips pemrograman dan perancangan API	92
BAB X.COMPONENT SWING DAN LAYOUT MANAGER.....	97
A.KONTAINER DAN KOMPONEN.....	97
B.TOP-LEVEL CONTAINERS	
KOMPONEN-KOMPONEN YANG BERADA PADA PUNCAK SETIAP HIRARKI APLIKASI SWING.	97
C.LAYOUT MANAGEMENT YANG EFEKTIF	101
1.Apa itu Layout Manager?.....	102
2.BorderLayout.....	102
3.FlowLayout.....	103
4.BoxLayout	105
5.GridLayout.....	106
6.CardLayout.....	107
7.GridBagLayout.....	108
8.nullLayout.....	109
9.AbsoluteLayout.....	109
BAB XI.OBSERVER DAN CUSTOM EVENT.....	127
A.OBSERVER.....	127
B.CUSTOM EVENT.....	133
BAB XII.IKHTISAR ATURAN-ATURAN JAVABEAN	138
1.Konvensi nama untuk atribut.....	138
B.EVENT-EVENT DALAM JAVA BEANS.....	138
1.Predefined Event Sets.....	139
B.MENGIRIMKAN EVENT KEPADA LISTENERS.....	140

1. Awal :	141
2. Menambah komponen :	141
3. Memodifikasi atribut dari komponen :	142
4. Menambah kode untuk fungsionalitas :	143
5. Menambahkan TimerBean, dan mensest sebuah event handler :	143
C. TAMBAHAN MODIFIKASI (UNTUK JAVA BEANS SUDAH SELESAI)	145
1. Menambah event handler	147
BAB XIII. EVOLUSI KODE	149
BAB XIV. SERVICE ORIENTED OBJECT	157
A. DATA ORIENTED MATRIX	157
B. SERVICE ORIENTED MATRIX	159
C. SERVICE ORIENTED STAMP DISPENSER	163
1. Versi light dengan pendekatan objek	164
2. Dengan pendekatan prosedural	166
3. Implementasi yang berbeda dengan State pattern	169
BAB XV. ACCOUNT BY BILL VENNER	172



Bab I. Your Survival Kit in Mastering Java

A. Sekilas mengenai teknologi Java

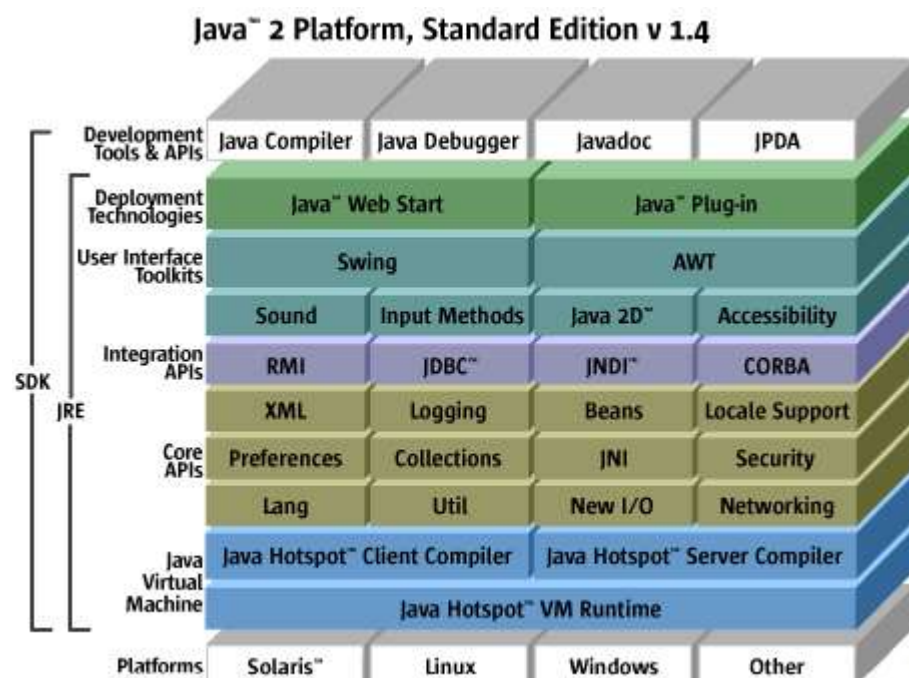
Globalisasi informasi dan konvergensi dari berbagai disiplin ilmu pengetahuan sekarang ini menyebabkan manusia mampu merealisasikan mimpi-mimpi generasi masa lalu. Perangkat keras yang lebih baik --dengan masih mematuhi hukum Moore yang menyatakan bahwa kapasitas microprocessor berlipat dua dengan harga setengahnya setiap 18 bulan --, perangkat lunak yang lebih besar dan kompleks --sekaligus juga mudah dipakai, lebih handal, bahkan gratis untuk Open Source software--, perkembangan jaringan baik itu Internet maupun wireless network dalam fase inflasioner dengan beraneka ragam perangkat keras dan lunak di dalamnya. Semua hal tadi menyebabkan kita harus berhenti untuk berfikir dan merenung sejenak bagaimana kita bisa mengelola kompleksitas yang tinggi dan perubahan yang cepat tersebut.

1. Mengapa kita perlu belajar bahasa Java?

a) Visi Java.

Java pertama kali diluncurkan pada tahun 1995 sebagai bahasa pemrograman umum (general purpose programming language) dengan kelebihan dia bisa dijalankan di web browser sebagai applet. Sejak awal, para pembuat Java telah menanamkan visi mereka ke dalam Java untuk membuat piranti-piranti yang ada di rumah (small embedded customer device) seperti TV, telepon, radio, dan sebagainya supaya dapat berkomunikasi satu sama lain. Tentu saja jalan menuju visi ini tidak mudah untuk ditemukan apalagi untuk ditempuh. Langkah pertama yang diambil oleh Sun Microsystem adalah dengan membuat JVM (Java Virtual Machine) yang kemudian diimplementasikan dalam bentuk JRE (Java Runtime Environment). JVM adalah lingkungan tempat eksekusi program Java berlangsung dimana para objek saling berinteraksi satu dengan yang lainnya. Virtual Machine inilah yang menyebabkan Java mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar.

Apabila kita hanya ingin menjalankan program Java, maka kita cukup memiliki JRE saja. Tapi seandainya kita ingin mengembangkan perangkat lunak sendiri, JRE saja tidak cukup. Untuk lebih meningkatkan produktivitas pengembang perangkat lunak, Sun juga meluncurkan SDK (Standard Development Kit) yang berisi kelas dan API untuk membuat program aplikasi berbasis Java. Pada tahun 1999 Sun meluncurkan J2EE (Java 2 Enterprise Edition) sebagai framework untuk membuat aplikasi enterpris berskala besar. Pada tahun 2001, Sun meluncurkan J2ME yang kelak menjadi salah satu standar pemrograman di dalam PDA maupun handphone. Komunitas OpenSource sendiri mempunyai platform yang disebut dengan Jini untuk merealisasikan visi awal dari Java.

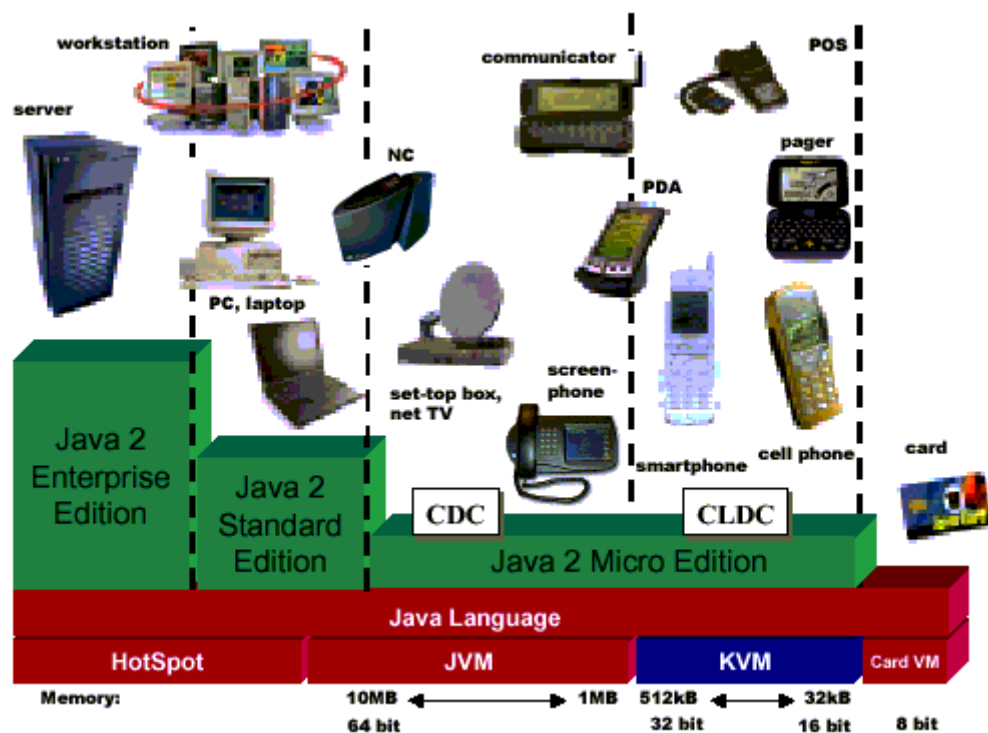


Gambar 1 Java API

b) Karakteristik Java

- Sederhana, semudah C dan seampuh C++: berlawanan dengan anggapan orang-orang bahwa bahasa Java sulit untuk dipelajari, Java gampang untuk dipelajari terutama untuk orang yang sudah mengenal pemrograman tapi belum terlalu terikat pada paradigma pemrograman prosedural. Tentu saja ini berarti bahwa kita harus siap mempelajari salah satu teknologi yang berkembang paling cepat di dunia dalam dua tahun terakhir ini dengan banyak membaca tentunya baik dari buku maupun melalui web.
- Sangat berorientasi objek (OOP) dengan implementasi yang sangat baik sehingga kita bukan hanya belajar bagaimana membuat program yang baik (*reusable*, *scalable*, dan *maintanable*) tetapi juga kita belajar bagaimana cara berfikir yang baik untuk mengenali struktur masalah yang sedang kita hadapi dan memecahkannya secara sistematis dengan pola-pola tertentu (*patterns*). Bahasa berorientasi objek biasanya mempunyai ciri-ciri sebagai berikut:
 - Abstraksi yang baik untuk memudahkan pemahaman dan komunikasi.
 - Enkapsulasi: penyembunyian informasi dari pihak-pihak yang tidak memerlukannya sehingga kompleksitas tetap tertangani dengan baik.
 - Pewarisan dan polymorphism, teknik yang menyebabkan objek menjadi modular dan mudah untuk dicopot dan dipasang objek yang lain

- Komposisi dan Interface, berguna untuk membuat tingkat kopling yang lebih rendah dan struktur hirarki objek-objek yang rapih sehingga kita bisa mengintegrasikan komponen-komponen untuk membuat sistem yang besar.
- OpenPlatform, Write Once Run Anywhere (WORA), portabel atau multi platform, program yang kita buat dapat dijalankan di Windows, Linux/Unix, Solaris, dan Macintosh tanpa perlu diubah maupun di kompilasi ulang. Java adalah juga bahasa yang paling sesuai digunakan bersama dengan XML yang membuat data menjadi portabel, ini karena kelahiran XML tidak terlepas dari dukungan parser-parser berbahasa Java. Selain itu Java turut serta dalam mengkonvergenkan protokol menjadi Open protokol yaitu IP (Internet Protocol) terutama dalam Micro Java
- Arsitekturnya yang kokoh dan pemrograman yang aman didukung oleh komunitas Open Source (ketiga terbesar setelah C dan C++ di SourceForge.net dan implementasi bahasa Java sudah menjadi milik umum). Dalam Java program yang kita buat tidak mudah untuk "hang" karena konflik pada memori biasanya diselesaikan dengan mengumpulkan objek-objek yang sudah tak terpakai lagi secara otomatis oleh garbage collector. Penanganan kesalahan juga dipermudah dalam Java dengan konsep Exception.
- Bukan sekedar bahasa tapi juga platform sekaligus arsitektur. Java mempunyai portabilitas yang sangat tinggi. Ia dapat berada pada smartcard, pager, POS (Point of Service), handphone, PDA, palm, TV, Embedded device (PLC, micro controller), laptop, pc, dan bahkan server). Menyadari akan hal ini (one size doesn't fit all) Sun membagi arsitektur Java menjadi tiga bagian, yaitu:
 1. Enterprise Java (J2EE) untuk aplikasi berbasis web, aplikasi sistem tersebar dengan beraneka ragam klien dengan kompleksitas yang tinggi. Merupakan superset dari Standar Java
 2. Standar Java (J2SE), ini adalah yang biasa kita kenal sebagai bahasa Java, dan merupakan fokus kita sekarang.
 3. Micro Java (J2ME) merupakan subset dari J2SE dan salah satu aplikasinya yang banyak dipakai adalah untuk wireless device / mobile device



Gambar 2 Hirarki dan Portabilitas Teknologi Java

- Fitur-fitur utama yang lain:
 - Mendukung multi-threading
 - Selalu memeriksa tipe object pada saat run-time
 - Mempunyai automatic garbage collection untuk membersihkan objek yang tidak terpakai dari memori
 - Mendukung exception sebagai salah satu cara penanganan kesalahan

Gambar di bawah ini adalah contoh dari peralatan J2ME pada tahun 2002 yang mempunyai JVM di dalamnya sehingga kita dapat membuat dan menjalankan program Java di dalamnya dan berkomunikasi dengan peralatan lain. Bayangkan bagaimana asyiknya jika kita bisa memrogram sendiri peralatan elektronik yang kita miliki. Bayangkan juga bagaimana perkembangan peralatan-peralatan tersebut lima sampai sepuluh tahun kedepan kelak.



c)Java di Indonesia dan Dunia

Di negeri dimana asal nama bahasa ini di ambil, bahkan di pulaunya sendiri yang banyak memiliki SDM di bidang IT, Java kurang populer bila dibandingkan dengan VisualBasic, Delphi dan Visual C++. Hal ini mungkin disebabkan oleh tidak adanya kemauan yang kuat dari kita untuk menjadi mandiri dalam bidang software. Tahukah anda bahwa tidak kurang dari 50% software-software di Amerika di impor dari negara-negara Asia terutama India dan Cina.

Visual Basic dan Delphi masih memegang posisi teratas untuk client-side programming, sedangkan untuk server-side, Java telah mengambil alih hampir seluruh market. Bisa dilihat dari produk-produk Application Server yang semuanya harus memenuhi

persyaratan J2EE compliance seperti IBM Web Sphere, Oracle Application Server, BEA WebLogic, Sun iPlanet ApplicationServer, JBoss dan lain-lain.

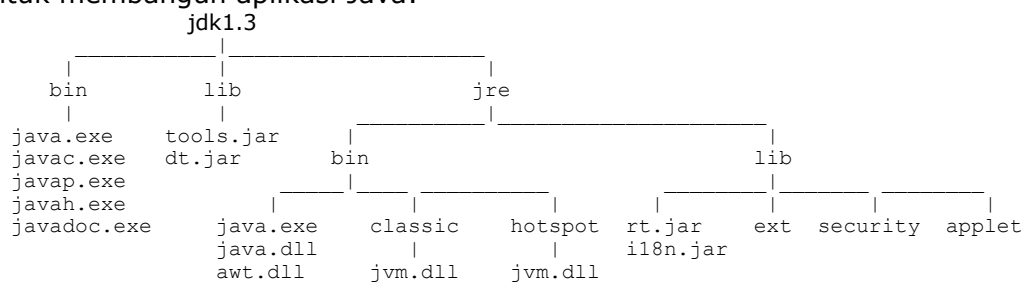
Karena portabilitasnya yang sangat tinggi maka Java merupakan platform yang ideal untuk dapat beralih ke OpenSource. Ini berarti perusahaan tidak kehilangan investasinya dalam perangkat lunak yang mahal dan sulit untuk di buat ulang. Contohnya sekarang ini banyak sekali perusahaan yang memakai sistem operasi maupun aplikasi umum yang ilegal (bajakan), kemudian perusahaan itu mengembangkan aplikasi sendiri yang berjalan pada sistem operasi maupun pada aplikasi office bajakan tersebut. Seandainya nanti peraturan semakin ketat, hukum ditegakkan maka perusahaan itu mempunyai pilihan untuk membayar lisensi atau menggunakan produk-produk gratis dari OpenSource, seperti Linux dan OpenOffice.

B.Instalasi dan Pengenalan Lingkungan Java

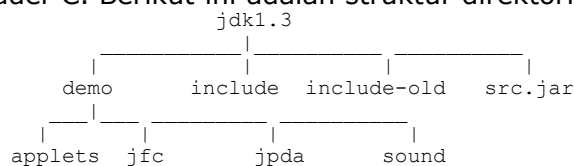
1.Struktur Direktori Java

Seluruh software yang berkaitan dengan Java dapat didownload secara gratis di java.sun.com (The Source of Java Technologies). Setelah mendownload J2SE (versi terakhir sekarang: 1.4.2_01) maka untuk menginstallnya cukup dengan menjalankan programnya saja atau dengan program instalasi (di Linux dengan Package Manager).

J2SE kemudian akan diekstrak pada struktur direktori tertentu Struktur berikut ini menampilkan direktori dan file-file penting yang sering digunakan untuk membangun aplikasi Java:



Kemudian ada lagi file-file dan direktori tambahan untuk demo, kode sumber dari Java API, dan file header C. Berikut ini adalah struktur direktorinya:



Untuk memastikan J2SE telah terpasang dengan benar dalam komputer kita, coba ketikkan pada MS-DOS prompt:

```
C:\>java -version
```

Jika keluarannya seperti ini:

```
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1-b24)
Java HotSpot(TM) Client VM (build 1.3.1-b24, mixed mode)
```

anda telah berhasil memasang JDK.Untuk versi yang lain tampilannya sudah pasti berbeda pula

Jika keluarannya: `Bad command or file name`, maka kita perlu menambah setting variabel lingkungan `PATH` pada sistem kita untuk menunjuk direktori `bin\` pada direktori instalasi Java.

Contoh: bila anda menggunakan WindowsXP maka anda bisa langsung menambahkan `PATH` pada Environment Variable pada properties dari My Computer. Jika anda menggunakan Windows versi sebelumnya maka anda perlu merubah file `Autoexec.bat` dan menambahkan `SET PATH = %PATH%;C:\jdk1.3.1_01\bin` lalu melakukan reboot.

2.Menjalankan Program Demo dan Melihat Dokumentasi API

Selanjutnya kita akan mencoba menjalankan program Demo untuk melihat bagaimana sebuah program Java dijalankan.

```
C:\jdk1.3.1_01\DEMO\JFC>cd SwingSet2
C:\jdk1.3.1_01\DEMO\JFC\SwingSet2>dir

Volume in drive C has no label
Volume Serial Number is 3852-15DD
Directory of C:\jdk1.3.1_01\DEMO\JFC\SwingSet2

.                <DIR>          10/11/01  5:52a  .
..               <DIR>          10/11/01  5:52a  ..
README          TXT             335  08/08/01  2:13p  README.TXT
SWINGS~1        HTM             314  08/08/01  2:13p  SwingSet2.html
SWINGS~1        JAR             1.357.238  08/08/01  2:13p  SwingSet2.jar
SWINGS~2        HTM             1.085  08/08/01  2:13p  SwingSet2Plugin.html
SRC              <DIR>          10/11/01  5:52a  SRC
                4 file(s)      1.358.972 bytes
                3 dir(s)      4.533.94 MB free

C:\jdk1.3.1_01\DEMO\JFC\SwingSet2>java -jar SwingSet2.jar

C:\jdk1.3.1_01\DEMO\JFC\SwingSet2>appletviewer SwingSet2.html
```

Sekarang cobalah untuk menjalankan `SwingSet2.html` dari browser anda, jika tidak berjalan maka ini berarti Java-Plugin belum terpasang. Untuk versi 1.4 otomatis memasang Java-Plugin. Kita coba lagi untuk menjalan satu program demo lagi untuk melihat kemampuan Java dalam grafik 2D

```
C:\jdk1.3.1_01\DEMO\JFC\Java2D>java -jar Java2Demo.jar
```

a)Dokumentasi API

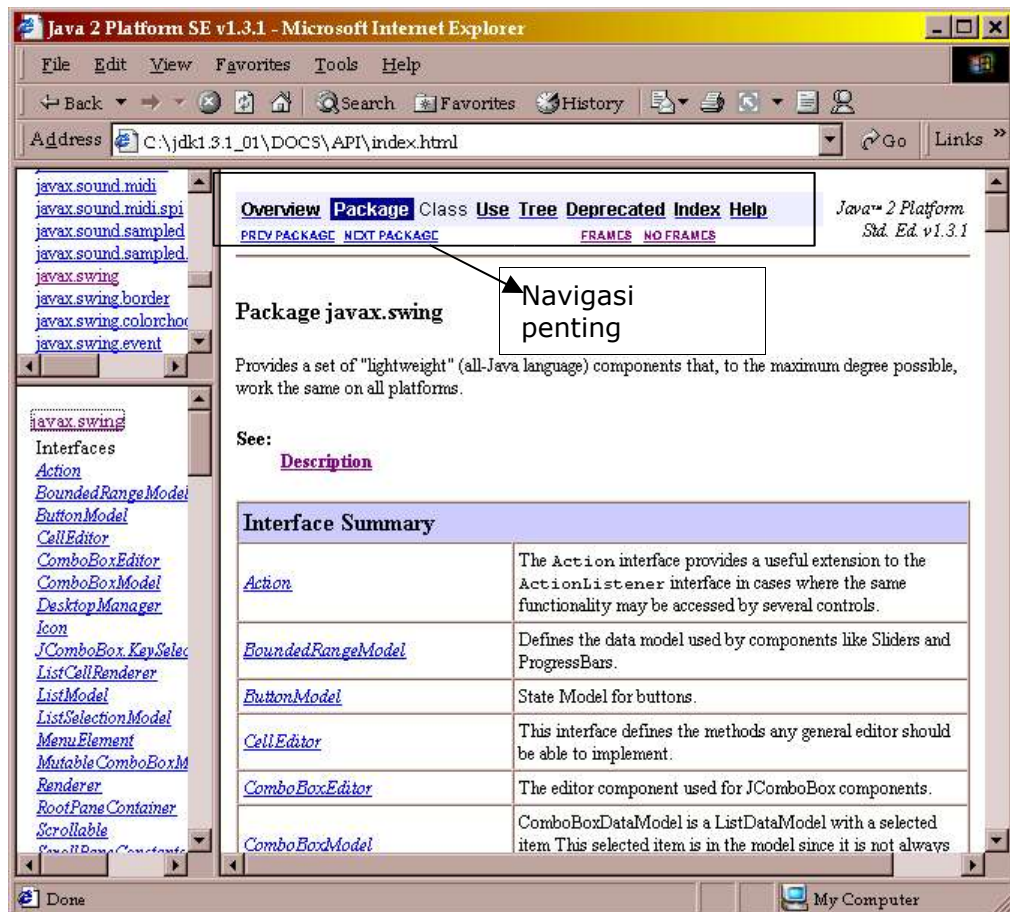
Dokumentasi pustaka class atau API bisa didownload secara langsung (terpisah dari SDK) atau kita dapat membuatnya dengan `javadoc`. Jika anda telah mendownloadnya maka ekstrak ke dalam direktori tempat Java dipasang. Dokumentasi API wajib untuk dimiliki setiap pemrogram dalam bahasa Java, karena ia merupakan pintu masuk dari semua fasilitas yang diberikan oleh Java. Sun sendiri mempunyai tujuan agar API ini merupakan abstraksi interface dari sistem operasi yang berarti kita tidak bergantung terhadap sistem operasi di mana kita berada.

Memrogram dalam bahasa Java membutuhkan pemahaman yang baik tentang API, karena semua masalah yang biasa dihadapi oleh pemrogram sehari-hari biasanya sudah diselesaikan dalam API. Jangan pernah ragu dalam menginvestasikan waktu anda untuk mempelajari API dan model-model pemrogramannya.

Dokumentasi API mempunyai format HTML dan mendeskripsikan interface-interface dan class-class yang sudah ada. Kadang-kadang juga diberikan cara penggunaan dari class tersebut. Jika kita ingin melihat contoh penggunaan dari API tersebut salah satunya adalah di <http://www.javaalmanac.com>

Contoh API yang ada:

- I/O: data dari/ke file, string, socket, ...
- GUI: AWT, event model, Swing
- Komponen-komponen: JavaBeans
- Pemrograman jaringan: Applets, java.net, RMI
- Mengakses basis-data SQL: JDBC



Gambar 3 Dokumentasi API

Biasakan untuk membaca ringkasan interface dari setiap package untuk mengetahui bagaimana mengoptimalkan pemanfaatan package tersebut. Selain itu, anda juga harus mengenal dengan baik navigasi di dalam dokumentasi API seperti yang ditunjukkan dalam gambar di atas.

Jika anda ingin membuat sebuah API yang nantinya akan digunakan oleh banyak orang, maka anda dapat melihat kode sumber dari API Java. Apabila anda mempelajarinya maka anda telah belajar bagaimana merancang object dengan baik. Cara melihatnya adalah dengan mengekstrak file `src.jar` pada direktori instalasi. Contoh:

```
C:\jdk1.3.1_01>dir src.jar

Volume in drive C has no label
Volume Serial Number is 3852-15DD
Directory of C:\jdk1.3.1_01

SRC          JAR          19.638.746   08/08/01   2:13p  SRC.JAR
1 file(s)   19.638.746 bytes
0 dir(s)    4.458.61 MB free
```

```
C:\jdk1.3.1_01>jar xvf src.jar
```

akan terbentuk direktori `src` yang berisi kode sumber API Java. Silahkan lihat bagaimana implementasi dari class `java.util.Date` pada file `src\java\util\Date.java`. Untuk J2SDK 1.4.0 maka filenya adalah `src.zip` dan dapat di ekstrak dengan WinZIP atau RAR

3.Package dan Classpath

Sekarang kita akan membuat sebuah program kecil untuk membiasakan berbicara dalam bahasa Java.

Buatlah sebuah direktori `learn` dan buat file text `HelloDate1.java` di dalam direktori tersebut. Ini bukanlah contoh program berorientasi objek yang baik, dan hanya dimaksudkan untuk menjelaskan cara kompilasi dan eksekusi, menerangkan konsep package secara ringkas, cara menampilkan hasil ke layar dan bagaimana menggunakan API

```
/*
 * HelloDate1.java
 *
 * Created on March 4, 2002, 8:50 AM
 */

package learn;
import java.util.Date;

/**
 *
 * @author Ginanjar Utama
 * @version
 */
public class HelloDate1 {

    /** Creates new HelloDate */
    public HelloDate1() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main (String args[]) {
        System.out.println("Sekarang tanggal " + new Date());
        //Contoh output Sekarang: tanggal Mon Mar 04 08:51:57 ICT 2002
    }

}
}
```

```
C:\NETBEANS\samplendir\learn>dir
```

```
Volume in drive C has no label
Volume Serial Number is 3852-15DD
Directory of C:\NETBEANS\samplendir\learn

.                <DIR>          04/03/02  8:27a  .
..               <DIR>          04/03/02  8:27a  ..
HELLOD~1 JAV      331  04/03/02  8:51a  HelloDate1.java~
NBATTR~1        84   04/03/02  8:50a  .nbattr
HELLOD~3 JAV     418  04/03/02  8:51a  HelloDate1.java
          3 file(s)                833 bytes
          2 dir(s)                5.184.84 MB free
```

Cara mengkompilasinya adalah

```
C:\NETBEANS\samplendir\learn>javac HelloDate1.java
```

Hasil kompilasinya adalah `HelloDate1.class`, inilah yang disebut dengan bytecode

```
C:\NETBEANS\samplendir\learn>dir
```

```
Volume in drive C has no label
Volume Serial Number is 3852-15DD
Directory of C:\NETBEANS\samplendir\learn

.                <DIR>          04/03/02  8:27a  .
..               <DIR>          04/03/02  8:27a  ..
HELLOD~1 JAV      331  04/03/02  8:51a  HelloDate1.java~
HELLOD~1 CLA      688  04/03/02  9:08a  HelloDate1.class
NBATTR~1         84  04/03/02  8:50a  .nbattr
HELLOD~3 JAV      418  04/03/02  8:51a  HelloDate1.java
               4 file(s)          1,521 bytes
               2 dir(s)          5,172.84 MB free
```

kemudian kita coba menjalankannya dengan

```
C:\NETBEANS\samplendir\learn>java HelloDate1.class
Exception in thread "main" java.lang.NoClassDefFoundError: HelloDate1/class
```

Untuk menjalankannya tidak perlu menggunakan ekstensi .class

```
C:\NETBEANS\samplendir\learn>java HelloDate1
Exception in thread "main" java.lang.NoClassDefFoundError: HelloDate1 (wrong name:
learn/HelloDate1)
    at java.lang.ClassLoader.defineClass0(Native Method)
    at java.lang.ClassLoader.defineClass(Unknown Source)
    at java.security.SecureClassLoader.defineClass(Unknown Source)
    at java.net.URLClassLoader.defineClass(Unknown Source)
    at java.net.URLClassLoader.access$100(Unknown Source)
    at java.net.URLClassLoader$1.run(Unknown Source)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

Karena program yang kita buat ada dalam package `learn` maka nama lengkap Class yang kita buat adalah `learn.HelloDate1`

```
C:\NETBEANS\samplendir\learn>java learn.HelloDate1
Exception in thread "main" java.lang.NoClassDefFoundError: learn/HelloDate1
```

Ternyata `jvm` juga belum berhasil menemukan Class yang ingin dijalankan.

```
C:\NETBEANS\samplendir\learn>cd ..
```

```
C:\NETBEANS\samplendir>java learn.HelloDate1
Sekarang tanggal Mon Mar 04 09:20:46 ICT 2002
```

Akhirnya berhasil juga kita menjalankan program ini.

Apabila kita masih gagal juga (terutama untuk lingkungan Linux), jangan menyerah dulu, ini berarti kita harus menset `CLASSPATH` untuk menunjuk ke *current directory* dengan `SET CLASSPATH=.`; atau pada Linux `export CLASSPATH=./:`

Empat cara memandang package:

1. Untuk menghindari nama class yang sama untuk implementasi yang berbeda dalam satu program Java. Contoh: `java.util.Date` berbeda dengan `java.sql.Date` kemudian implementasi `javax.vecmath.Matrix` mungkin berbeda implementasinya dengan `com.ginanjjar.Matrix`.
2. Cara pengorganisasian objek-objek yang mempunyai keterkaitan tinggi atau merupakan sub sistem. Misal: `com.ginanjjar.accounting.domain` untuk Domain Model, `com.ginanjjar.accounting.presentation` untuk file-file GUI dan `com.ginanjjar.accounting.persistent` untuk layer yang menangani database.
3. Package sebagai pustaka class. Contoh: `java.util`, `javax.swing`, `java.text`, `java.sql`, `java.security`. Apabila kita ingin melepas class buatan sendiri di Internet maka ikuti aturan penamaan berikut: reversed domain name + '.' + package name, misalnya `com.bruceeckel.util.Arrays`
4. Untuk *implementation hiding*. Kita dapat membuat class-class di dalam suatu package hanya bisa diakses dari dalam package itu sendiri.

Sekarang kita akan melihat lebih dalam mengenai cara kerja program ini

```
C:\NETBEANS\samplendir>javap learn.HelloDate1
Compiled from HelloDate1.java
public class learn.HelloDate1 extends java.lang.Object {
    public learn.HelloDate1();
    public static void main(java.lang.String[]);
}
```

dengan menggunakan disassembler `javap`, kita mendapat informasi bahwa `learn.HelloDate1` adalah turunan dari Class `Object`, walaupun kita tidak secara eksplisit menuliskannya dalam program.

Setiap class yang kita buat merupakan Object artinya ia adalah turunan dari class `Object` sehingga mewarisi atribut dan method-methodnya

```
C:\NETBEANS\samplendir>javap java.lang.Object
Compiled from Object.java
public class java.lang.Object {
    public java.lang.Object();
    public final native java.lang.Class getClass();
    public native int hashCode();
    public boolean equals(java.lang.Object);
    protected native java.lang.Object clone() throws
        java.lang.CloneNotSupportedException;
    public java.lang.String toString();
    public final native void notify();
    public final native void notifyAll();
    public final native void wait(long) throws java.lang.InterruptedException;
    public final void wait(long, int) throws java.lang.InterruptedException;
    public final void wait() throws java.lang.InterruptedException;
    protected void finalize() throws java.lang.Throwable;
    static {};
}
```

`learn.HelloDate1` berisi dua method, yaitu:

1. `public learn.HelloDate1();` ini adalah method khusus yang disebut dengan konstruktor dan isinya kosong
2. `public static void main(java.lang.String[]);` ini adalah method yang harus ada pada setiap program yang ingin untuk dieksekusi. Jika program yang ingin dieksekusi tidak memiliki method main maka akan ditampilkan
`java.lang.NoSuchMethodError: main`
`Exception in thread "main"`

Pada program ini kita ingin menampilkan waktu pada saat ini. Untuk menampilkan text dilayar digunakan static method `System.out.println(String string);` dan instansasi dari class `java.util.Date`. Karena class `Date` bukan dari package `java.lang` maka apabila kita tidak menuliskan dalam FQN (Fully Qualified Name), kita harus mengimpornya dengan pernyataan `import java.util.Date;`

Agar lebih jelas lagi silahkan lihat dokumentasi API untuk java.util.Date

```

Compiled from Date.java
public class java.util.Date extends java.lang.Object implements java.io.Serializable,
java.lang.Cloneable, java.lang.Comparable {
    public java.util.Date();
    public java.util.Date(long);
    public java.util.Date(int,int,int);
    public java.util.Date(int,int,int,int,int);
    public java.util.Date(int,int,int,int,int,int);
    public java.util.Date(java.lang.String);
    public java.lang.Object clone();
    public static long UTC(int, int, int, int, int, int);
    public static long parse(java.lang.String);
    public int getYear();
    public void setYear(int);
    public int getMonth();
    public void setMonth(int);
    public int getDate();
    public void setDate(int);
    public int getDay();
    public int getHours();
    public void setHours(int);
    public int getMinutes();
    public void setMinutes(int);
    public int getSeconds();
    public void setSeconds(int);
    public long getTime();
    public void setTime(long);
    public boolean before(java.util.Date);
    public boolean after(java.util.Date);
    public boolean equals(java.lang.Object);
    public int compareTo(java.util.Date);
    public int compareTo(java.lang.Object);
    public int hashCode();
    public java.lang.String toString();
    public java.lang.String toLocaleString();
    public java.lang.String toGMTString();
    public int getTimezoneOffset();
    static {};
}

```

Untuk menginstansiasi Date ada 5 pilihan konstruktor yang dapat digunakan. Dalam contoh ini kita menggunakan konstruktor yang tidak memiliki masukan untuk mengambil waktu sekarang.

Sekarang kita gabungkan menjadi: `System.out.println("Sekarang tanggal " + new Date());` Satu-satunya operator yang dioverload di dalam Java adalah operator "+". Operasinya pun terbatas hanya untuk penyambungan String saja. Apabila "+" diikuti oleh suatu object maka secara otomatis method toString() dari objek tersebut akan dipanggil dan akan mengembalikan suatu object yang bertipe String.

Sekarang kita akan mencoba program yang kedua, masih tentang waktu yaitu belajar memformat waktu dengan SimpleDateFormat

```

/*
 * HelloDate2.java
 *
 * Created on March 4, 2002, 10:33 AM
 */

package learn;
import java.text.SimpleDateFormat;
import java.util.Date;
/**
 *
 * @author Ginanjar Utama
 * @version
 */
public class HelloDate2 {

    /** Creates new HelloDate2 */

```

```
public HelloDate2() {
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    Date currentTime = new Date();
    SimpleDateFormat formatter1 = new SimpleDateFormat("yyyy.MM.dd G 'at' hh:mm:ss a zzz");
    String dateString1 = formatter1.format(currentTime);
    SimpleDateFormat formatter2 = new SimpleDateFormat("hh 'o''clock' a, zzzz");
    String dateString2 = formatter2.format(currentTime);
    System.out.println("It is now " + dateString1);
    System.out.println("It is now " + dateString2);
}
}
Contoh output:
```

```
It is now 2002.03.04 AD at 10:41:59 AM ICT
It is now 10 o'clock AM, Indochina Time
```

Untuk mengetahui lebih jauh tentang `SimpleDateFormat` silahkan lihat dokumentasi API.

Pada program ini kita membuat dua object dari satu class `SimpleDateFormat` yaitu `formatter1` dan `formatter2`. Kemudian variabel `currentTime` yang bertipe `Date` dan diinisialisasi nilainya dengan waktu sekarang, menjadi masukan dari method `format` dari kedua objek bertipe `SimpleDateFormat` yang menghasilkan variable `dateString1` dan `dateString2` yang bertipe `String`

Class `Date` merepresentasikan sebuah waktu tertentu secara spesifik dengan ketepatan millidetik. Apabila kita ingin mengambil nilai-nilai tertentu dari `Date` seperti tahun, bulan, minggu, hari, jam, menit, detik dan sebagainya maka sebaiknya kita menggunakan class `java.util.Calendar`

Program HelloDate3

```
/*
 * HelloDate3.java
 *
 * Created on March 4, 2002, 11:06 AM
 */

package learn;
import java.util.Calendar;
import java.text.SimpleDateFormat;
/**
 *
 * @author Ginanjar Utama
 * @version
 */
public class HelloDate3 {

    /** Creates new HelloDate3 */
    public HelloDate3() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main (String args[]) {
        Calendar currentTime = Calendar.getInstance();
        System.out.println(" " + currentTime);
        String timeFormat = "hh:mm:ss";
        SimpleDateFormat formatter = new SimpleDateFormat(timeFormat);
        String timeTxt = formatter.format(currentTime.getTime());
        System.out.println("Sekarang jam " + timeTxt);
        System.out.println("hari ke " + currentTime.get(currentTime.DAY_OF_YEAR) +
            " tahun " + currentTime.get(currentTime.YEAR) + " di " +
            currentTime.getTimeZone().getID());
    }
}
Contoh Output:
```

```
java.util.GregorianCalendar[time=1015214944490,areFieldsSet=true,
areAllFieldsSet=true,lenient=true,
zone=java.util.SimpleTimeZone[id=Asia/Bangkok,offset=25200000,
```

```
dstSavings=3600000,useDaylight=false,startYear=0,startMode=0,
startMonth=0,startDay=0,startDayOfWeek=0,startTime=0,startTimeMode=0,
endMode=0,endMonth=0,endDay=0,endDayOfWeek=0,endTime=0,endTimeMode=0],
firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2002,MONTH=2,
WEEK_OF_YEAR=10,WEEK_OF_MONTH=2,DAY_OF_MONTH=4,DAY_OF_YEAR=63,
DAY_OF_WEEK=2,DAY_OF_WEEK_IN_MONTH=1,AM_PM=0,HOUR=11,HOUR_OF_DAY=11,
MINUTE=9,SECOND=4,MILLISECOND=490,ZONE_OFFSET=25200000,DST_OFFSET=0]
Sekarang jam 12:40:17
hari ke 63 tahun 2002 di Asia/Bangkok
```

Catatan: banyak method pada class Date yang digantikan fungsinya oleh class Calendar. Untuk memformat penampilan waktu tetap diperlukan tipe Date sehingga diperlukan method getTime() pada Calendar.

Perhatikan pada inisialisasi variabel currentTime digunakan factory method getInstance() karena Calendar dideklarasikan abstrak maka konstruktornya tidak dapat dipanggil secara langsung dan hasil instansiasinya adalah turunannya yaitu GregorianCalendar.

C.Latihan:

1. Temukan lokasi dari class System, dan mengapa class ini tidak perlu diimpor?
2. Buat format representasi yang berbeda pada program HelloDate2 dengan membaca dokumentasi API.
3. Bandingkanlah class-class berikut ini:
 - java.util.Date
 - java.util.Calendar
 - java.util.GregorianCalendar
 - java.sql.Date
 - java.sql.Time
 - java.sql.Timestamp
 - java.text.DateFormat
 - java.text.SimpleDateFormat

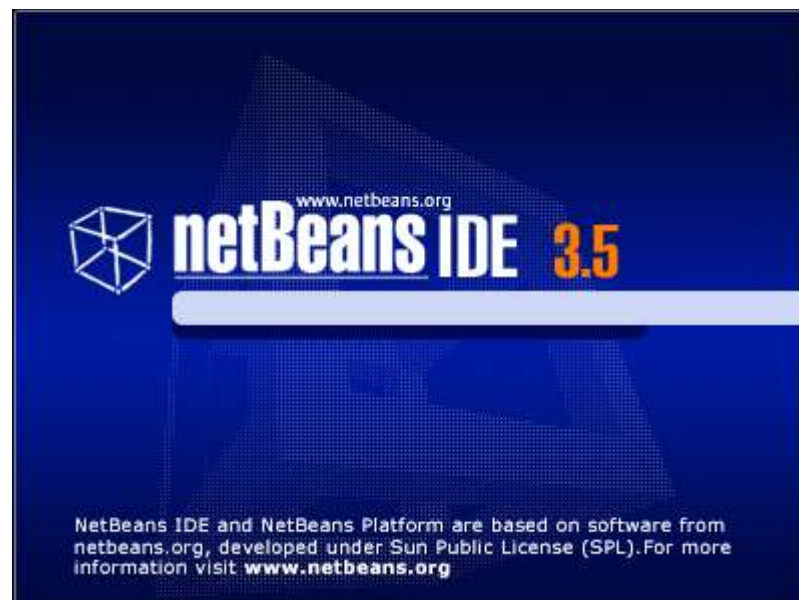
Bab II.Pengenalan IDE NetBeans dan Eclipse

Apabila kita sudah mengerti konsep kompilasi dan penggunaan classpath, maka kita bisa beralih menggunakan kakas IDE (*Integrated Development Environment*) untuk Java. IDE ini ada yang komersial seperti IntelliJIDEA dan Jbuilder, ada yang mempunyai lisensi terbatas seperti JDeveloper, dan ada juga yang free seperti jEdit, NetBeans dan Eclipse. Dalam buku ini kita hanya akan membahas NetBeans dan Eclipse.

Kedua-duanya adalah IDE OpenSource yang gratis. Versi terbaru mereka cepat dan powerfull. Kedua IDE ini merupakan contoh yang sangat baik untuk pemrograman modular. Netbeans dibangun di atas komponen-komponen yang disebut dengan module sedangkan Eclipse dibangun diatas banyak plugin. Kita dapat menemukan banyak module dan plugin di Internet untuk menambah kemampuan mereka. Kedua-duanya juga dapat menjadi platform yang sangat baik untuk menjadi dasar pengembangan aplikasi desktop.

Netbeans dibeli oleh SunMicroSystem dari sebuah perusahaan kecil bernama NetBeans di Chekoslowakia kemudian dilepas ke komunitas OpenSource. NetBeans ini juga menjadi platform dasar dari IDE komersialnya Sun yaitu SunOne (dulu dikenal dengan Forte). IBM membeli Eclipse dari OTI (*Object TechnolgyInternational*), juga perusahaan kecil di Canada yang berlatar belakang pembuat IDE untuk SmallTalk. Eclipse kemudian di open sourcekan setahun setelah NetBeans go public.

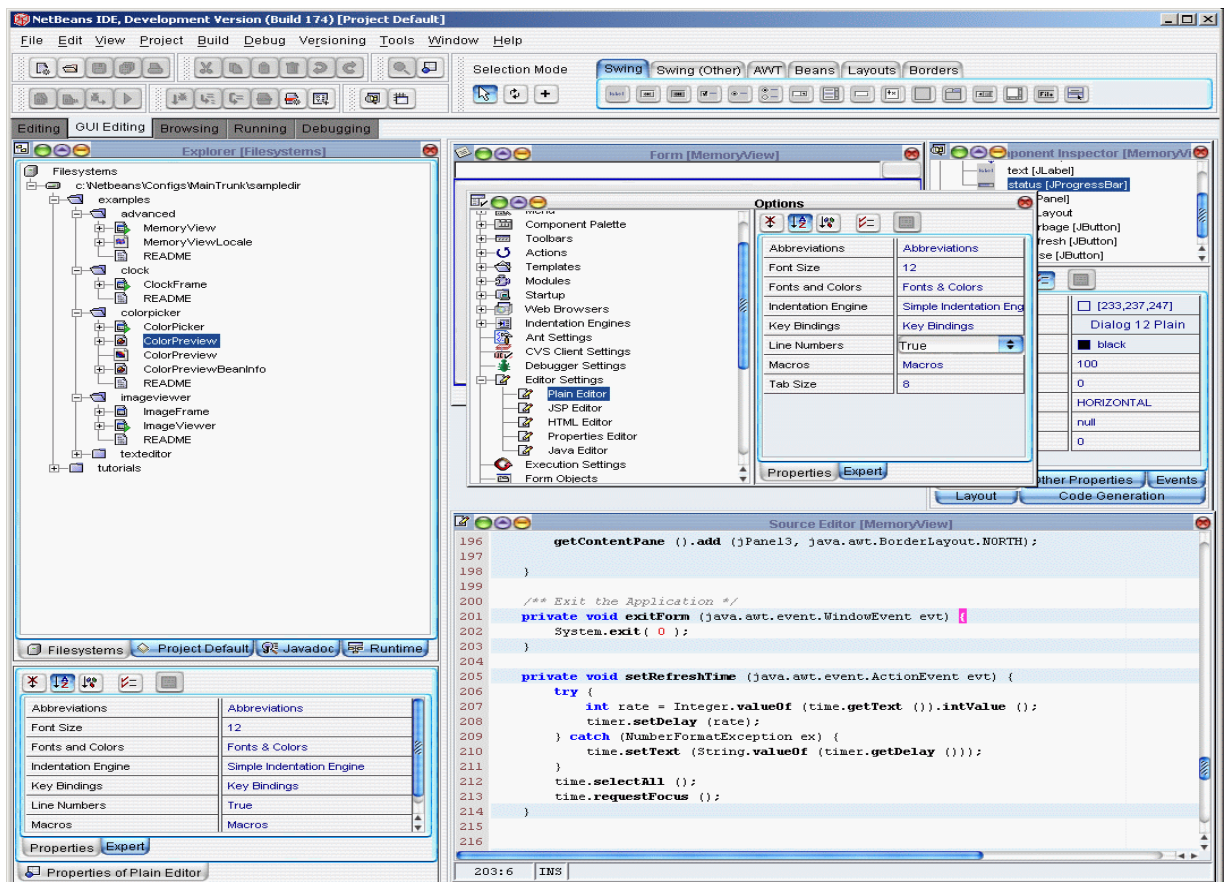
Karena memiliki visi dan latar belakang yang berbeda maka masing-masing mempunyai kelebihan yang khusus dan istimewa. NetBeans sangat bagus untuk membuat komponen bean baik non visual maupun yang visual dengan berbasiskan Swing. Eclipse mempunyai builtin JUnit dan Refactoring support yang sangat baik. Untuk pengembangan GUI aplikasi desktop, Eclipse tidak menggunakan Swing tapi menggunakan SWT dan JFace. SWT ini semacam GTK atau Motif di Linux, sedangkan JFace lebih mirip dengan WindowsManagernya.



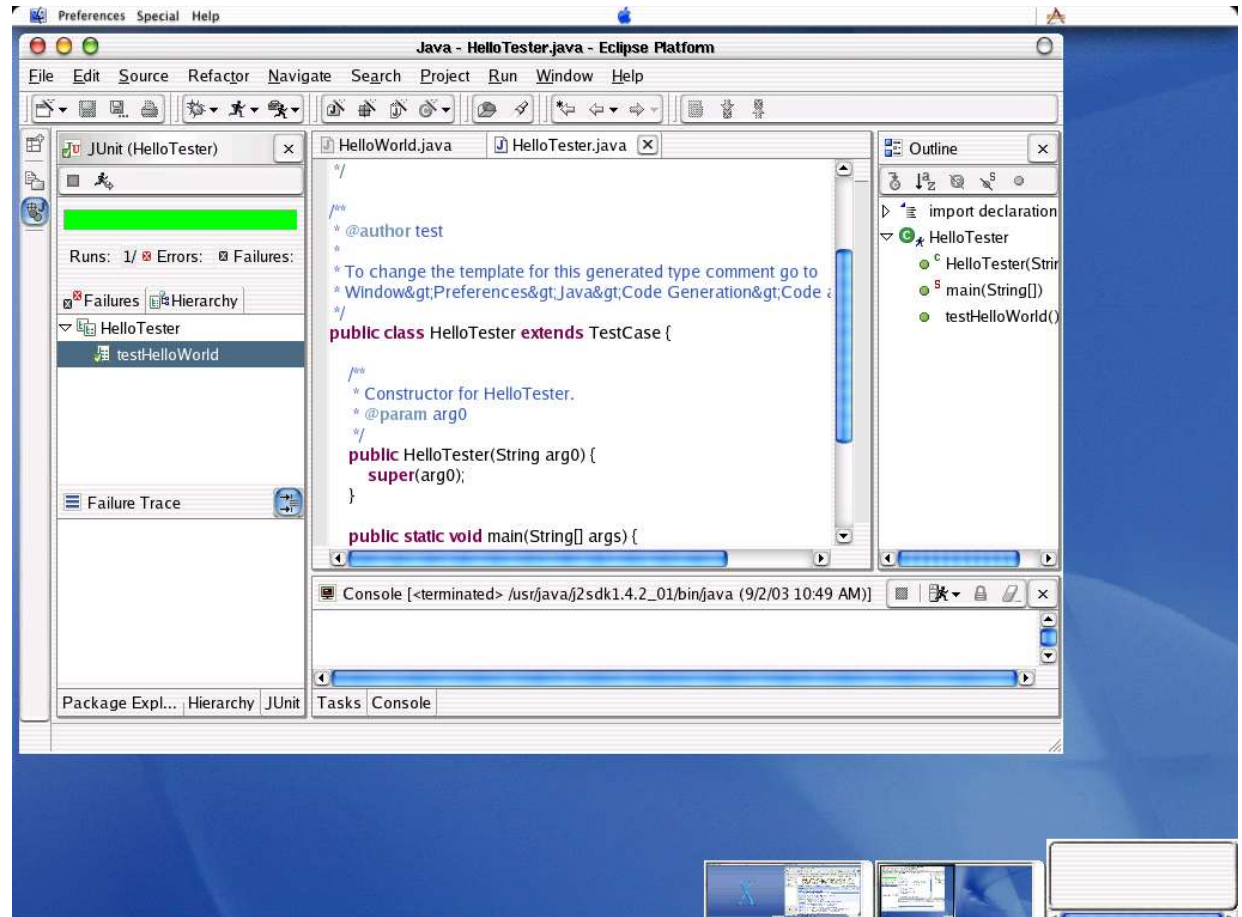
Gambar 4NetBeans 3.5 Splash Screen



Gambar 5Eclipse 3.0 Splash Screen



Gambar 6NetBeans di Windows dengan skin I2fprod aqua theme



Pada file ini kita akan membahas netbeans dulu eclipse saya taruh di file yang berbeda mungkin suatu saat yang berkaitan dengan IDE akan menjadi file tersendiri :). bagaimana menurut anda?

A.Konsep – konsep penting

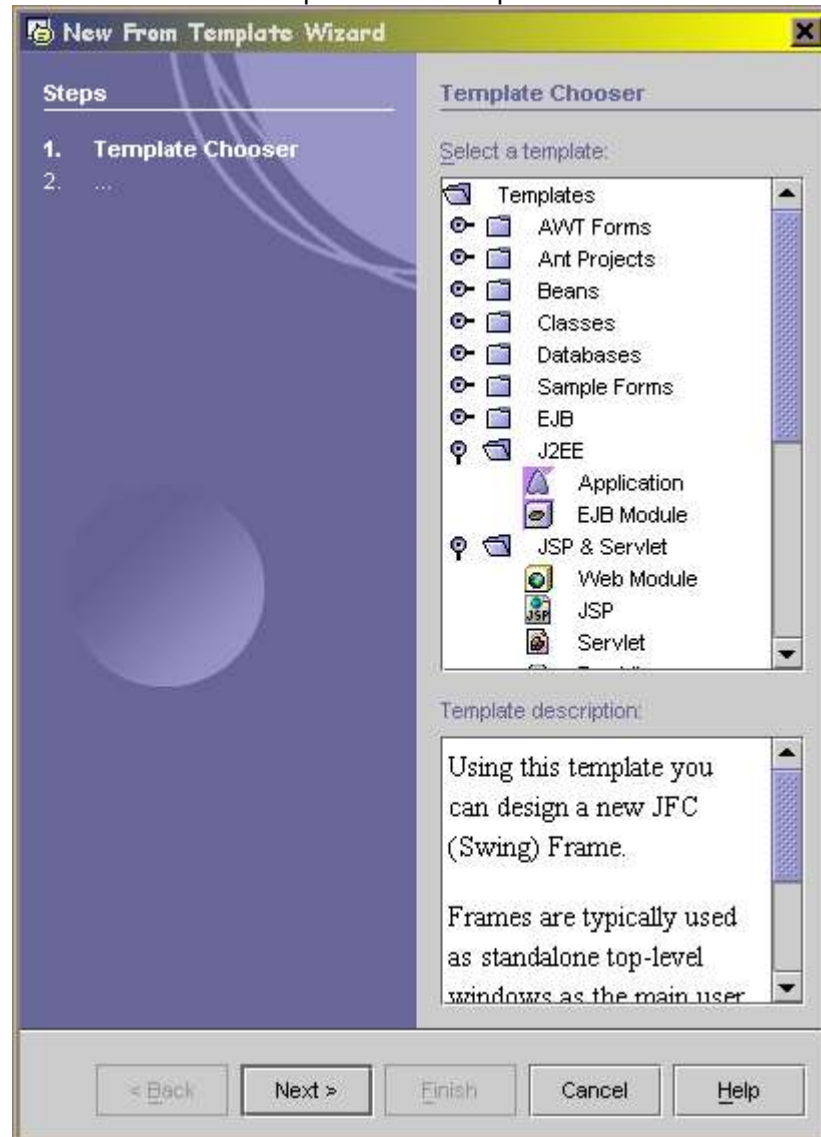
Bekerja dalam IDE membuat kita harus membiasakan diri dengan konsep-konsep di bawah ini:

- Templates
- Filesystems
- Modules
- Projects

1. Templates

Template menentukan penampakan dan perilaku awal dari objek. Untuk menciptakan objek baru kita tinggal menggunakan template yang sudah ada sehingga menghemat waktu dan usaha untuk membuatnya sendiri. Komponen-komponen Java, seperti kontainer Swing dan AWT disediakan sebagai template standar. Template juga tersedia untuk applet, class, dialog box, file HTML, file text, dan bookmark. Apabila kita memasang Enterprise IDE Toolkit, maka kita juga akan mempunyai template untuk Enterprise Java Beans. Atau kita memasang plugin untuk wireless development seperti Siemens ToolKit SL45i maka kita bisa memakai template untuk Midlet.

Untuk membuka Template Chooser pilih `File>New` atau klik icon `New`.

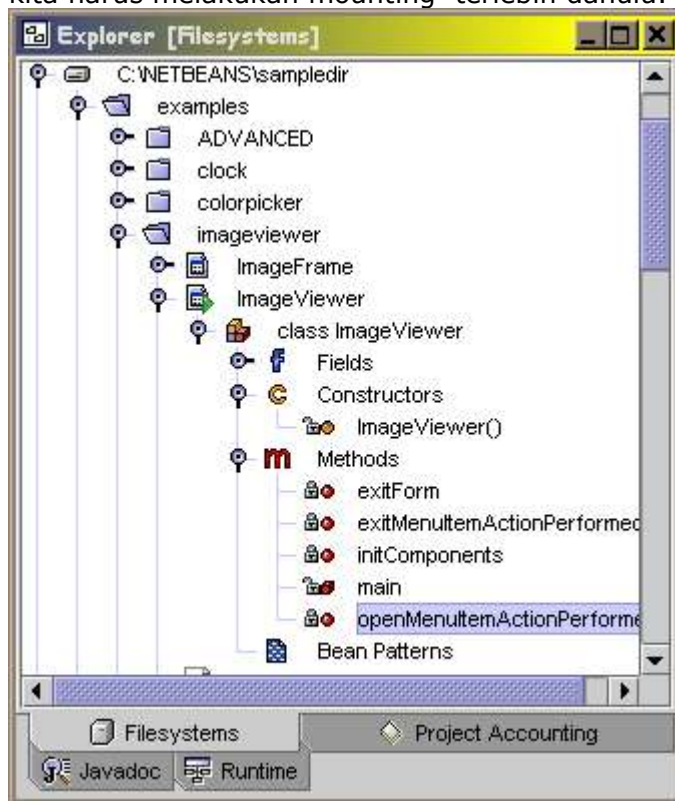


Gambar 8 Template Chooser

2. Filesystems

Dalam IDE, panel Filesystem menunjukkan organisasi hirarkis dari Java classpath dan direktori-direktorinya. Dengan mengikuti setiap node sampai ujungnya dalam jendela Explorer, kita dapat melihat banyak objek, class, method, atau file yang berada dalam

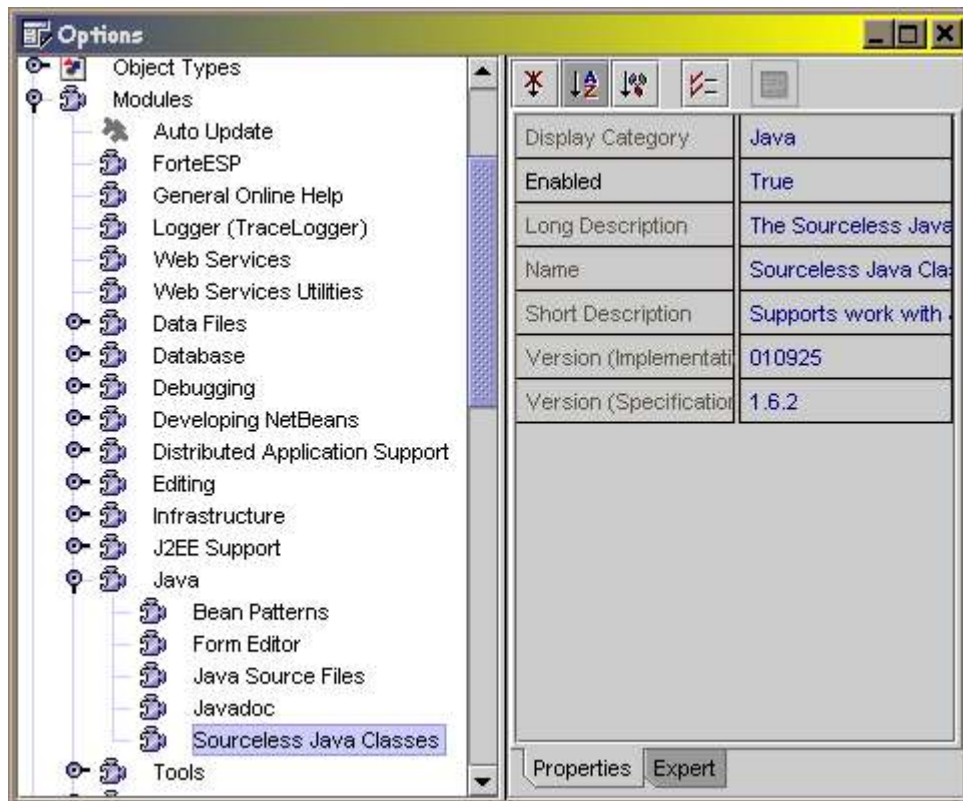
classpath. Jika kita mau bekerja dengan file-file yang belum tampak di Filesystem maka kita harus melakukan mounting terlebih dahulu.



Gambar 9 Jendela Eksplorer

3. Modules

Netbeans IDE seluruhnya dibangun dari modul-modul. Bahkan fitur-fitur utama seperti Source Editor, Debugger, dan Form Editor adalah modul. Kita dapat menambahkan modul sendiri jika kita mau. Untuk melihat modul apa saja yang sekarang terpasang pilih `Tools>Options`. Buka node modul dan klik kanan lalu pilih properties untuk melihat maupun merubah sifat dari modul.



Gambar 10 Options, untuk merubah setting IDE

4. Projects

Jika kita bekerja dengan IDE maka kita bisa mengorganisasikan aplikasi-aplikasi yang akan kita buat dalam projects. Kita bisa melakukan operasi secara keseluruhan dalam satu project. Misalnya kita bisa mengkompilasi atau membangun seluruh file yang ada dalam satu project yang mungkin terdiri dari banyak struktur direktori. Project dibuat dan dikelola dengan menu Project. Untuk melihat file-file pada project yang aktif, klik tab Project pada jendela Explorer.

Untuk setiap project, kita dapat mengatur hal-hal seperti, jenis kompiler dan debugger yang digunakan melalui `Project>Settings`.



Gambar 11 Project Manager



Gambar 12 Project Setting

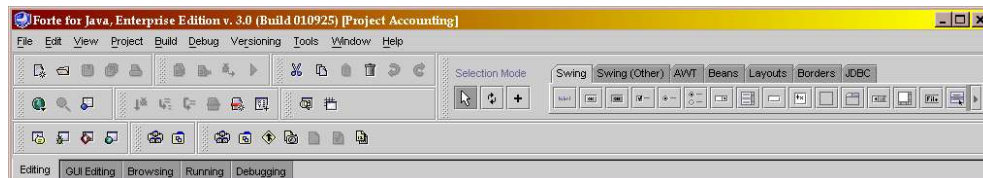
B. Berkeliling dalam IDE

Ketika pertama kali memulai beberapa hal terjadi bersamaan:

- IDE berada pada editing workspace
- Jendela Utama terbuka
- Jendela Explorer and Properties terbuka

1. Jendela Utama

Jendela pada atas layar disebut jendela utama yang merupakan pusat perintah pada IDE. Di sini kita temukan menu bar, beberapa toolbar, component palette, dan tab-tab workspace (Editing, GUI Editing, Browsing, Running, dan Debugging). Dari menu kita dapat mengakses ke semua jendela seperti Source Editor, jendela Properties, jendela debugger, jendela Options, dan jendela Project Settings.



Gambar 13 Jendela Utama IDE

a) Menus dan Toolbars

Pengelompokan menu pada toolbar adalah sebagai berikut:

System

Perintah-perintah untuk membuka template baru, buka simpan tutup file-file, dan membuka jendela Object Browser dan Explorer.

Edit

Perintah-perintah untuk mengedit kode sumber

Data

Perintah-perintah untuk mencari atau melihat informasi

Build

Perintah-perintah untuk mengkompilasi atau menyusun

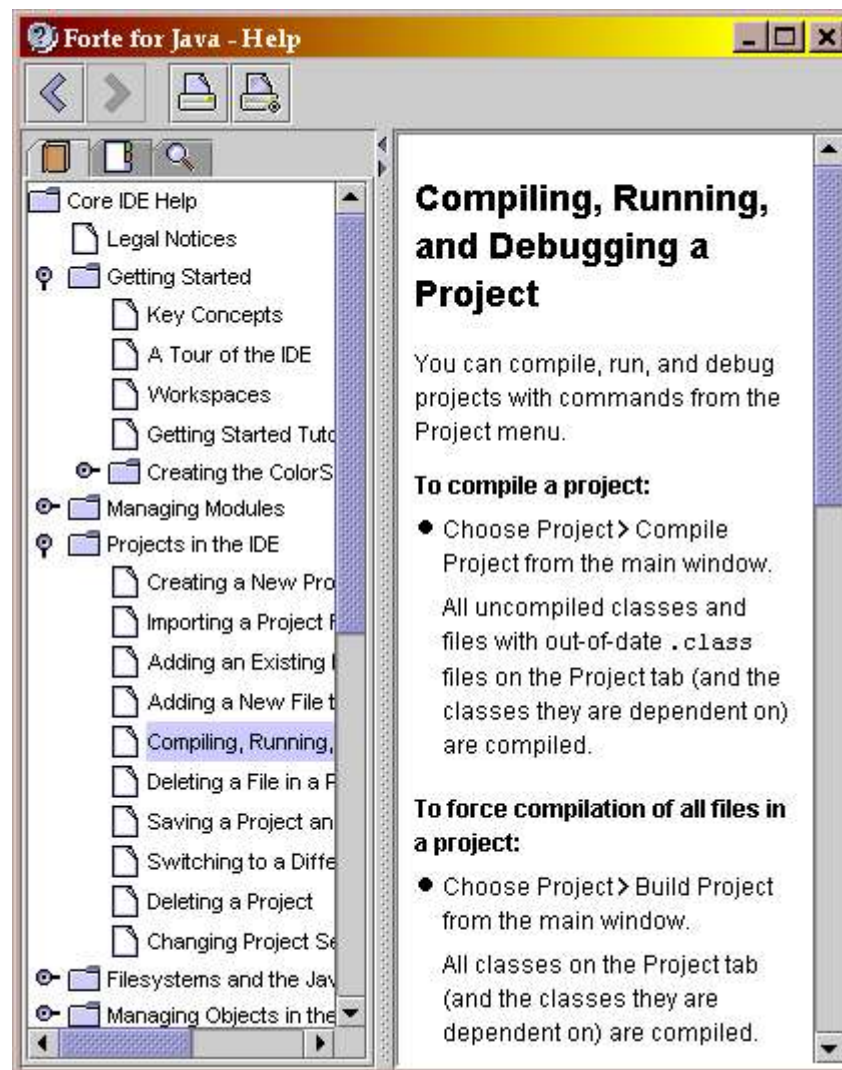
Debug

Perintah-perintah untuk menetapkan breakpoint, menambah pengawas, memperoleh trace information, dan perintah-perintah debugging lainnya

Form

Perintah-perintah untuk membuka Component Inspector, menampilkan grid pada Form Editor, dan berpindah dari mode rancang ke mode tes.

Help



Gambar 14 Jendela Help

Pada sisi kiri dari setiap toolbar dan component palette adalah drag bar untuk mengatur posisi dari toolbar dan component palette.

Component Palette

Pada sebelah kanan dari jendela utama kita dapat melihat banyak tab untuk berbagai macam komponen AWT, Swing, and JavaBeans, bersama pilihan layout manager dan border.

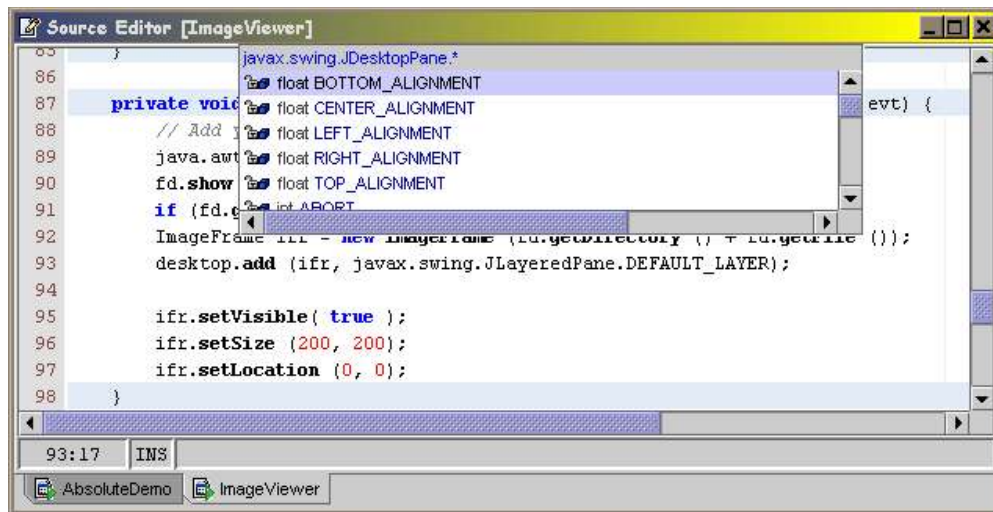
2.Workspaces

Sebuah workspace adalah kumpulan dari jendela-jendela yang saling berhubungan erat untuk melakukan tugas-tugas tertentu. Dari jendela utama kita dapat membuka workspace default, yaitu: Editing, GUI Editing, Browsing, Running, and Debugging. Pertama kali IDE dijalankan, Editing workspace membuka. Kita dapat mengkonfigurasi workspace sesuai keperluan.

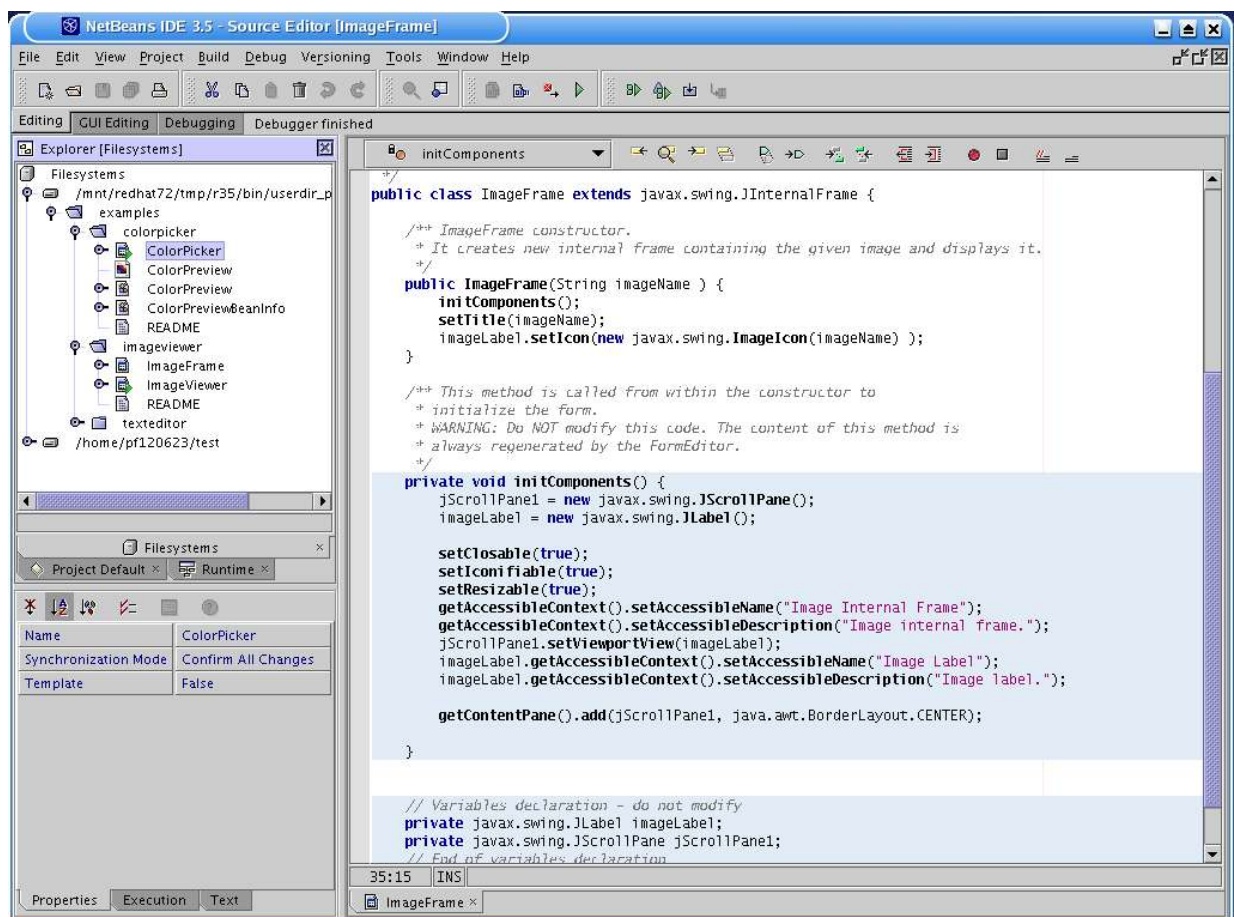
Editing

Terdiri dari jendela Explorer dan Properties. Saat kita membuka sebuah file, Source Editor secara otomatis terbuka. Source Editor digunakan untuk mengedit file-file Java, HTML, dan plain text. Kode sumber diwarnai secara sintaksis ---keyword default contohnya diberi

warna biru. Source Editor mendukung *dynamic code completion*; yaitu, kita dapat mengetikkan beberapa karakter pertama dari ekspresi dan melihat daftar class, methods dan variable yang dapat digunakan untuk menyelesaikan ekspresi tersebut

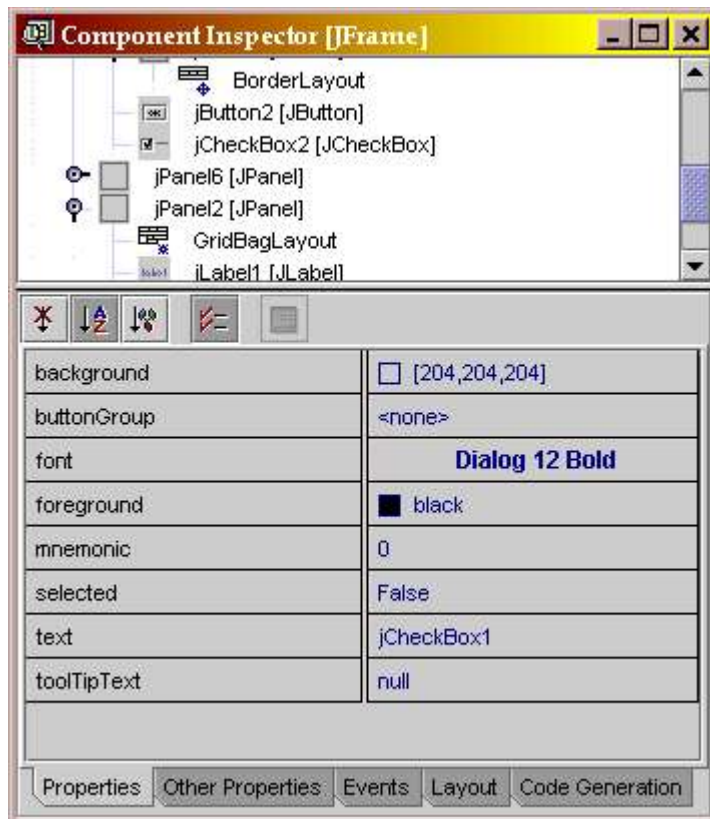


Gambar 15 Source Editor

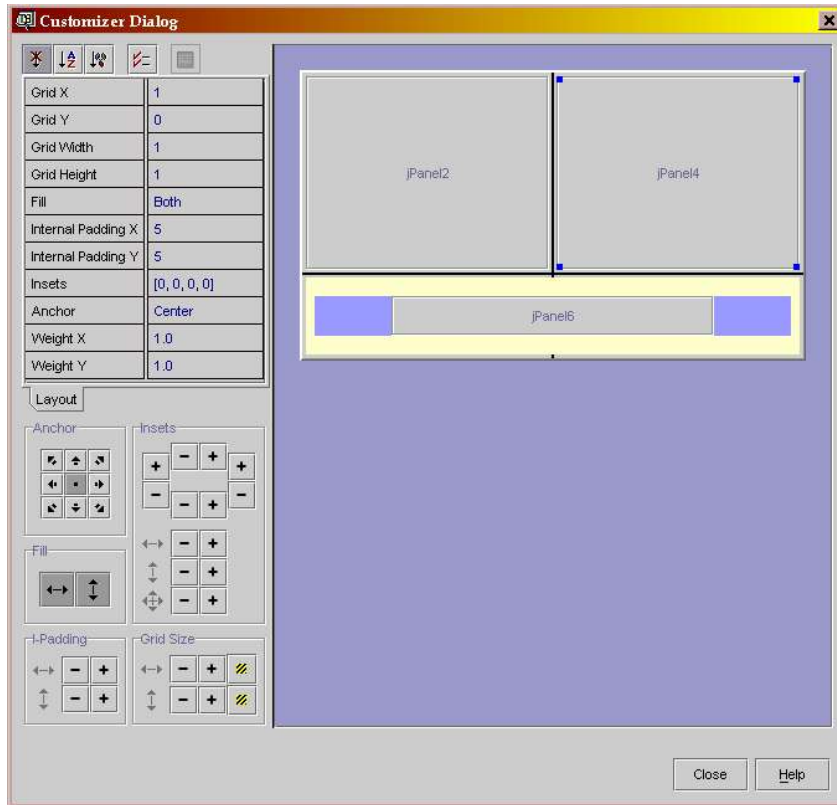


3.GUI Editing

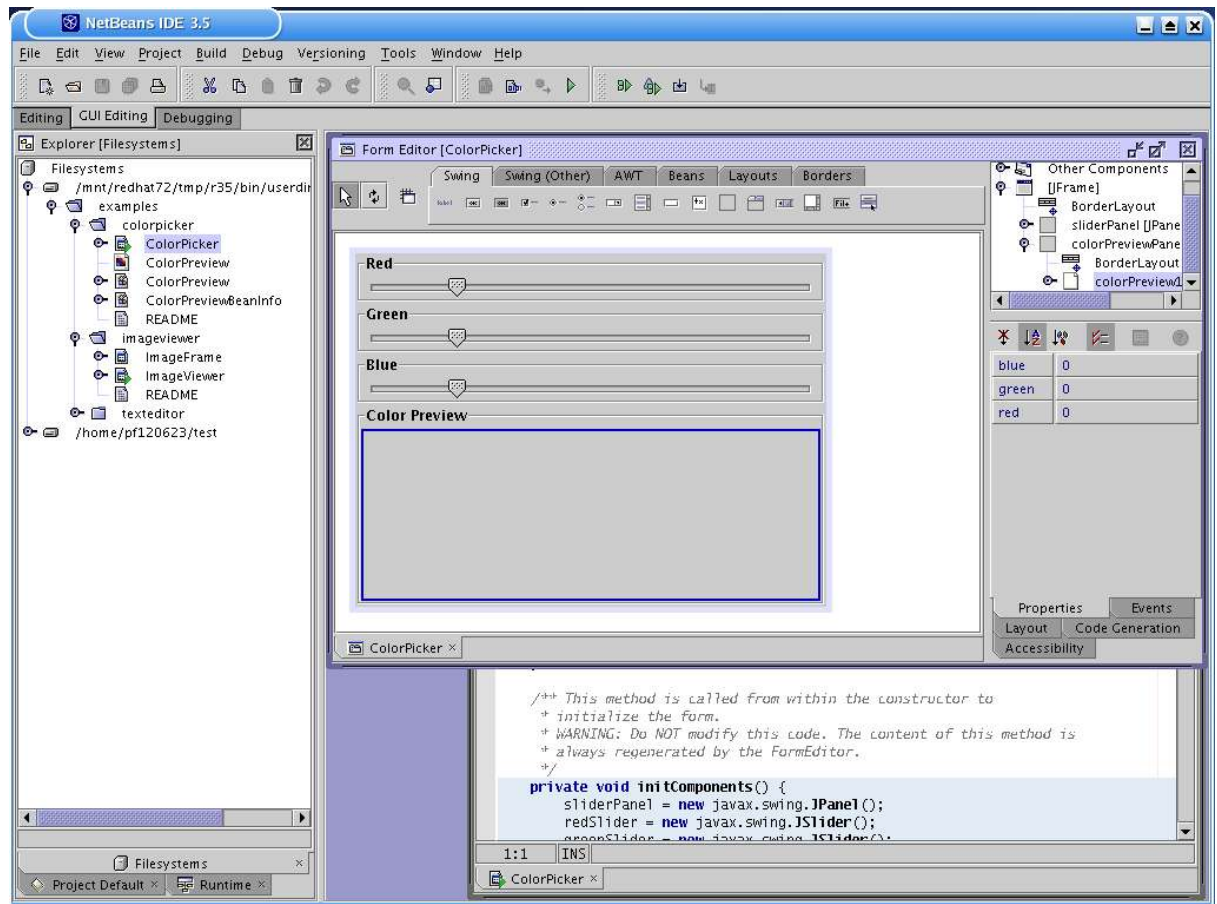
Di dalamnya akan kita dapatkan Explorer, Form Editor, Component Inspector, dan Source Editor. Kita menggunakan workspace GUI Editing untuk merancang dan membangun GUI (graphical user interfaces). Component Inspector memungkinkan kita untuk melihat komponen-komponen apa saja baik yang tampak (visual component) maupun yang tidak tampak (non-visual component seperti Beans dan Database Access). Selanjutnya kita juga bisa mengubah properti dari komponen-komponen tersebut di dalam Component Inspector. Jendela Form Editor merupakan daerah utama untuk membuat dan memodifikasi sebuah GUI. Kode yang dihasilkan oleh Form Editor ditampilkan dengan latar belakang yang berbeda dan tidak dapat diedit secara manual. Jika kita membuka jendela Form Editor pada workspace yang lain, IDE secara otomatis berpindah ke GUI Editing workspace.



Gambar 16 Component Inspector

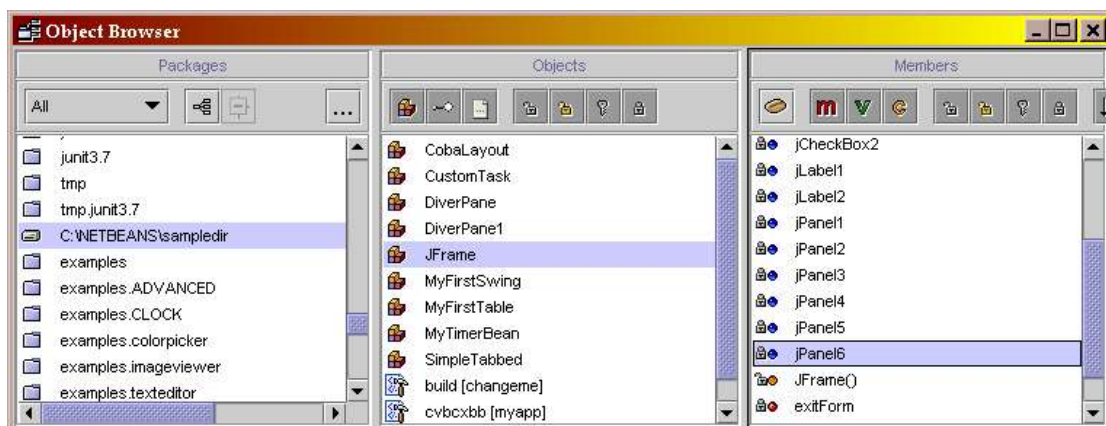


Gambar 17 GridBag Customizer

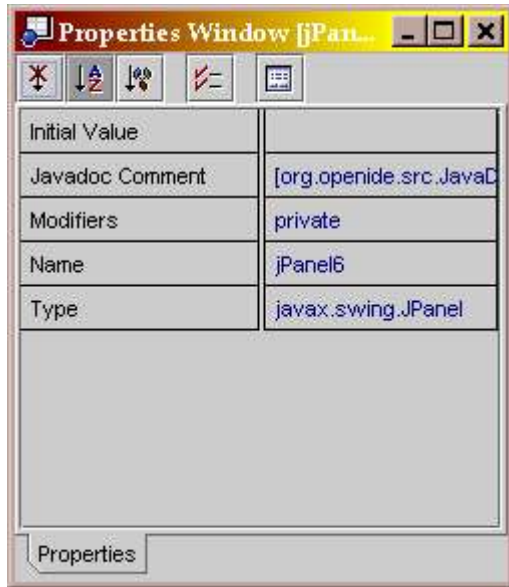


4. Browsing

Di dalamnya terdapat Object Browser (terdiri atas tiga panel) dan jendela Properties. Dengan Object Browser kita dapat melihat hirarki dari package, objek-objek (class and interface), dan anggota-anggotanya (method, variable, dan constructor) dalam program anda. Kita dapat membuka kode sumber dari Object Browser dengan mengklik-ganda sebuah nama baik pada panel Objects maupun panel Members. Jendela Properties memungkinkan kita untuk melihat dan mengedit sifat-sifat dari objek yang kita pilih dalam Object Browser.



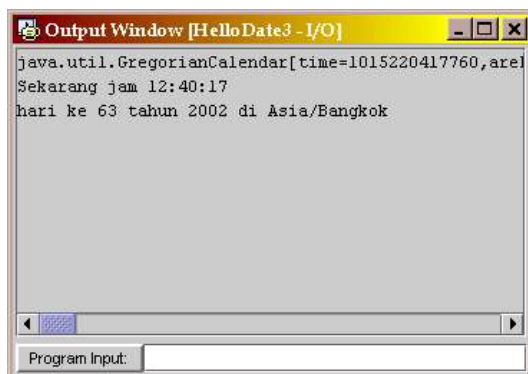
Gambar 18 Object Browser



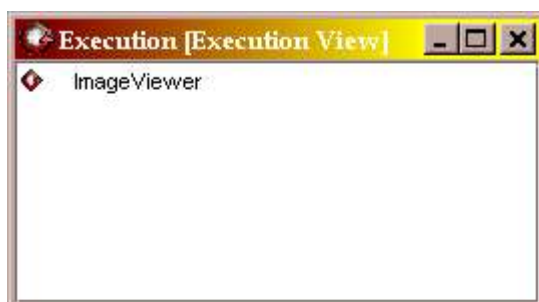
Gambar 19 Jendela Properties

5. Running

Didalamnya terdapat jendela Execution View dan Output. Begitu kita mengeksekusi program, maka IDE akan berpindah secara otomatis ke Running workspace. Jika tidak ada kesalahan dalam eksekusi, aplikasi akan diluncurkan sehingga kita dapat mentesnya. Kesalahan yang mungkin terjadi akan ditampilkan dalam jendela Output.



Gambar 20 Jendela Keluaran

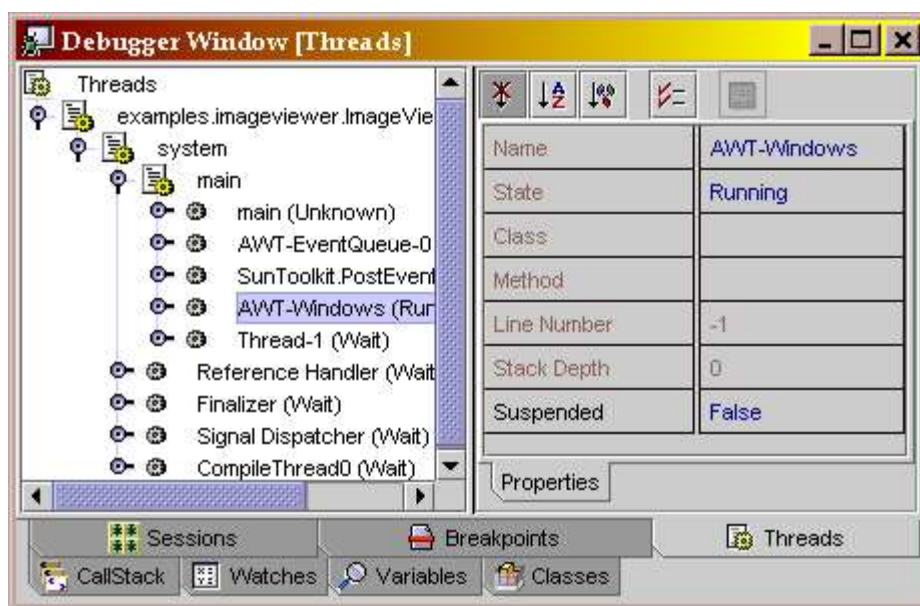


Gambar 21 Melihat program apa yang sedang berjalan

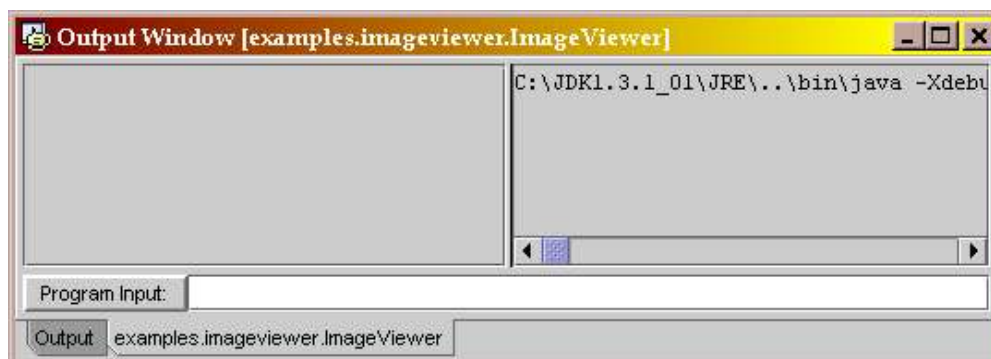
6.Debugging

Terdiri dari jendela Debugger dan Output. Jendela Debugger mempunyai tab panel untuk menset breakpoint, memonitor thread, dan mengawasi nilai variabel-variabel. Jendela Output menampilkan pesan dari debugger. Jika ada file yang terbuka, Debugging workspace juga mengandung Source Editor, yang menandai breakpoints dengan warna magenta.

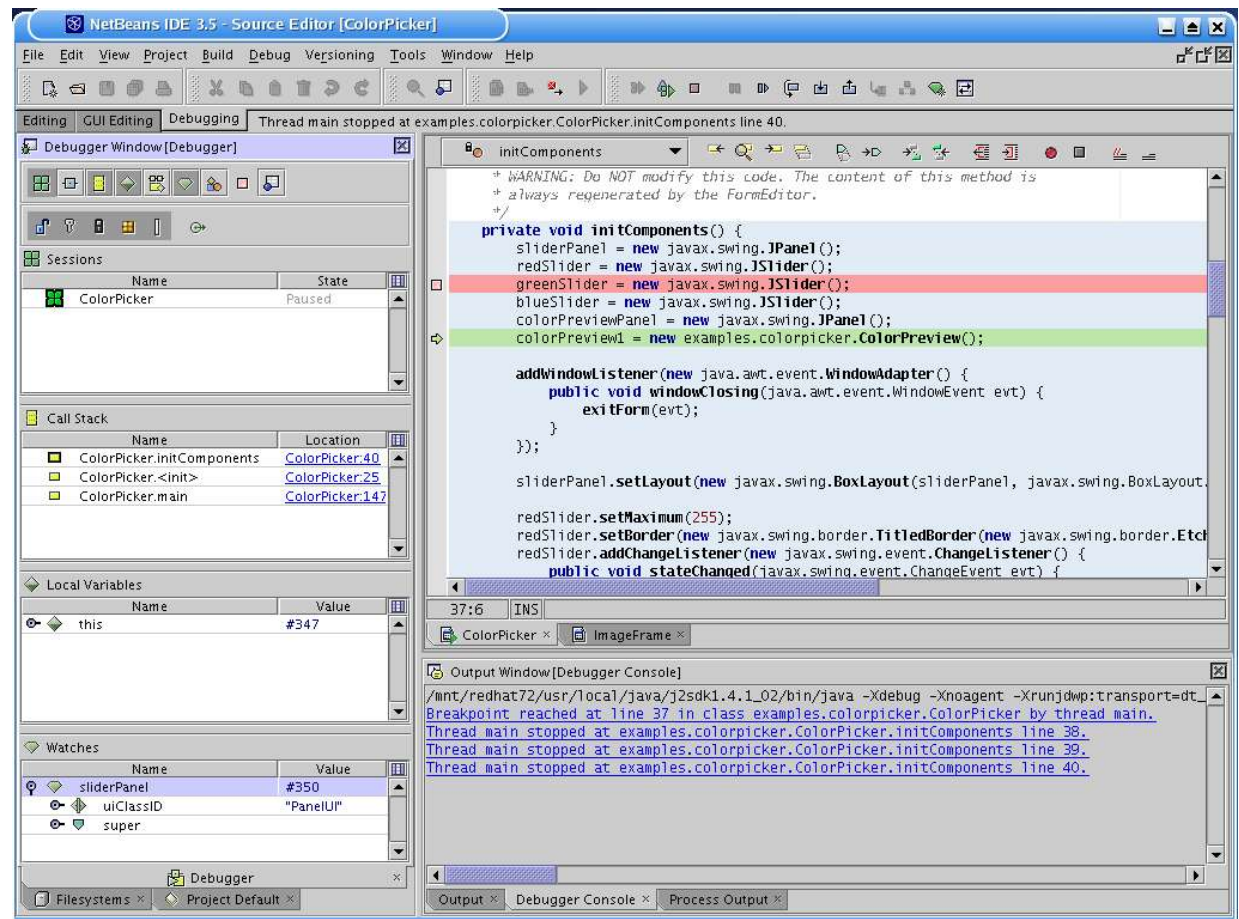
Workspace yang sekarang digunakan tidak membatasi berapa jumlah atau jenis jendela yang dapat kita buka. kita dapat menggunakan menu View pada jendela utama untuk membuka jendela apa saja kapan saja. Ketika kita keluar dari IDE, keadaan terakhir dari setiap workspace disimpan. Kali berikutnya kita menggunakan IDE, jendela-jendela pada workspace akan muncul seperti terakhir kali kita meninggalkannya.



Gambar 22 Mencari kesalahan dengan debugger



Gambar 23 Jendela keluaran dari Debugger



Bab III.Core Java at Minimum

Setelah kita mengenal bahasa Java sekarang kita akan menggali lebih dalam lagi tentang Java API sebagai interface untuk mengakses sumber daya atau layanan dari sistem operasi.

A.Data teks

Pada bagian sebelumnya kita telah mengenal class-class yang berhubungan dengan waktu. Sekarang kita akan mempelajari tentang `Character` dan `String`

Platform Java mengandung tiga class yang berguna dalam bekerja dengan data teks

1. `Character`, yang digunakan untuk menyimpan nilai dari satu karakter. Class ini juga berisi method-method untuk memanipulasi ataupun memeriksa nilai sebuah karakter tunggal
2. `String`, biasa digunakan untuk data immutable (tidak berubah sejak pertama kali ia dibuat) yang terdiri atas banyak karakter. Objek `String` bisa diisi dengan nilai null dan tidak akan bermasalah jika kita coba menampilkan hasilnya
3. `StringBuffer` merupakan class untuk menyimpan dan memanipulasi data yang dapat berubah yang terdiri atas banyak karakter.

1.Character

Objek `character` diperlukan (bukannya tipe primitif `char`) pada saat kita ingin melewati suatu nilai karakter ke dalam method yang akan mengubah nilai tersebut, atau kalau kita ingin menempatkannya pada suatu struktur data yang memerlukan tipe objek. Contoh:

```
package learn.core;
public class CharacterDemo {
    public static void main(String args[]) {
        Character a = new Character('a');
        Character a2 = new Character('a');
        Character b = new Character('b');

        int difference = a.compareTo(b);

        if (difference == 0) {
            System.out.println("a is equal to b.");
        } else if (difference < 0) {
            System.out.println("a is less than b.");
        } else if (difference > 0) {
            System.out.println("a is greater than b.");
        }
        System.out.println("a is " + ((a.equals(a2)) ? "equal" : "not equal")
            + " to a2.");
        System.out.println("The character " + a.toString() + " is "
            + (Character.isUpperCase(a.charValue()) ? "upper" : "lower") + " case.");
    }
}
```

keluarannya adalah sebagai berikut:

```
a is less than b.
a is equal to a2.
The character a is lowercase.
```

2.String

Untuk menciptakan objek `String` ada beberapa cara, yaitu:

- dengan langsung menggunakan string literal, contoh: `String str = "Hello World!"`;
- menggunakan konstruktor dengan parameter `String str = new String("Hello World!")`;

keduanya akan menghasilkan objek yang sama. Meskipun demikian, cara yang menggunakan literal string adalah lebih baik.

Cara yang lain adalah dengan konstruktor yang menggunakan array seperti contoh berikut:

```
package learn.core;
public class StringDemo {
    public static void main(String args[]) {
        char data[] = {'a', 'b', 'c'};
        String str = null;
        System.out.println(str);
        str = new String(data);
        System.out.println(data);
        System.out.println("abc");
        String cde = "cde";
        for (int i = 0; i < cde.length(); ++i) {
            System.out.println(cde.charAt(i));
        }
        System.out.println("abc" + cde);
        String c = "abc".substring(2,3);
        System.out.println(c);
        String d = cde.substring(0, 2);
        System.out.println(d);
        String newCDE = new String("CDE");
        System.out.println(cde.equals(newCDE));
        System.out.println(cde.equalsIgnoreCase(newCDE));
    }
}
```

Output:

```
null
abc
abc
c
d
e
abccde
c
cd
false
true
```

3.StringBuffer

`StringBuffer` adalah dekorator objek `String` yang mempunyai nilai sekaligus kapasitas penyangga untuk perubahan nilai.

Konstruktor dari `StringBuffer`:

```
public StringBuffer() // initial value "", initial capacity 16
public StringBuffer(int length) // init value "", capacity length
public StringBuffer(String str) // initial value same as str, initial capacity str.length()+16
```

Beberapa method yang sering digunakan:

```
public StringBuffer append(String str)
public StringBuffer append(char[] chars)
public StringBuffer insert(int i, String str)
public StringBuffer insertIint i, char[] chars)
public void setCahrAt(int i, char character)
```

Contoh:

```
StringBuffer strBuffer = new StringBuffer("wirto");
```

```
String insertString = "na";
strBuffer.insert(2, insertString); // strBuffer menjadi winarto
StringBuffer strBuff = new StringBuffer("itb");
char character = 'M';
strBuff.setCharAt(1,character); // strBuff menjadi iMb
```

Contoh penggunaan:

```
package learn.core;
public class StringBufferDemo {
    private static void print(StringBuffer b) {
        System.out.println("nilai b sekarang: " + b);
    }
    public static void main(String args[]) {
        StringBuffer b = new StringBuffer("Mow");
        print(b);
        char c = b.charAt(0);
        System.out.println("c: " + c);
        b.setCharAt(0, 'N'); //tidak bisa kalau String
        print(b);
        b.append(' '); //ditambah spasi kosong
        print(b);
        b.append("is the time.");
        print(b);
        b.append(23); //ditambah integer
        print(b);
        b.insert(6, "n't");
        print(b);
        b.replace(4, 9, "is");
        print(b);
        b.delete(16, 18);
        print(b);
        b.deleteCharAt(2);
        print(b);
        b.setLength(5);
        print(b);
        b.reverse();
        print(b);
        String s = b.toString();
        System.out.println("s: " + s); //dikembalikan menjadi immutable String
        s = b.substring(1,2);
        System.out.println("s: " + s);
        b.setLength(0); // menghapus buffer
        print(b);
    }
}
```

Keluarannya adalah:

```
nilai b sekarang: Mow
c: M
nilai b sekarang: Now
nilai b sekarang: Now
nilai b sekarang: Now is the time.
nilai b sekarang: Now is the time.23
nilai b sekarang: Now isn't the time.23
nilai b sekarang: Now is the time.23
nilai b sekarang: Now is the time.
nilai b sekarang: No is the time.
nilai b sekarang: No is
nilai b sekarang: si oN
s: si oN
s: i
nilai b sekarang:
```

Sekarang kita akan melihat sebuah program yang menggunakan String dan StringBuffer secara ekstensif:

```
package learn.core;
public class Palindrome {
    public static boolean isPalindrome(String stringToTest) {
        String workingCopy = removeJunk(stringToTest);
        String reversedCopy = reverse(workingCopy);

        return reversedCopy.equalsIgnoreCase(workingCopy);
    }
}
```

```
    }

    protected static String removeJunk(String string) {
        int i, len = string.length();
        StringBuffer dest = new StringBuffer(len);
        char c;

        for (i = (len - 1); i >= 0; i--) {
            c = string.charAt(i);
            if (Character.isLetterOrDigit(c)) {
                dest.append(c);
            }
        }

        return dest.toString();
    }

    protected static String reverse(String string) {
        StringBuffer sb = new StringBuffer(string);

        return sb.reverse().toString();
    }

    public static void main(String[] args) {
        String string = "Madam, I'm Adam.";

        System.out.println();
        System.out.println("Testing whether the following "
            + "string is a palindrome:");
        System.out.println("    " + string);
        System.out.println();

        if (isPalindrome(string)) {
            System.out.println("It IS a palindrome!");
        } else {
            System.out.println("It is NOT a palindrome!");
        }
        System.out.println();
    }
}
```

Output:

```
Testing whether the following string is a palindrome:
    Madam, I'm Adam.
```

```
It IS a palindrome!
```

Contoh yang lain lagi:

```
package learn.core;
public class Anagram {

    /**
     * Tests whether the passed-in strings are anagrams --
     * containing the exact same number of each letter.
     * Punctuation, case, and (obviously) order don't matter.
     *
     * @return true if the strings are anagrams; otherwise, false
     */
    public static boolean areAnagrams(String string1,
        String string2) {

        String workingCopy1 = removeJunk(string1);
        String workingCopy2 = removeJunk(string2);

        workingCopy1 = workingCopy1.toLowerCase();
        workingCopy2 = workingCopy2.toLowerCase();

        workingCopy1 = sort(workingCopy1);
        workingCopy2 = sort(workingCopy2);

        return workingCopy1.equals(workingCopy2);
    }

    /**
     * Removes punctuation, spaces -- everything except letters
     * and digits from the passed-in string.
     */
}
```

```
*
* @return a stripped copy of the passed-in string
*/
protected static String removeJunk(String string) {
    int i, len = string.length();
    StringBuffer dest = new StringBuffer(len);
    char c;

    for (i = (len - 1); i >= 0; i--) {
        c = string.charAt(i);
        if (Character.isLetterOrDigit(c)) {
            dest.append(c);
        }
    }

    return dest.toString();
}

/**
 * Sorts the passed-in string. Reimplement this method
 * if you want to use this class in pre-Java-2 versions
 * of the platform.
 *
 * @return a sorted copy of the passed-in string
 */
protected static String sort(String string) {
    int length = string.length();
    char[] charArray = new char[length];

    string.getChars(0, length, charArray, 0);

    //NOTE: The following line of code causes pre-1.2
    //compilers to choke.
    java.util.Arrays.sort(charArray);

    return new String(charArray);
}

public static void main(String[] args) {
    String string1 = "Cosmo and Laine:";
    String string2 = "Maid, clean soon!";

    System.out.println();
    System.out.println("Testing whether the following "
        + "strings are anagrams:");
    System.out.println("    String 1: " + string1);
    System.out.println("    String 2: " + string2);
    System.out.println();

    if (areAnagrams(string1, string2)) {
        System.out.println("They ARE anagrams!");
    } else {
        System.out.println("They are NOT anagrams!");
    }
    System.out.println();
}
}
```

Outputnya:

```
Testing whether the following strings are anagrams:
String 1: Cosmo and Laine:
String 2: Maid, clean soon!

They ARE anagrams!
```

a)Konversi dari String ke bilangan

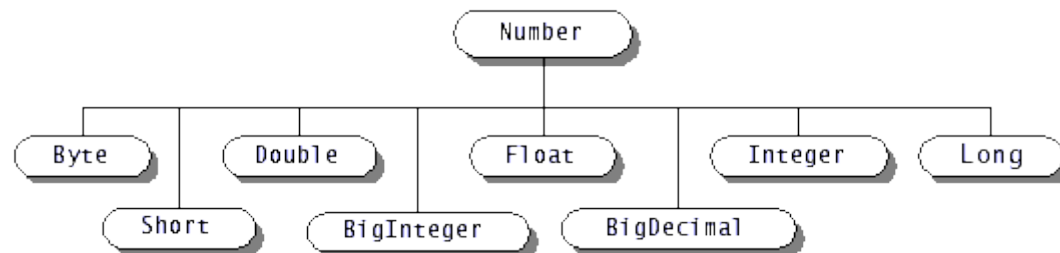
Untuk mengkonversi dari String ke sebuah bilangan bisa digunakan method valueOf dari wrapper class seperti Integer, Double, Float dan Long. Contohnya:

```
String piStr = "3.14159";
Float pi = Float.valueOf(piStr);
```


B.Bilangan dan Matematika

1.Angka-angka

Setelah mempelajari cara menangani teks, kita akan belajar bagaimana menyimpan nilai suatu bilangan dan merepresentasikannya dalam bentuk yang kita inginkan. Class `java.lang.Number` digunakan untuk menyimpan nilai bilangan. Ia mempunyai hirarki seperti gambar di bawah ini. Terlihat bahwa selain dari tipe-tipe primitif, Java juga menyediakan objek wrapper dari tipe-tipe primitifnya.



```

Math.PI           // 3.14159265358979323846
Math.E           // 2.7182818284590452354
  
```

a)

b)Mengkonversi Number dari dan ke String

Sebuah program Java yang bekerja pada bilangan harus mendapat masukan dari suatu tempat. Seringkali program harus membaca representasi tekstual dari sebuah bilangan, sehingga harus dikonversi terlebih dahulu.

Contoh method-method untuk mengkonversi String ke `Number`:

```

String s = "-42";
byte b = Byte.parseByte(s);
short sh = Short.parseShort(s);
int i = Integer.parseInt(s);
long l = Long.parseLong(s);
float f = Float.parseFloat(s);
double d = Double.parseDouble(s);

byte b = Byte.parseByte("1011", 2); // 1011 in binary is 11 in decimal
short sh = Short.parseShort("ff", 16); // ff in base 16 is 255 in decimal

int i = Integer.valueOf("egg", 17).intValue(); // Base 17!
  
```

Contoh program yang memanfaatkan method `decode()` dan method balikkannya pada class `Integer`

```

package learn.core;
public class NumberDemo {
    public static void main(String args[]) {
        // The decode() method handles octal, decimal, or hexadecimal, depending
        // on the numeric prefix of the string
        short sh = Short.decode("0377").shortValue(); // Leading 0 means base 8
        int i = Integer.decode("0xff").intValue(); // Leading 0x means base 16
        long l = Long.decode("255").longValue(); // Other numbers mean base 10

        System.out.println("0377 basis 8 berarti: " + sh + " dalam basis 10");
        System.out.println("0xff basis 16 berarti: " + i + " dalam basis 10");
        System.out.println("255 basis 10 berarti: " + l + " dalam basis 10");

        // Integer class can convert numbers to strings
        String decimal = Integer.toString(42);
        String binary = Integer.toBinaryString(42);
    }
}
  
```

```
String octal = Integer.toOctalString(42);
String hex = Integer.toHexString(42);
String base36 = Integer.toString(42, 36);

System.out.println("42 dalam bentuk decimal: " + decimal);
System.out.println("42 dalam bentuk biner: " + binary);
System.out.println("42 dalam bentuk octal: " + octal);
System.out.println("42 dalam bentuk hexa: " + hex);
System.out.println("42 dalam bentuk base36: " + base36);
}
}
```

Keluarannya adalah:

```
0377 basis 8 berarti: 255 dalam basis 10
0xff basis 16 berarti: 255 dalam basis 10
255 basis 10 berarti: 255 dalam basis 10
42 dalam bentuk decimal: 42
42 dalam bentuk biner: 101010
42 dalam bentuk octal: 52
42 dalam bentuk hexa: 2a
42 dalam bentuk base36: 16
```

c)Memformat angka-angka

Bilangan biasanya dicetak atau ditampilkan berbeda pada negara-negara yang lain. Contohnya, di Indonesia dan Eropa berlaku tanda titik untuk pemisah ribuan dan tanda koma digunakan untuk memisahkan angka desimal sedangkan di banyak negara lain berlaku sebaliknya. Perbedaan bentuk ini dapat berkembang lebih jauh lagi saat menampilkan bilangan yang merepresentasikan nilai uang. Untuk mengkonversi bilangan ke String untuk ditampilkan, akan lebih baik bila kita gunakan class

`java.text.NumberFormat` untuk mengkonversi secara setting/locale tertentu.

Program dibawah ini menampilkan bilangan dalam bentuk tampilan yang berbeda, yaitu bilangan biasa (Integer dan Double), nilai mata uang dan persen. Perhatikan keluarannya dan lihat bahwa default Locale/setting sudah menunjukkan spesifik bahasa dan negara Indonesia mengikuti setting sistem operasi yang kita miliki, hanya saja formatnya belum benar, kelak ini akan kita sempurnakan.

```
package learn.core;
import java.util.*;
import java.text.*;

public class NumberFormatDemo {
    static public void displayNumber(Locale currentLocale) {
        Integer quantity = new Integer(123456);
        Double amount = new Double(345987.246);
        NumberFormat numberFormatter;
        String quantityOut;
        String amountOut;

        numberFormatter = NumberFormat.getNumberInstance(currentLocale);
        quantityOut = numberFormatter.format(quantity);
        amountOut = numberFormatter.format(amount);
        System.out.println(quantityOut + " " + currentLocale.toString());
        System.out.println(amountOut + " " + currentLocale.toString());
    }
    static public void displayCurrency(Locale currentLocale) {
        Double currency = new Double(9876543.21);
        NumberFormat currencyFormatter;
        String currencyOut;

        currencyFormatter = NumberFormat.getCurrencyInstance(currentLocale);
        currencyOut = currencyFormatter.format(currency);
        System.out.println(currencyOut + " " + currentLocale.toString());
    }
    static public void displayPercent(Locale currentLocale) {
        Double percent = new Double(0.75);
        NumberFormat percentFormatter;
        String percentOut;
    }
}
```

```

        percentFormatter = NumberFormat.getPercentInstance(currentLocale);
        percentOut = percentFormatter.format(percent);
        System.out.println(percentOut + " " + currentLocale.toString());
    }
    static public void main(String[] args) {
        Locale[] locales = {
            Locale.getDefault(),
            new Locale("fr","FR"),
            new Locale("de","DE"),
            new Locale("en","US")
        };
        for (int i = 0; i < locales.length; i++) {
            System.out.println();
            displayNumber(locales[i]);
            displayCurrency(locales[i]);
            displayPercent(locales[i]);
        }
    }
}

```

Keluarannya adalah:

```

123,456   in_ID
345,987.246   in_ID
¤ 9,876,543.21   in_ID
75%   in_ID

```

```

123 456   fr_FR
345 987,246   fr_FR
9 876 543,21 F   fr_FR
75%   fr_FR

```

```

123.456   de_DE
345.987,246   de_DE
9.876.543,21 DM   de_DE
75%   de_DE

```

```

123,456   en_US
345,987.246   en_US
$9,876,543.21   en_US
75%   en_US

```

Mengapa negara lain sudah mempunyai format tertentu di dalam Java yang mengikuti aturan kebiasaan di negaranya, sedangkan di negara kita tidak? Saya tidak tahu, tapi saya mengerti bagaimana ini terjadi. Hal ini disebabkan setting yang ada (available locales) tidak menyediakan ResourceBundle untuk Indonesia.

Sekarang coba anda lihat setting negara apa saja yang sudah ada.

```

package learn.core;
import java.util.*;
public class AvailableLocales {
    public static void main(String args[]) {
        Locale[] available = Locale.getAvailableLocales();
        for (int i = 0; i < available.length; i++){
            System.out.println(available[i]);
        }
    }
}

```

Kemudian periksa keluarannya, tidak ada in_ID bukan?

Sekarang kita akan mencoba membuat sebuah class yang akan mencetak bilangan dalam format Indonesia. Class ini hanya berisi method static yang membungkus method factory dari NumberFormat

```

package com.ginangjar.util;
import java.text.*;
import java.util.Locale;
public class IndonesianFormat {
    public static NumberFormat getNumberInstance(){
        NumberFormat nf = NumberFormat.getNumberInstance();
        DecimalFormat df = (DecimalFormat)nf;
        DecimalFormatSymbols idSymbols = df.getDecimalFormatSymbols();
        idSymbols.setGroupingSeparator('.');
    }
}

```

```

        idSymbols.setDecimalSeparator(',');
        df.setDecimalFormatSymbols(idSymbols);
        return nf;
    }
    public static NumberFormat getCurrencyInstance(){
        NumberFormat cf = NumberFormat.getCurrencyInstance();
        DecimalFormat df = (DecimalFormat)cf;
        DecimalFormatSymbols idSymbols = df.getDecimalFormatSymbols();
        idSymbols.setCurrencySymbol("Rp");
        idSymbols.setInternationalCurrencySymbol("IDR");
        idSymbols.setGroupingSeparator('.');
        idSymbols.setMonetaryDecimalSeparator(',');
        df.setDecimalFormatSymbols(idSymbols);
        df.setDecimalSeparatorAlwaysShown(true);
        return cf;
    }
    public static NumberFormat getPercentInstance(){
        return NumberFormat.getPercentInstance();
    }
    public static void main(String args[]) {
        Integer quantity = new Integer(123456);
        Double amount = new Double(345987.246);
        Double currency = new Double(9876543.2154);
        Double percent = new Double(0.75);

        NumberFormat numberFormatter = IndonesianFormat.getNumberInstance();
        NumberFormat currencyFormatter = IndonesianFormat.getCurrencyInstance();
        NumberFormat percentFormatter = IndonesianFormat.getPercentInstance();

        String quantityOut = numberFormatter.format(quantity);
        String amountOut = numberFormatter.format(amount);
        String currencyOut = currencyFormatter.format(currency);
        String percentOut = percentFormatter.format(percent);

        System.out.println(quantityOut + " " + Locale.getDefault());
        System.out.println(amountOut + " " + Locale.getDefault());
        System.out.println(currencyOut + " " + Locale.getDefault());
        System.out.println(percentOut + " " + Locale.getDefault());
    }
}

```

Kita lihat keluarannya:

```

123.456   in_ID
345.987,246   in_ID
Rp 9.876.543,22   in_ID
75%   in_ID

```

Tentu saja untuk mata uang lain, kita tidak perlu bersulit-sulit seperti ini ☺.

2.Fungsi-fungsi matematis

Class `java.lang.Math` mendefinisikan sejumlah method static untuk operasi-operasi trigonometrik, logaritmik, eksponensial, operasi pembulatan, dan lain-lain. Class ini hanya mempunyai static field dan static method, sehingga tidak diperlukan instansiasi untuk memanggilnya. Untuk fungsi-fungsi trigonometrik, sudut dinyatakan dalam radian. Fungsi logaritma dan eksponensia berdasarkab basis alami e , bukan dengan basis 10. Untuk fungsi-fungsi yang lebih ketat hasilnya bisa digunakan class `java.lang.StrictMath`. Beberapa contoh diantaranya:

```

double d = Math.toRadians(27);           // Convert 27 degrees to radians
d = Math.cos(d);                         // Take the cosine
d = Math.sqrt(d);                        // Take the square root
d = Math.log(d);                          // Take the natural logarithm
d = Math.exp(d);                          // Do the inverse: e to the power d
d = Math.pow(10, d);                      // Raise 10 to this power
d = Math.atan(d);                         // Compute the arc tangent
d = Math.toDegrees(d);                   // Convert back to degrees
double up = Math.ceil(d);                 // Round to ceiling

```

```
double down = Math.floor(d);           // Round to floor
long nearest = Math.round(d);         // Round to nearest
```

a) Bilangan acak (Random Numbers)

Class `Math` juga menyediakan method `random()` untuk membangkitkan bilangan pseudo-random, tapi untuk kegunaan yang lebih luwes dapat kita gunakan class `java.util.Random`. Jika kita membutuhkan bilangan yang sangat pseudo-random, gunakan saja class `java.security.SecureRandom`:

```
import java.util.Random;
import java.security.SecureRandom;
// A simple random number
double r = Math.random();           // Returns d such that: 0.0 <= d < 1.0

// Create a new Random object, seeding with the current time
Random generator = new Random(System.currentTimeMillis());
double d = generator.nextDouble();  // 0.0 <= d < 1.0
float f = generator.nextFloat();    // 0.0 <= f < 1.0
long l = generator.nextLong();      // Chosen from the entire range of long
int i = generator.nextInt();        // Chosen from the entire range of int
i = generator.nextInt(limit);       // 0 <= i < limit
boolean b = generator.nextBoolean(); // true or false later
d = generator.nextGaussian();        // Mean value: 0.0; std. deviation: 1.0
byte[] randomBytes = new byte[128];
generator.nextBytes(randomBytes);   // Fill in array with random bytes

// For cryptographic strength random numbers, use the SecureRandom subclass
SecureRandom generator2 = new SecureRandom();
// Have the generator generate its own 16-byte seed; takes a *long* time
generator2.setSeed(generator2.generateSeed(16)); // Extra random 16-byte seed
// Then use SecureRandom like any other Random object
generator2.nextBytes(randomBytes);   // Generate more random bytes
```

3. Bilangan Besar

Package `java.math` mengandung class `BigInteger` dan `BigDecimal`. Class-class ini memungkinkan kita bekerja dengan angka-angka yang sangat besar dan ketelitian yang sangat tinggi. Misalnya kita akan menghitung factorial dari 1000

```
package learn.core;
import java.math.*;
public class BigIntegerDemo {
    public static void main(String args[]) {
        // Compute the factorial of 1000
        BigInteger total = BigInteger.valueOf(1);
        for(int i = 2; i <= 1000; i++)
            total = total.multiply(BigInteger.valueOf(i));
        System.out.println(total.toString());
    }
}
```

Keluarannya seperti ini (kalau bisa membacanya dengan tepat, jago deh):

```
40238726007709377354370243392300398571937486421071463254379991042993851239862902059204420848696
94048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887
32519779505950995276120874975462497043601418278094646496291056393887437886487337119181045825783
64784997701247663288983595573543251318532395846307555740911426241747434934755342864657661166779
73966688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782
97308431392844403281231558611036976801357304216168747609675871348312025478589320767169132448426
2361314125087802080002616831510273418279770478463586817016436502415369139828126481021309276124
48963599287051149649754199093422215668325720808213331861168115536158365469840467089756029009505
3761647584772842188967964624494516076535340819890138544248798495995331910172335556602139450399
73628075013783761530712776192684903435262520001588853514733161170210396817592151090778801939317
81141945452572238655414610628921879602238389714760885062768629671466746975629112340824392081601
53780889893964518263243671616762179168909779911903754031274622289988005195444414282012187361745
99264295658174662830295557029902432415318161721046583203678690611726015878352075151628422554026
51704833042261439742869330616908979684825901254583271682264580665267699586526822728070757813918
58178889652208164348344825993266043367660176999612831860788386150279465955131156552036093988180
61213855860030143569452722420634463179746059468257310379008402443243846565724501440282188525247
0935190620929023136493273497565513958720559654228749774011413346962715422845862373875382304838
6568897646192738381490014076731044664025989949022221765904339901886018566526485061799702356193
89701786004081188972991831102117122984590164192106888438712185564612496079872290851929681937238
86426148396573822911231250241866493531439701374285319266498753372189406942814341185201580141233
44828015051399694290153483077644569099073152433278288269864602789864321139083506217095002597389
```

86355427719674282224875758676575234422020757363056949882508796892816275384886339690995982628095
61214509948717012445164612603790293091208890869420285106401821543994571568059418727489980942547
42173582401063677404595741785160829230135358081840096996372524230560855903700624271243416909004
153690105933983835777939410970027753472000
000
000
000

Bab IV.Class dan Object

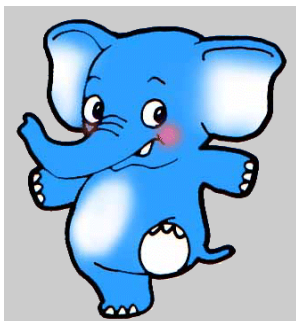
Apa hubungan antara class dan objek? class adalah cetak biru dari object. Ini berarti kita bisa membuat banyak objek dari satu macam class.

A.Konsep objek

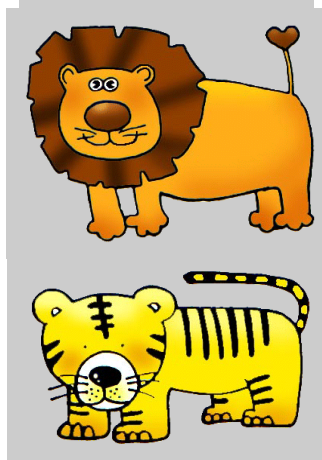
Lalu, objek itu apa? Secara umum, objek adalah sesuatu yang mempunyai identitas (nama), pada umumnya memiliki pengetahuan baik tentang dirinya maupun objek lain dan memiliki ketrampilan untuk melakukan sesuatu atau beberapa hal dan bisa bekerja sama dengan objek lain. Ini tampaknya batasan yang sangat umum, tetapi sebenarnya tidak, karena memang begitulah adanya.

Sebuah objek juga mempunyai peranan atau tanggung jawab tertentu. Artinya suatu objek memanfaatkan pengetahuan dan ketampilan yang dimilikinya untuk melaksanakan perannya pada sistem di mana dia hidup. Seperti halnya bakteri yang memiliki fungsi biodekomposisi, ataupun seorang pedagang yang bertanggung jawab dalam pendistribusian barang dagangannya. Dalam software, pengetahuan dari suatu objek biasa disebut dengan *state* atau *attribute* atau *field*. Sedang ketrampilan dan perilakunya disebut dengan *behaviour/function/method*.

Sebuah objek juga mungkin terdiri atas objek-objek lain. Seperti halnya objek mobil terdiri atas mesin, ban, kerangka mobil, pintu, karoseri dan lain-lain. Atau suatu objek boleh jadi merupakan turunan dari objek lain sehingga mewarisi sifat-sifat induknya. Misal Gajah, Singa dan Macan adalah Mamalia, sehingga Gajah, Singa dan Macan mempunyai sifat-sifat yang dimiliki oleh class Mamalia. Namun selain sifat-sifat umum Mamalia seperti melahirkan dan menyusui serta berdarah panas, Gajah, Singa dan Macan juga memiliki perilakunya sendiri yang khas.



adalah Mamalia



adalah Mamalia



adalah Mamalia

B. Model sebagai abstraksi dari dunia nyata

Untuk menyelesaikan masalah di dunia nyata dengan komputer, maka kita perlu mengubah representasi masalah tersebut ke dalam bentuk yang dimengerti oleh komputer. Seperti disiplin-disiplin ilmu rekayasa lainnya, dilakukan pemodelan yang berguna untuk membuat apa yang kita pikirkan tentang masalah tersebut menjadi lebih terstruktur, sehingga tidak hanya dimengerti oleh kita sendiri tapi juga orang lain. Model sendiri adalah abstraksi terhadap kenyataan atau realita yang ada. Mengapa kita perlu melakukan abstraksi? Dengan abstraksi, kita dapat menekankan mana bagian yang penting dalam pemecahan masalah dan mengabaikan mana yang tidak, sehingga kita dapat mengelola kompleksitas yang ada dan perubahan yang akan terjadi. Sebuah abstraksi yang baik dalam suatu ranah masalah belum tentu menjadi baik dalam ranah masalah yang lain. **Jadi model tidaklah dinilai dari benar salahnya tapi sejauh mana kegunaan dan kecocokannya dalam konteks penyelesaian masalah.** Untuk masalah yang sangat rumit kita mungkin akan mempunyai banyak model yang tersusun dalam sebuah hirarki model.

Nah, sekarang pertanyaannya adalah apa hubungannya antara model dan objek? Jawabannya akan kita bahas pada bab merancang objek.

1. Hirarki Model

Misal kita ingin melakukan simulasi permainan bola biliard. Kita dapat melakukannya tanpa permodelan sama sekali dengan cara mengambil gambar bola-bola biliard yang sedang bergerak pada setiap saat dengan video recorder untuk mengetahui posisinya. Dari data yang kita peroleh maka kita dapat melakukan simulasi ulang dengan komputer, tentu saja dengan cara seperti ini kita tidak dapat memperoleh informasi tambahan atau memprediksi hasil-hasil selanjutnya.

Cara yang lebih baik adalah dengan mencoba memahami fenomena atau proses Bergeraknya bola-bola tersebut. Kita tahu bahwa ada sunnatulloh atau hukum-hukum alam yang mengatur bagaimana bola-bola tersebut bergerak. Kita juga tentunya mengharapkan hukum-hukum ini berlaku terus sepanjang masa dan tidak berganti atau berubah sedikitpun. Semua orang yang pernah melewati masa SMA atau belajar fisika, pasti tahu setidaknya mengenai hukum mekanika Newton. Atau jika dia benar-benar menyukai fisika maka besar kemungkinan dia mengetahui relativitas khusus Einstein.

Secara sederhana kita dapat memodelkan bola biliard tersebut sebagai partikel pejal dengan massa dan diameter tertentu dan tidak saling mempengaruhi antara satu bola dengan bola lainnya. Tentu saja kita bisa memasukkan gaya gravitasi (hei, bola-bola itu punya massa bukan!) atau bahkan gaya-gaya antar partikel lainnya, tapi tampaknya kita belum memerlukannya sekarang. Abstraksi kita tidak memperhitungkan molekul-molekul atau atom-atom penyusun bola biliard tersebut apalagi sampai level elektron. Bahkan sifat-sifat termal, mekanik, optik, listrik dan magnetik yang merupakan sifat-sifat bahan, juga tidak perlu diperhitungkan. Don't add complexity that you don't need. Atau mungkin kita mempermasalahkan penggunaan hukum Newton karena dia memiliki pemahaman yang keliru mengenai konsep ruang-waktu dan kita menganggap bahwa Einsteinlah yang benar. Hey, like I said, there is nothing right and wrong with the model which is human's artifact. It is only more or less useful, and sometimes we may be getting more closer to reality. Kita mungkin akan memakai hukum Einstein jika kita bermain biliard dengan kecepatan bola yang mendekati kecepatan cahaya untuk memperhitungkan perubahan massa pada bola. YAGNI (You Ain't Gonna Need It) so don't add flexibility you are unlikely to use.

Model sederhana yang cukup untuk menyelesaikan masalah bola biliard tadi adalah dengan menggunakan hukum Newton ditambah hukum konservasi energi dan momentum. Dengan ini saja kita dapat mensimulasikan pergerakan bola biliard diatas meja terus-

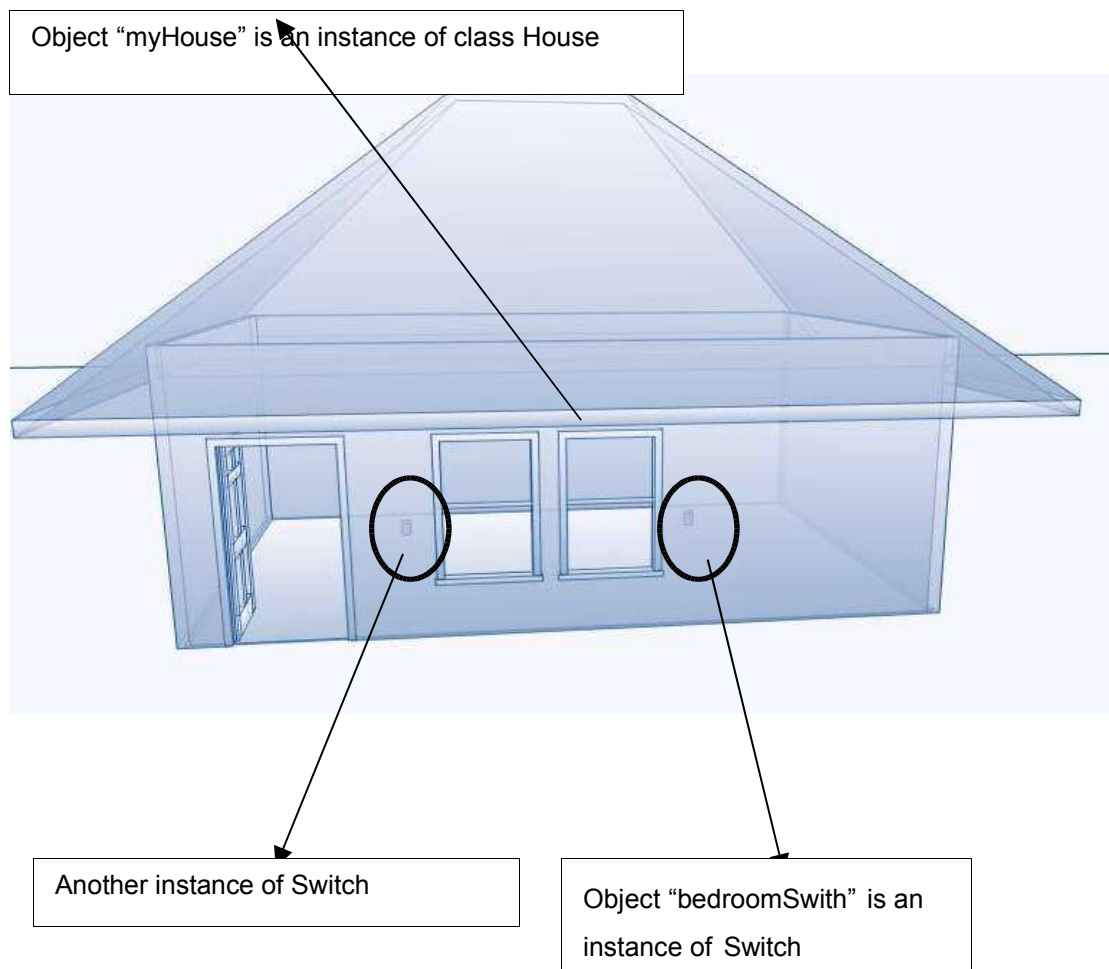
menerus tanpa berhenti, apabila kita ingin tampak lebih nyata maka kita tinggal memperhitungkan gaya gesek antara meja dan bola.

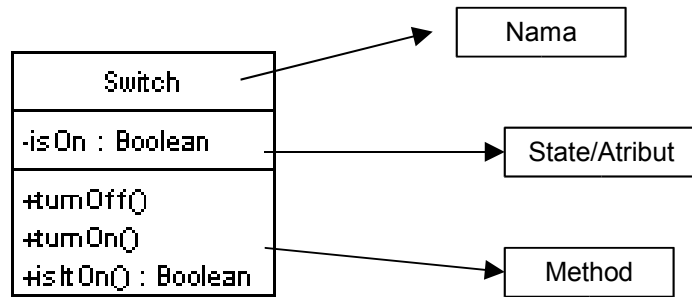
Contoh variasi permodelan partikel dalam UML (Unified Modeling Language)

Particle
-position : \Vector
-velocity : \Vector
-mass : double
-history : \Vector
+force(forceType:Set, other:Particle) : \Vector

Particle
-momentum : \Vector
-energy : \Vector
-history : \Vector
+force(forceType:Set, other:Particle)

C.Contoh abstraksi dari rumah dan switch





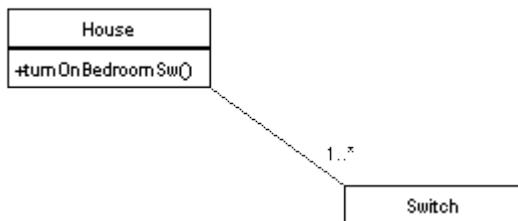
Class (UML)

Implementasinya dalam kode

```
public class Switch {
    private boolean isOn = false;

    /** Creates new Switch */
    public Switch() {
    }
    public void turnOn() { isOn = true;}
    public void turnOff() { isOn = false;}
    public boolean isItOn() {
        return isOn;
    }
}
```

Dalam contoh rumah ini kita melihat bahwa objek myHouse mempunyai dua objek Switch yaitu hallWaySwitch dan bedroomSwitch



UML Class Diagram

```

public class House {
    private Switch bedroomSwitch = new Switch();
    private Switch hallWaySwitch = new Switch();

    /** Creates new House */
    public House() {
    }

    public void turnOnBedroomSw() {
        bedroomSwitch.turnOn();
        System.out.println("bedroom switch is now on");
    }

    public static void main(String[] arg){
        House myHouse = new House();
        System.out.println("now, I turn on the bedroom switch");
        myHouse.turnOnBedroomSw();
    }
}

```

D.Referensi

Bagaimana cara kita menggunakan dan memanipulasi suatu objek? Pertama adalah kita harus memperoleh referensi dari objek yang ingin kita manipulasi.

Contoh:

```

boolean isOn; //pembuatan referensi isOn bertipe data boolean (deklarasi variabel)
isOn = false; //pemberian nilai false pada variabel isOn

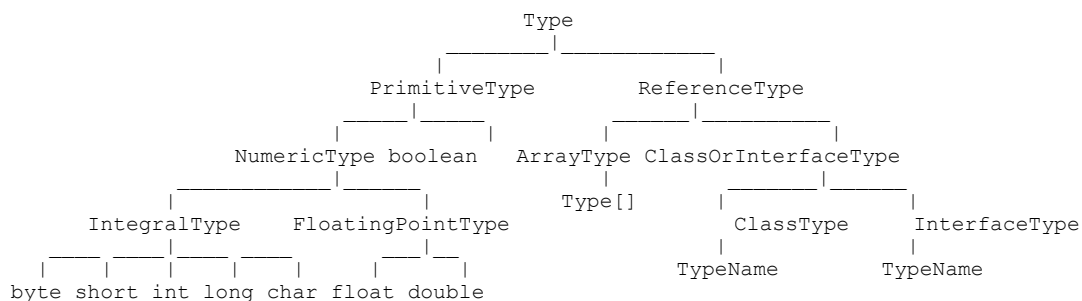
```

```

Switch bedroomSwitch; //pembuatan referensi bedroomSwitch bertipe objek Switch
bedroomSwitch = new Switch(); //penciptaan objek baru dan variabel bedroomSwitch dijadikan referensi

```

Dari sini kita bisa menyimpulkan bahwa secara umum variabel referensi ada dua jenis, yaitu tipe primitif dan tipe bentukan/objek.



Skema Hirarki Type

Tabel kata kunci untuk tipe data primitif

Keyword	Penggunaan	Keyword	Penggunaan
boolean	nila Boolean .	char	16 bit, 1 karakter Unicode.
byte	1 byte, bil. bulat.	float	4 byte, presisi-tunggal.
short	2 byte, bil. bulat.	double	8 byte, presisi-ganda.
long	8 byte, bil. bulat	int	4 byte, bil. bulat.

contoh penggunaan untuk tipe primitif:

```
int count = 0;
float velocity = 3.4F;
final double PI = 3.14156D;
boolean isOn = true, done = false;

velocity = 4.5F;
```

```
class Test {
    public static void main(String[] args) {
        int i = 1000000;
        System.out.println(i * i);
        long l = i;
        System.out.println(l * l);
        System.out.println(20296 / (l - i));
    }
}
```

outputnya:

```
-727379968
1000000000000
```

dilanjutkan dengan `ArithmeticException` pada pembagian dengan `l - i`, karena `l - i` adalah nol.

Tipe-tipe primitif mempunyai portabilitas yang sangat tinggi karena semua bahasa memiliki tipe-tipe data seperti ini. Masing-masing tipe mempunyai objek pembungkus yang immutable, yaitu: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean`, `Character`

contoh penggunaan tipe referensi Class dan Array

```
class Point {
    int x, y;
    Point() { System.out.println("default"); }
    Point(int x, int y) { this.x = x; this.y = y; }
    // A Point instance is explicitly created at class initialization time:
    static Point origin = new Point(0,0);
    // A String can be implicitly created by a + operator:
    public String toString() {
        return "(" + x + "," + y + ")";
    }
}

class Test {
    public static void main(String[] args) {
        // A Point is explicitly created using newInstance:
        Point p = null;
        try {
            p = (Point)Class.forName("Point").newInstance();
        } catch (Exception e) {
            System.out.println(e);
        }
        // An array is implicitly created by an array constructor:
        Point a[] = { new Point(0,0), new Point(1,1) };
        // Strings are implicitly created by + operators:
        System.out.println("p: " + p);
        System.out.println("a: { " + a[0] + ", " + a[1] + " }");
        // An array is explicitly created by an array creation expression:
        String sa[] = new String[2];
        sa[0] = "he"; sa[1] = "llo";
        System.out.println(sa[0] + sa[1]);
    }
}
```

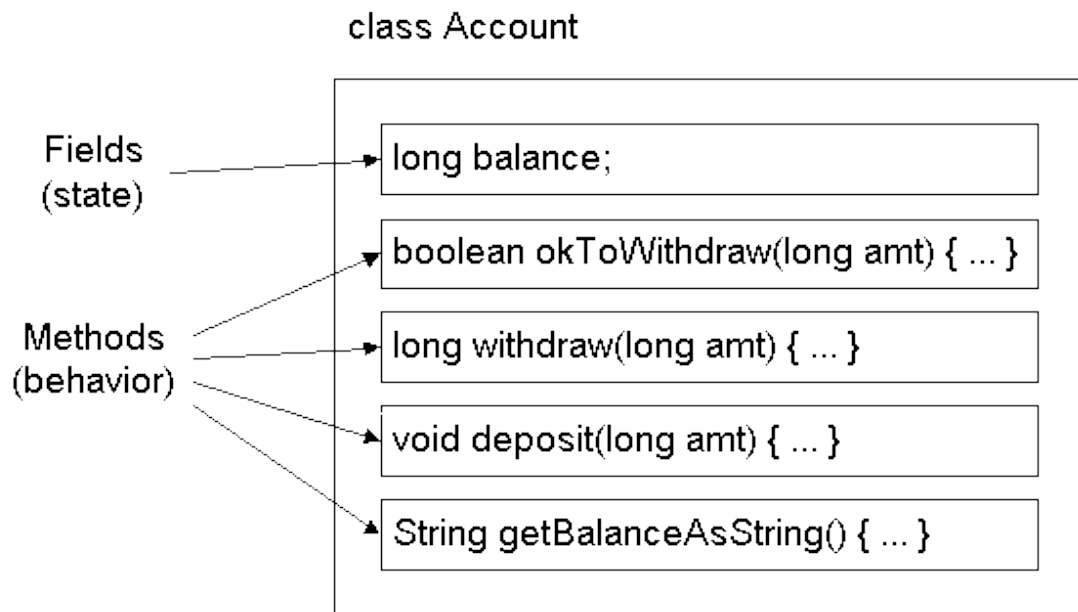
yang menghasilkan output:

```
default
p: (0,0)
a: { (0,0), (1,1) }
hello
```

E.Class sebagai cetak biru dari objek

Class mendefinisikan sebuah tipe dari objek. Di dalam class kita dapat mendeklarasikan variabel dan menciptakan objek (instansiasi). Sebuah class mempunyai anggota (member) yang terdiri atas field dan method. Contoh:

```
class Account {
    // class definition goes here
}
Account acct = new Account();
```



a)Konvensi penamaan Class:

- Seluruh kata ditulis bersambung kecuali untuk konstanta
- Nama Class – gunakan kata benda dan huruf pertama dari tiap kata ditulis dengan huruf besar: `String`, `ArrayIndexOutOfBoundsException`
- Nama Method – gunakan kata kerja dan kecuali huruf yang pertama, huruf awal tiap kata ditulis kapital: `replace()`, `equalsIgnoreCase()`
- Konstanta – Semuanya ditulis dengan huruf besar; pemisah antar kata menggunakan garis bawah: `MAX_VALUE`, `DECIMAL_DIGIT_NUMBER`

1.Field

Field adalah variabel data dan biasanya mempunyai nilai default dimana setiap objek mempunyai himpunan fieldnya (instance variable) sendiri

```
class Account {
    private long balance;
    // the methods...
}
```

2.Method

Method adalah fungsi yang mempunyai masukan dalam bentuk parameter dan mungkin menghasilkan nilai kembalian. Method hanya dapat didefinisikan di dalam class. Method terdiri atas dua bagian, yaitu *operation* dan *method body*.

```
public class Account {
    private long balance;
    public boolean okToWithdraw(long amount) {
        return (amount <= balance);
    }
}
```

```

    }
    public long withdraw(long amount)
        throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException(
                amount - balance);
        }
        balance -= amount;
        return amount;
    }
    public void deposit(long amount) {
        balance += amount;
    }
}
Account acct = new Account();
acct.deposit(15000L);
long cash = acct.withdraw(4000L);

```

3.Parameter

Parameter atau argumen adalah daftar tipe primitif dan referensi objek yang dipisahkan oleh tanda koma. Masing-masingnya harus mempunyai nama variabel. Semua parameter termasuk referensi objek di *pass by value*

```

public int lastIndexOf(String str, int fromIndex) {
    //...
}

```

Contoh *pass by value*

```

class Value { int val; }
class Test {
    public static void main(String[] args) {
        int i1 = 3;
        int i2 = i1;
        i2 = 4;
        System.out.print("i1==" + i1);
        System.out.println(" but i2==" + i2);
        Value v1 = new Value();
        v1.val = 5;
        Value v2 = v1;
        v2.val = 6;
        System.out.print("v1.val==" + v1.val);
        System.out.println(" and v2.val==" + v2.val);
    }
}

```

output yang dihasilkan silahkan direnungkan baik-baik:

```

i1==3 but i2==4
v1.val==6 and v2.val==6

```

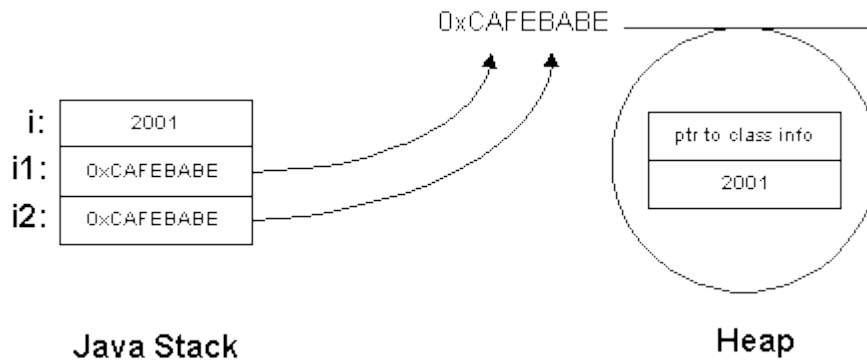
4.Letak data dalam memori.

Untuk tipe data primitif, referensi dan nilainya berada di dalam stack. Sedangkan untuk objek, referensinya berada di dalam stack yang menunjukkan alamat objek di dalam Heap

```

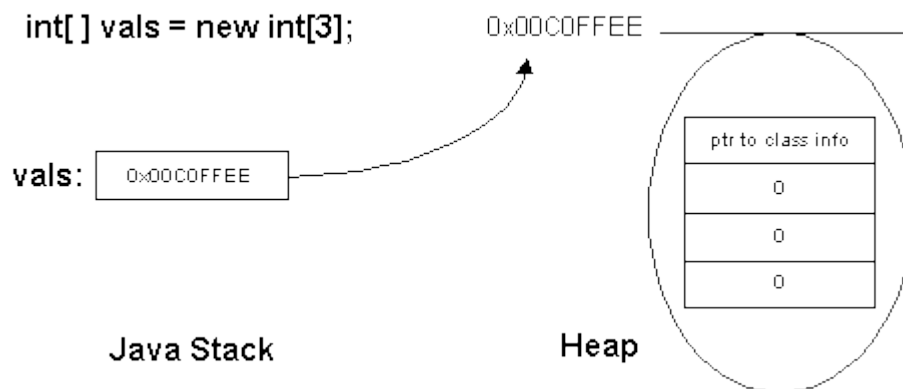
int i = 2001;
Integer i1 = new Integer(2001);
Integer i2 = i1;

```



5.Array

Array dalam Java juga diimplementasikan sebagai objek. JVM melakukan bounds checking pada saat run-time



6.Lingkup dari deklarasi variabel

```

{
    int i = 10;
    // hanya i yang bisa di akses di sini
    {
        int j = 20;
        // i dan j bisa diakses di sini
        // tapi i tidak dapat didefinisikan ulang
        // i = 23; illegal
    }
    // i masih bisa diakses, j tidak dapat diakses
}
{
    Integer i = new Integer(10);
    {
        Character c = new Character('q');
    }
    // c berada di luar lingkungannya
    // objek Character tidak mempunyai referensi
}
// i di luar lingkungannya
// objek Integer tidak mempunyai referensi

```

7.Kata kunci Static

digunakan untuk membuat class variable dan class method

class variable atau *private static field* bisa diakses oleh semua instan dari class tersebut dan juga oleh *class method*. Ia menjadi atribut yang umum bagi semua instan.

Contoh:

```
class Point {
    int x, y, useCount;
    Point(int x, int y) { this.x = x; this.y = y; }
    final static Point origin = new Point(0, 0);
}
class Test {
    public static void main(String[] args) {
        Point p = new Point(1,1);
        Point q = new Point(2,2);
        p.x = 3; p.y = 3; p.useCount++; p.origin.useCount++;
        System.out.println("(" + q.x + "," + q.y + ")");
        System.out.println(q.useCount);
        System.out.println(q.origin == Point.origin);
        System.out.println(q.origin.useCount);
    }
}
```

Menghasilkan:

```
(2,2)
0
true
1
```

class method dapat dipanggil tanpa melalui referensi dari objek.

Contoh: `main(String[] args)`, `System.out`, `java.lang.Math`, `Collections`, `Arrays`. Silahkan lihat dokumentasi API dari class-class tersebut.

Bab V.Inisialisasi dan CleanUp

- Apa itu constructor?
- diskusi tentang method dan constructor overloading
- constructor default, `this`, inisialisasi variable serta array
- urutan inisialisasi class dan object
- diskusi tentang finalization dan cleanup

Mengapa kita membutuhkan constructor? Sebelum kita bekerja dengan suatu objek, maka kita harus yakin bahwa objek itu berada pada satu keadaan tertentu atau ia tidak berada dalam kondisi yang tidak pasti (*indeterminate*). Artinya objek itu harus siap untuk bekerja terlebih dahulu dengan menentukan statenya. Caranya adalah dengan melakukan inisialisasi. Jika tidak ada constructor maka biasanya programmer dahulu menyiapkan fungsi `init()` atau `setup()`. Dengan adanya constructor maka fungsi inisialisasi ini secara otomatis akan dipanggil pada saat sebuah objek diciptakan

A.Constructor

```
public class Account {  
    private long balance;  
  
    public Account() {  
        balance = 0;  
    }  
  
    //...  
}  
public class Example1 {  
    public static void main(String[] args) {  
        // Create an empty account  
        Account acct = new Account();  
    }  
}
```

B.Method Overloading

Overloaded method mempunyai nama yang sama tetapi berbeda jumlah dan jenis parameternya:

```
public int indexOf(int ch) {  
    //...  
}  
public int indexOf(String str) {  
    //...  
}
```

C.Overloading and Return Value

Tidak dapat overload pada nilai kembalian...

```
public char[] substring(int beginIndex) {  
    //...  
}  
public String substring(int beginIndex) {  
    //...  
}
```

supaya tidak menimbulkan kebingungan seperti ini:

```
s.substring(5);
```

D.Overloading Constructors

```
public class Account {
    private long balance;

    public Account() {
        balance = 0;
    }

    public Account(long initDeposit) {
        balance = initDeposit;
    }

    //...
}

public class Example2 {
    public static void main(String[] args) {
        // Create an empty account
        Account acct1 = new Account();

        // Create an account with $ 1 Billion
        Account acct2 = new Account(1000000000000L);
    }
}
```

E.Constructor Default

Jika kita tidak mendefinisikan constructor...

```
public class Account {
    private long balance;
    // No constructor declared
    public boolean okToWithdraw(long amount) {
        return (amount <= balance);
    }
    // ...
}
```

maka compiler akan membuat *default constructor*:

```
public class Account {
    private long balance;

    public Account() {
    }

    public boolean okToWithdraw(long amount) {
        return (amount <= balance);
    }

    // ...
}
```

F.Referensi this

this adalah referensi terhadap diri sendiri (*self*):

```
public class Account {
    private long balance;

    public Account() {
        balance = 0;
    }

    public Account(long balance) {
        this.balance = balance;
    }
}
```

```
public boolean okToWithdraw(long amount) {
    return (amount <= this.balance);
}

//...
}
```

G.Pemanggilan `this` dalam Constructor

`this()` memanggil constructor yang lain dari dalam sebuah constructor. Ia harus berada pada baris pertama pada constructor dan hanya boleh ada satu untuk setiap constructor

Can't call `this()` from methods

```
public class Account {

    private long balance;
    private long minBal;

    public Account() {
        this(0, 200000);
    }

    public Account(long initDeposit) {
        this(initDeposit, 200000);
    }

    public Account(long initDeposit, long minBal) {
        balance = initDeposit;
        this.minBal = minBal;
    }

    //...
}
```

H.Inisialisasi Variabel

Instance (dan class) variabels mempunyai default value:

```
public class Account {

    private long balance;

    public void deposit(long amount) {
        balance += amount;
    }

    // ...
}
```

variabel local harus secara explicit diinisialisasi:

```
// THIS WON'T COMPILE
public class Example7 {

    public static void main(String[] args) {
        Account acct;
        // acct = new Account();
        acct.deposit(15000);
    }
}
```

I.Urutan Inisialisasi Object

Constructor dikompilasi menjadi `<init>()` method

Pertama `<init>()` memanggil `<init>()` yang lain

Kedua, initializer dan instance initialization block dieksekusi secara berurutan

Ketiga, tubuh constructor dijalankan

```
public class Account {  
  
    private long balance = 200;  
    private long minBal = 100;  
  
    {  
        balance = 400;  
        minBal = 200;  
    }  
  
    public Account(long initDeposit, long minBal) {  
  
        balance = initDeposit;  
        this.minBal = minBal;  
    }  
  
    // ...  
}  
  
public class Example8 {  
  
    public static void main(String[] args) {  
        Account acct = new Account(800, 400);  
    }  
}
```

J.Urutan Inisialisasi Class

Static initialization code dikompilasi menjadi method `<clinit>()`

Static initializers dan static initialization blocks dieksekusi secara berurutan

```
class CoffeeCup {  
  
    private static int cupCount = 355;  
  
    static {  
        cupCount =  
            PersistentStorage.getLastCupCount();  
    }  
  
    //...  
}
```

Class diinisialisasi pada penggunaannya pertama kali.

K.Insialisasi Array

Dua cara untuk mendeklarasikan array:

```
int[] value;  
int value[];
```

Untuk memperoleh object array, harus secara eksplisit:

```
private int[] value = new int[10];  
  
{  
    for (int i = 0; i < value.length; ++i) {  
        value[i] = i;  
    }  
}
```

Dapat juga menggunakan tanda kurung kriting (curly braces):

```
private int[] value1 = { 0, 1, 2, 3, 4 };  
  
private Integer[] value2 = {  
    new Integer(0),  
    new Integer(1)  
};
```

`arrayVar.length` memberikan panjang dari sebuah array

Multi-dimensional array diimplementasikan sebagai arrays of arrays

L.Finalization dan Cleanup

- Memori untuk object yang tak terpakai lagi secara otomatis dibebaskan oleh garbage collector
- `finalize()` secara otomatis dipanggil oleh garbage collector sebelum memori dibebaskan
- Object mungkin saja tidak di garbage collected
- Tidak boleh bergantung pada `finalize()` untuk melepaskan sumber daya yang terbatas.

Bab VI.Reusing Classes

Actually we don't "reuse", we simply "use"

Ada dua cara untuk menggunakan suatu class yang sudah ada, yaitu:

1. dengan komposisi
2. dengan pewarisan atau inheritansi

Kemudian kita akan belajar apa itu polymorphism dan bagaimana menggunakannya secara tepat.

A.Komposisi objek.

Secara sederhana adalah suatu objek menggunakan (*use*) objek lain atau objek mengandung/memiliki objek lain.

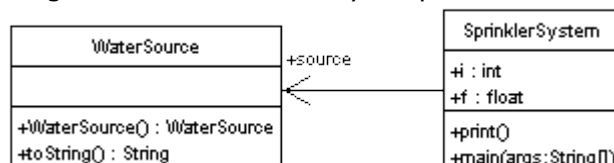
Contoh:

```
package learn.reuse.composition;
class WaterSource {
    private String s;
    WaterSource() {
        System.out.println("WaterSource()");
        s = new String("Constructed");
    }
    public String toString() { return s; }
}
public class SprinklerSystem {
    private String valve1, valve2, valve3, valve4;
    WaterSource source;
    int i;
    float f;
    void print() {
        System.out.println("valve1 = " + valve1);
        System.out.println("valve2 = " + valve2);
        System.out.println("valve3 = " + valve3);
        System.out.println("valve4 = " + valve4);
        System.out.println("i = " + i);
        System.out.println("f = " + f);
        System.out.println("source = " + source);
    }
    public static void main(String[] args) {
        SprinklerSystem x = new SprinklerSystem();
        x.print();
    }
}
```

Keluarannya:

```
valve1 = null
valve2 = null
valve3 = null
valve4 = null
i = 0
f = 0.0
source = null
```

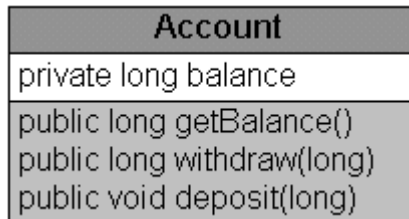
Diagram UML sederhananya seperti ini:



B.Composition with forwarding

Ini adalah kasus khusus dari komposisi objek di mana kompositor memanggil method dari objek yang dikandungnya.

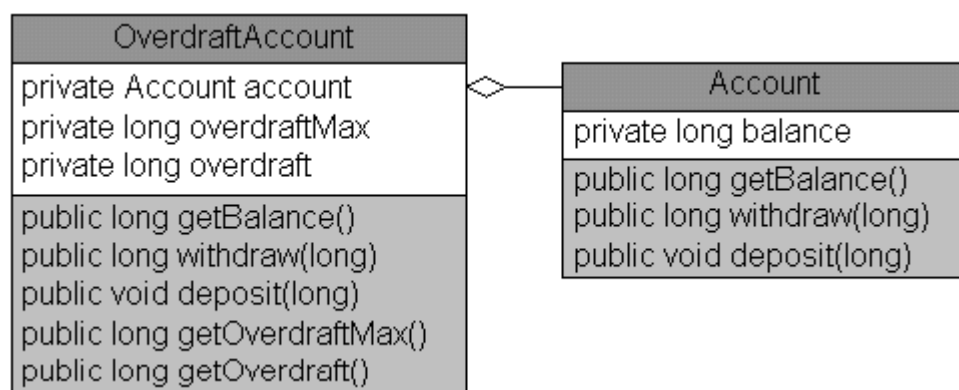
Contoh:



```
public class Account {
    private long balance;
    public long withdraw(long amount)
        throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        if (amount > balance) {
            throw new InsufficientFundsException(amount - balance);
        }
        balance -= amount;
        return amount;
    }
    public void deposit(long amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        long newBal = balance + amount;
        if (newBal < 0) {
            throw new ArithmeticException();
        }
        balance = newBal;
    }
    public long getBalance() {
        return balance;
    }
}
```

Apabila kita ingin menciptakan jenis Account yang lain kita bisa menggunakan class Account yang sudah tersedia.

Contoh:



```
public class OverdraftAccount {
    private Account account = new Account();
    private long overdraft;
    public OverdraftAccount(long overdraftMax) {
        this.overdraftMax = overdraftMax;
    }
    public long getOverdraft() {
        return overdraft;
    }
}
```

```
}
public long getOverdraftMax() {
    return overdraftMax;
}
public long getBalance() {
    return account.getBalance();
}
public long withdraw(long amount)
throws InsufficientFundsException {
    if (amount <= 0) {
        throw new IllegalArgumentException();
    }
    long bal = account.getBalance();
    if (bal >= amount) {
        return account.withdraw(amount);
    }
    long shortfall = amount - bal;
    long extraAvailable = overdraftMax - overdraft;
    if (shortfall > extraAvailable) {
        throw new InsufficientFundsException(shortfall
        - extraAvailable);
    }
    overdraft += shortfall;
    account.withdraw(amount - shortfall);
    return amount;
}
public void deposit(long amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException();
    }
    if (overdraft > 0) {
        if (amount < overdraft) {
            overdraft -= amount;
        }
        else {
            long diff = amount - overdraft;
            overdraft = 0;
            account.deposit(diff);
        }
    }
    else {
        account.deposit(amount);
    }
}
}
```

C. Inheritansi atau pewarisan

Bagian ini adalah bagian yang paling banyak diberikan contohnya jadi selamat menarik kesimpulan.

Contoh:

```
package learn.reuse.inheritance;

public class A {
    private int i = 0; // hidden from all other objects, except instances of A
    protected double d1 = 0.0D; // subclasses of A will inherit this
    protected double d2 = 0.0D; // subclasses of A will inherit this
    public void setI(int in) {
        i = in;
    }
    public void setd1(double din) {
        d1 = din;
    }
    public void setd2(double din) {
        d2 = din;
    }
    public void print() {
        System.out.println("A's print() called");
        System.out.println("d1: "+d1+", d2:"+d2);
    }
    public double acc(double d1) {
        System.out.println("A's acc(double) called");
        return this.d1+this.d2+d1;
    }
}
```



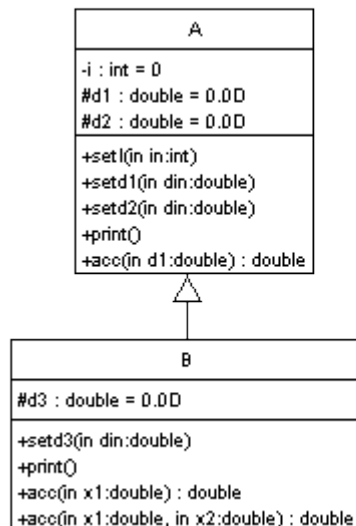
```
package learn.reuse.inheritance;
public class B extends A {
    protected double d3 = 0.0D;
    public void setd3(double din) {
        d3 = din;
    }
    // print() overrides print() in class A
    public void print() {
        System.out.println(" B's print() called");
        //System.out.println("i1: "+i); i is a private data member of A,
        // compile time error
        System.out.println("d1: "+d1+", d2: "+d2+", d3: "+d3);
        // d1, d2, are protected data members of A, and are inherited by B
    }
    // acc(double) overrides acc(double) in class A
    public double acc(double x1) {
        System.out.println(" B's acc(double) called");
        return super.acc(x1)+d3;
    }
    // acc(double, double) overloads acc(double) in A
    public double acc(double x1, double x2) {
        System.out.println(" B's acc(double, double) called");
        return d1+d2+d3+x1+x2;
    }
}
package learn.reuse.inheritance;

public class MainClass {
    public static void main(String[] args) {
        B myB = new B();
        myB.setd1(3.4D); // invoking setd1(double) inherited from A
        myB.setd2(4.4D); // invoking setd2(double) inherited from A
        myB.setd3(5.5D);
        myB.print();
        System.out.println(myB.acc(5.5D));
        System.out.println(myB.acc(5.5D, 6.5D));
        A myA = new B(); // myA is declared as a reference to an object of
        // type A, but is actually "pointing" to an object of type B
        myA.setd1(2.3D);
        myA.setd2(2.6D);
        // myA.setd3(6.6D); setd3 not defined in A. Compile time error!
        myA.print(); // polymorphic B's print() called
        myA.acc(5.6D); // polymorphic B's acc(double) called
        // myA.acc(3.4D, 5.6D); acc(double, double) not defined in A
        // compile time error!
    }
}
```

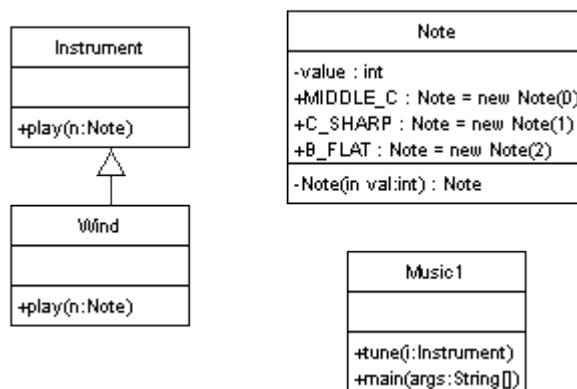
Outputnya sebagai berikut:

```
B's print() called
d1: 3.4, d2: 4.4, d3: 5.5
 B's acc(double) called
A's acc(double) called
18.8
 B's acc(double, double) called
25.3
 B's print() called
d1: 2.3, d2: 2.6, d3: 0.0
 B's acc(double) called
A's acc(double) called
```

Diagram UML dari kelas di atas adalah:



Lima contoh selanjutnya adalah tentang instrumen musik



```

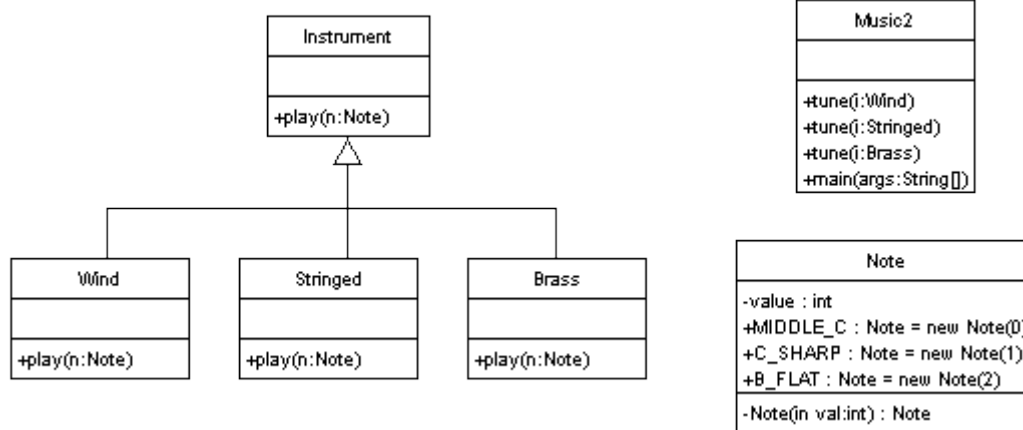
package learn.reuse.inheritance.music1;

class Note {
    private int value;
    private Note(int val) { value = val; }
    public static final Note
    MIDDLE_C = new Note(0),
    C_SHARP = new Note(1),
    B_FLAT = new Note(2);
} // Etc.
class Instrument {
    public void play(Note n) {
        System.out.println("Instrument.play()");
    }
}
// Wind objects are instruments
// because they have the same interface:
class Wind extends Instrument {
    // Redefine interface method:
    public void play(Note n) {
        System.out.println("Wind.play()");
    }
}
public class Music1 {
    public static void tune(Instrument i) {
        // ...
        i.play(Note.MIDDLE_C);
    }
    public static void main(String[] args) {
        Instrument flute = new Wind();
        tune(flute); // Upcasting
    }
}
    
```

Outputnya sederhana saja:

Wind.play()

Contoh kedua dari alat-alat musik.



```

package learn.reuse.inheritance.music2;

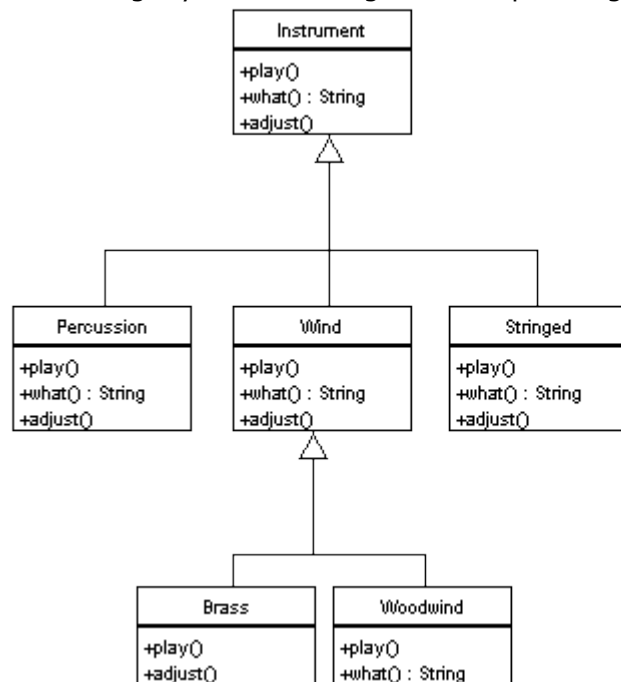
class Note {
    private int value;
    private Note(int val) { value = val; }
    public static final Note
    MIDDLE_C = new Note(0),
    C_SHARP = new Note(1),
    B_FLAT = new Note(2);
} // Etc.
class Instrument {
    public void play(Note n) {
        System.out.println("Instrument.play()");
    }
}
class Wind extends Instrument {
    public void play(Note n) {
        System.out.println("Wind.play()");
    }
}
class Stringed extends Instrument {
    public void play(Note n) {
        System.out.println("Stringed.play()");
    }
}
class Brass extends Instrument {
    public void play(Note n) {
        System.out.println("Brass.play()");
    }
}
public class Music2 {
    public static void tune(Wind i) {
        i.play(Note.MIDDLE_C);
    }
    public static void tune(Stringed i) {
        i.play(Note.MIDDLE_C);
    }
    public static void tune(Brass i) {
        i.play(Note.MIDDLE_C);
    }
    public static void main(String[] args) {
        Wind flute = new Wind();
        Stringed violin = new Stringed();
        Brass frenchHorn = new Brass();
        tune(flute); // No upcasting
        tune(violin);
        tune(frenchHorn);
    }
}

```

Keluarannya adalah:

Wind.play()
Stringed.play()
Brass.play()

Versi ketiganya mulai mengenalkan upcasting



```

package learn.reuse.inheritance.music3;

import java.util.*;
class Instrument {
    public void play() {
        System.out.println("Instrument.play()");
    }
    public String what() {
        return "Instrument";
    }
    public void adjust() {}
}
class Wind extends Instrument {
    public void play() {
        System.out.println("Wind.play()");
    }
    public String what() { return "Wind"; }
    public void adjust() {}
}
class Percussion extends Instrument {
    public void play() {
        System.out.println("Percussion.play()");
    }
    public String what() { return "Percussion"; }
    public void adjust() {}
}
class Stringed extends Instrument {
    public void play() {
        System.out.println("Stringed.play()");
    }
    public String what() { return "Stringed"; }
    public void adjust() {}
}
class Brass extends Wind {
    public void play() {
        System.out.println("Brass.play()");
    }
    public void adjust() {
        System.out.println("Brass.adjust()");
    }
}
}
    
```

```

class Woodwind extends Wind {
    public void play() {
        System.out.println("Woodwind.play()");
    }
    public String what() { return "Woodwind"; }
}
}
public class Music3 {
    // Doesn't care about type, so new types
    // added to the system still work right:
    static void tune(Instrument i) {
        // ...
        i.play();
    }
    static void tuneAll(Instrument[] e) {
        for(int i = 0; i < e.length; i++)
            tune(e[i]);
    }
    public static void main(String[] args) {
        Instrument[] orchestra = new Instrument[5];
        int i = 0;
        // Upcasting during addition to the array:
        orchestra[i++] = new Wind();
        orchestra[i++] = new Percussion();
        orchestra[i++] = new Stringed();
        orchestra[i++] = new Brass();
        orchestra[i++] = new Woodwind();
        tuneAll(orchestra);
    }
}
}

```

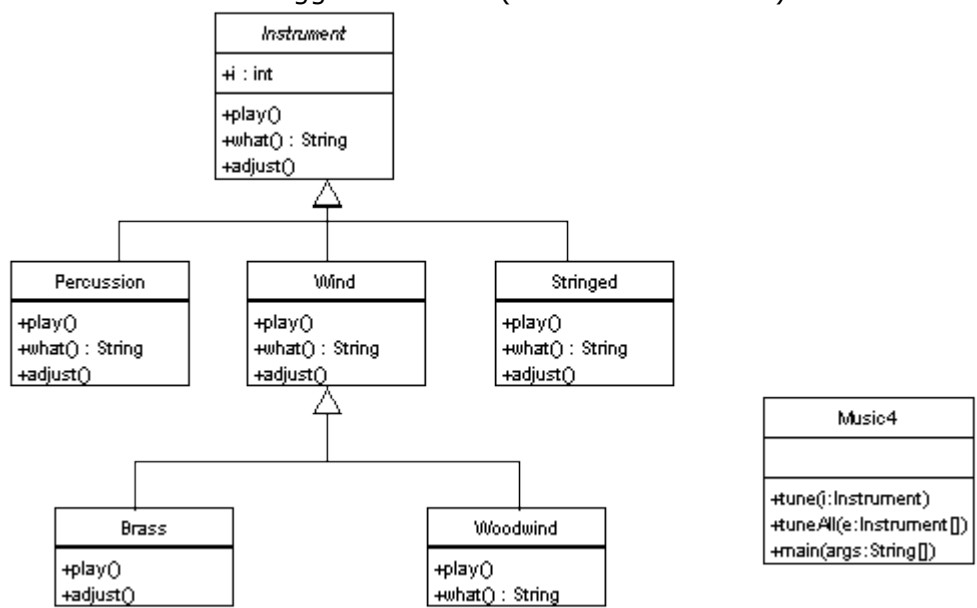
Hasilnya:

```

Wind.play()
Percussion.play()
Stringed.play()
Brass.play()
Woodwind.play()

```

Edisi ke 4 mulai menggunakan ABC (Abstract Base Class) dari Instrument



```

package learn.reuse.inheritance.music4;
import java.util.*;
abstract class Instrument {
    int i; // storage allocated for each
    public abstract void play();
    public String what() {
        return "Instrument";
    }
    public abstract void adjust();
}
class Wind extends Instrument {
    public void play() {
        System.out.println("Wind.play()");
    }
}

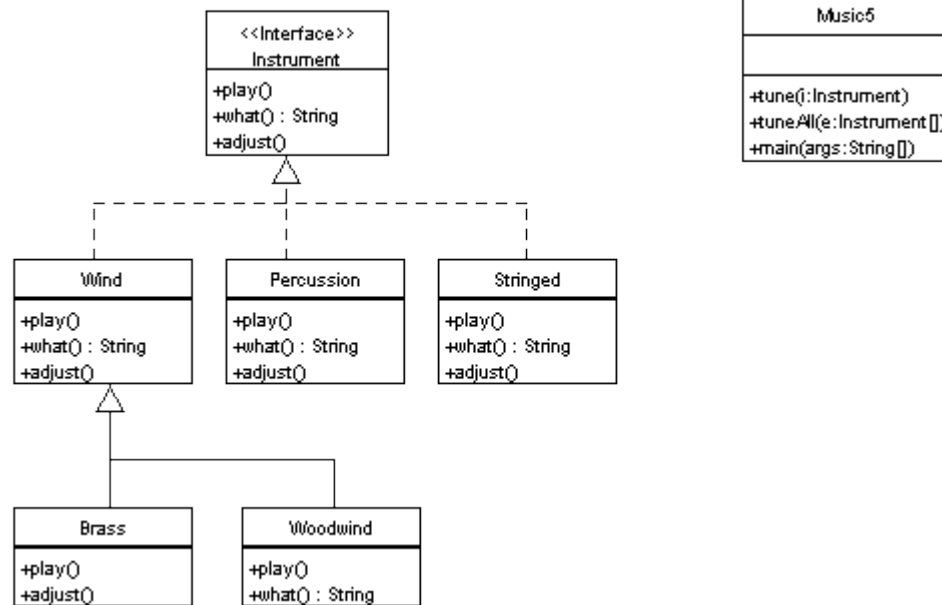
```

```
    }
    public String what() { return "Wind"; }
    public void adjust() {}
}
class Percussion extends Instrument {
    public void play() {
        System.out.println("Percussion.play()");
    }
    public String what() { return "Percussion"; }
    public void adjust() {}
}
class Stringed extends Instrument {
    public void play() {
        System.out.println("Stringed.play()");
    }
    public String what() { return "Stringed"; }
    public void adjust() {}
}
class Brass extends Wind {
    public void play() {
        System.out.println("Brass.play()");
    }
    public void adjust() {
        System.out.println("Brass.adjust()");
    }
}
class Woodwind extends Wind {
    public void play() {
        System.out.println("Woodwind.play()");
    }
    public String what() { return "Woodwind"; }
}
public class Music4 {
    // Doesn't care about type, so new types
    // added to the system still work right:
    static void tune(Instrument i) {
        // ...
        i.play();
    }
    static void tuneAll(Instrument[] e) {
        for(int i = 0; i < e.length; i++)
            tune(e[i]);
    }
    public static void main(String[] args) {
        Instrument[] orchestra = new Instrument[5];
        int i = 0;
        // Upcasting during addition to the array:
        orchestra[i++] = new Wind();
        orchestra[i++] = new Percussion();
        orchestra[i++] = new Stringed();
        orchestra[i++] = new Brass();
        orchestra[i++] = new Woodwind();
        tuneAll(orchestra);
    }
}
```

Keluarannya:

```
Wind.play()
Percussion.play()
Stringed.play()
Brass.play()
Woodwind.play()
```

Contoh terakhir dari alat-alat musik dengan diperkenalkannya interface sebagai pure abstract class.



```

package learn.reuse.inheritance.music5;

import java.util.*;
interface Instrument {
    // Compile-time constant:
    int i = 5; // static & final
    // Cannot have method definitions:
    void play(); // Automatically public
    String what();
    void adjust();
}
class Wind implements Instrument {
    public void play() {
        System.out.println("Wind.play()");
    }
    public String what() { return "Wind"; }
    public void adjust() {}
}
class Percussion implements Instrument {
    public void play() {
        System.out.println("Percussion.play()");
    }
    public String what() { return "Percussion"; }
    public void adjust() {}
}
class Stringed implements Instrument {
    public void play() {
        System.out.println("Stringed.play()");
    }
    public String what() { return "Stringed"; }
    public void adjust() {}
}
class Brass extends Wind {
    public void play() {
        System.out.println("Brass.play()");
    }
    public void adjust() {
        System.out.println("Brass.adjust()");
    }
}
class Woodwind extends Wind {
    public void play() {
        System.out.println("Woodwind.play()");
    }
    public String what() { return "Woodwind"; }
}
public class Music5 {
    // Doesn't care about type, so new types
    // added to the system still work right:
    static void tune(Instrument i) {

```

```
        // ...
        i.play();
    }
    static void tuneAll(Instrument[] e) {
        for(int i = 0; i < e.length; i++)
            tune(e[i]);
    }
    public static void main(String[] args) {
        Instrument[] orchestra = new Instrument[5];
        int i = 0;
        // Upcasting during addition to the array:
        orchestra[i++] = new Wind();
        orchestra[i++] = new Percussion();
        orchestra[i++] = new Stringed();
        orchestra[i++] = new Brass();
        orchestra[i++] = new Woodwind();
        tuneAll(orchestra);
    }
}
```

Keluarannya juga sama:

```
Wind.play()
Percussion.play()
Stringed.play()
Brass.play()
Woodwind.play()
```

D.Kapan menggunakan composititon atau inheritance

Prinsip kedua dari pemrograman OOP yang baik adalah "favour composition over inheritance".

Composition mempunyai keuntungan-keuntungan: low coupling, menjaga encapsulation kerugiannya perlu kerja yang lebih banyak untuk membungkus method-method yang sudah standar dan tidak dapat polymorphism.

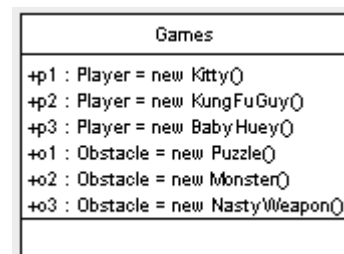
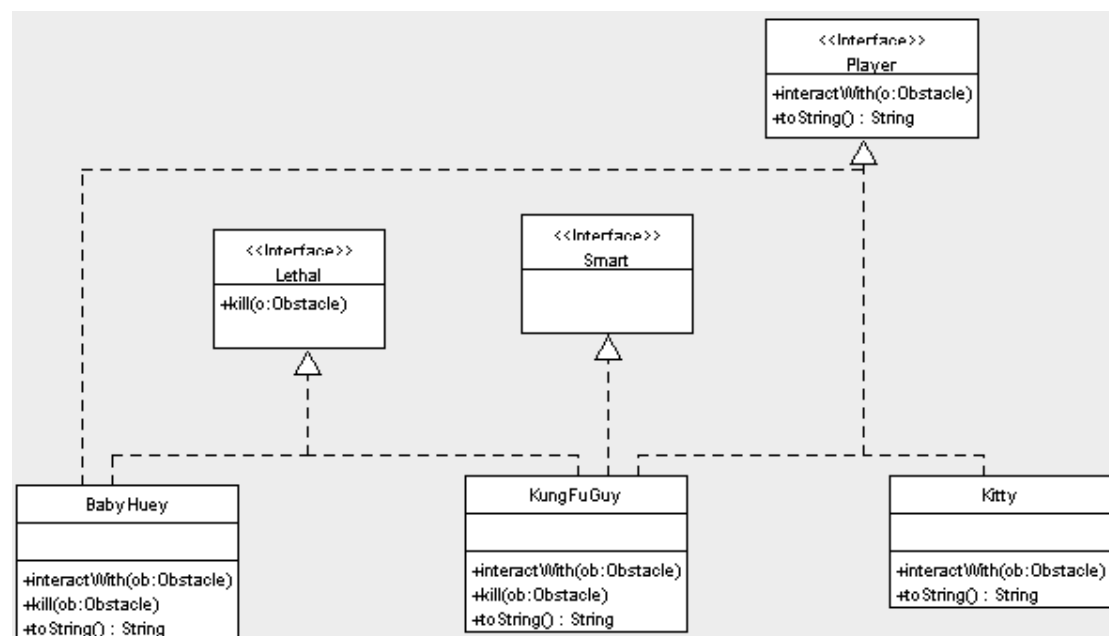
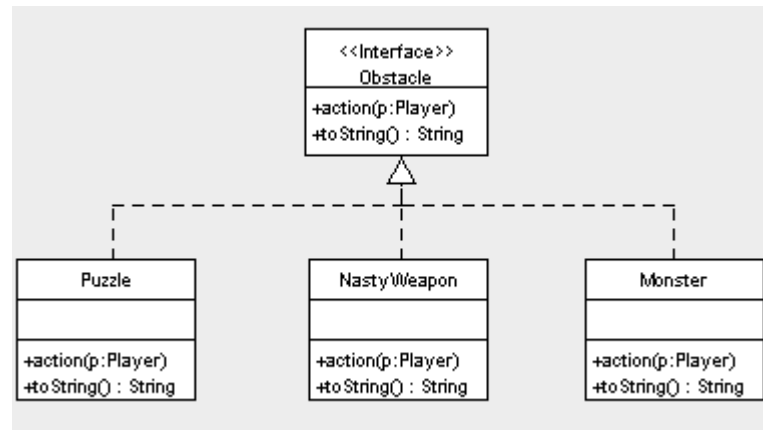
Sedang inheritance keuntungannya adalah polymorphism dan mudah untuk menambah data apabila behaviour nya tidak jauh beda. Kerugiannya adalah tight coupling dan encapsulation yang mulai terbuka.

Tentu saja kedua kekuatan ini bisa digabungkan dengan menggunakan interface :)
contoh:

TBD

Bab VII.Polymorphism dan Interface

Konsep pemrograman dengan interface dalam konteks multiple role yang membuat program kita modular sehingga mudah untuk mengganti atau menambah implementasi.



Fase pertama kita mempunyai tiga interface, dua player dan dua obstacle

```

package learn.reuse.games;
interface Obstacle {
    void action(Player p);
    String toString();
}
interface Player {

```

```
        void interactWith(Obstacle o);
        String toString();
    }
    interface Smart {
    }
    interface Lethal {
        void kill(Obstacle o);
    }
    class Kitty implements Player {
        public void interactWith(Obstacle ob) {
            System.out.println("Kitty has encountered a " + ob);
            ob.action(this);
        }
        public String toString(){
            return "Kitty";
        }
    }
    class KungFuGuy implements Player,Smart,Lethal {
        public void interactWith(Obstacle ob) {
            System.out.println("KungFuGuy now battles a " + ob);
            ob.action(this);
        }
        public void kill(Obstacle ob) {
            System.out.println("Sorry " + ob + ", you are terminated, hahaha!");
        }
        public String toString(){
            return "KungFuGuy";
        }
    }

    class Puzzle implements Obstacle {
        public void action(Player p) {
            if (p instanceof Smart){
                System.out.println("He then solves the puzzle");
            } else System.out.println("This puzzle perplexes me");
        }
        public String toString(){
            return "Puzzle";
        }
    }
    class Monster implements Obstacle {

        public void action(Player p) {
            if ( p instanceof Lethal) {
                Lethal lethal = (Lethal)p;
                lethal.kill(this);
            } else System.out.println("I think the Monster gonna eat me...");
        }
        public String toString(){
            return "Monster";
        }
    }
}
package learn.reuse.games;
public class Games{
    Player p1 = new Kitty();
    Player p2 = new KungFuGuy();
    Obstacle o1 = new Puzzle();
    Obstacle o2 = new Monster();
}

package learn.reuse.games;
public class Games1 {
    public static void main(String args[]) {
        Games g = new Games();
        Player[] players = {g.p1, g.p2};
        Obstacle[] obstacles = {g.o1, g.o2};
        for (int i = 0; i < players.length; i++) {
            for (int j = 0; j < obstacles.length; j++) {
                players[i].interactWith(obstacles[j]);
            }
        }
    }
}
```

Output dari Games1

```
Kitty has encountered a Puzzle
This puzzle perplexes me
```

```
Kitty has encountered a Monster
I think the Monster gonna eat me...
KungFuGuy now battles a Puzzle
He then solves the puzzle
KungFuGuy now battles a Monster
Sorry Monster, you are terminated, hahaha!
```

Perhatikan bahwa penggunaan `if-then-else` di lokalisasi pada masing-masing kelas sehingga perubahan logic lebih mudah untuk dilakukan. Sedang pada program utama tampak bahwa "tidak ada hal yang istimewa". Jika anda belum terbiasa dengan polymorphism mungkin anda masih bingung, tapi jika sudah terbiasa semua akan tampak mengalir saja.

Selanjutnya adalah menambah sebuah halangan lagi.

```
class NastyWeapon implements Obstacle {
    public void action(Player p) {
        if ((p instanceof Smart) && (p instanceof Lethal)){
            System.out.println("He then destroyed Nasty Weapon");
        } else System.out.println("This nasty weapon scares me");
    }
    public String toString(){
        return "Nasty Weapon";
    }
}

package learn.reuse.games;
public class Games{
    Player p1 = new Kitty();
    Player p2 = new KungFuGuy();
    Obstacle o1 = new Puzzle();
    Obstacle o2 = new Monster();
    Obstacle o3 = new NastyWeapon();
}

package learn.reuse.games;
public class Games2 {
    public static void main(String args[]) {
        Games g = new Games();
        Player[] players = {g.p1, g.p2};
        Obstacle[] obstacles = {g.o1, g.o2, g.o3};
        for (int i = 0; i < players.length; i++) {
            for (int j = 0; j < obstacles.length; j++) {
                players[i].interactWith(obstacles[j]);
            }
        }
    }
}
```

Output Games2 adalah sebagai berikut:

```
Kitty has encountered a Puzzle
This puzzle perplexes me
Kitty has encountered a Monster
I think the Monster gonna eat me...
Kitty has encountered a Nasty Weapon
This nasty weapon scares me
KungFuGuy now battles a Puzzle
He then solves the puzzle
KungFuGuy now battles a Monster
Sorry Monster, you are terminated, hahaha!
KungFuGuy now battles a Nasty Weapon
He then destroyed Nasty Weapon
```

Dengan polymorphism perubahan menjadi lebih mudah bukan!
Kemudian kita coba untuk menambah player baru

```
class BabyHuey implements Player,Lethal {
```

```
        public void interactWith(Obstacle ob) {
            System.out.println("BabyHuey now plays with a " + ob);
            ob.action(this);
        }
        public void kill(Obstacle ob) {
            System.out.println("Oops, are you dead, " + ob + "?");
        }
        public String toString(){
            return "BabyHuey";
        }
    }
}
package learn.reuse.games;
public class Games{
    Player p1 = new Kitty();
    Player p2 = new KungFuGuy();
    Player p3 = new BabyHuey();
    Obstacle o1 = new Puzzle();
    Obstacle o2 = new Monster();
    Obstacle o3 = new NastyWeapon();
}
package learn.reuse.games;
public class Games3 {
    public static void main(String args[]) {
        Games g = new Games();
        Player[] players = {g.p1, g.p2, g.p3};
        Obstacle[] obstacles = {g.o1, g.o2, g.o3};
        for (int i = 0; i < players.length; i++) {
            for (int j = 0; j < obstacles.length; j++) {
                players[i].interactWith(obstacles[j]);
            }
        }
    }
}
}
```

Output dari Games3 ini adalah:

```
Kitty has encountered a Puzzle
This puzzle perplexes me
Kitty has encountered a Monster
I think the Monster gonna eat me...
Kitty has encountered a Nasty Weapon
This nasty weapon scares me
KungFuGuy now battles a Puzzle
He then solves the puzzle
KungFuGuy now battles a Monster
Sorry Monster, you are terminated, hahaha!
KungFuGuy now battles a Nasty Weapon
He then destroyed Nasty Weapon
BabyHuey now plays with a Puzzle
This puzzle perplexes me
BabyHuey now plays with a Monster
Oops, are you dead, Monster?
BabyHuey now plays with a Nasty Weapon
This nasty weapon scares me
```

Perhatikan penggunaan interface `Lethal` yang hanya memuat satu method yaitu `kill` (`Obstacle`) oleh `BabyHuey` yang diterima dengan baik oleh sang `Monster`

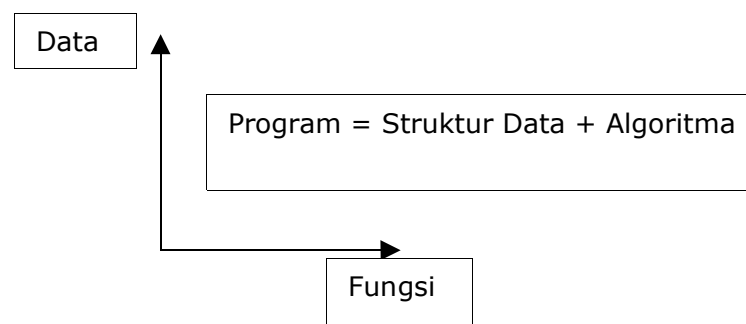
Bab VIII.Merancang Class

Setelah kita belajar bagaimana menggunakan class buatan orang lain yakni API, selanjutnya kita akan mencoba sendiri untuk membuat sebuah class. Kita sudah paham bahwa class adalah prototipe dari sebuah objek sementara objek adalah representasi dari sebuah konsep di dunia nyata.

Sebelum dikenalnya teknologi objek, maka representasi dari dunia nyata terbagi menjadi dua bagian besar, yaitu keadaan dan proses. Contoh: apabila kita ingin merepresentasikan sebuah sinyal elektromagnetik, kita dapat menuliskannya dalam bentuk data yaitu dengan kumpulan titik-titik yang bernilai tertentu pada satu waktu (keadaan), atau dengan fungsi-fungsi yang menunjukkan perubahan nilai tersebut pada sebuah interval waktu (proses).

Apabila sinyalnya tidak terlalu rumit maka fungsi-fungsi yang sederhana sudah cukup untuk melukiskan sinyal tersebut, contoh apabila kita tahu sinyalnya berbentuk linear, gelombang atau bersifat periodik maka fungsinya adalah fungsi-fungsi yang mempunyai sifat/perilaku seperti itu juga. Tapi jika sinyalnya adalah sinyal derau atau sinyal yang selalu berubah-ubah frekuensinya, pokoknya rumit deh, maka representasi data akan lebih baik untuk melukiskan sinyal tersebut.

Yang terpenting disini adalah cara pandang kita yaitu bentuk representasi apa yang kita ambil menentukan informasi apa yang akan kita peroleh dan apa yang mungkin kita bisa lakukan terhadap informasi tersebut.



Paradigma ini yang dipakai dalam bahasa-bahasa tak berbasis objek, bahkan Bahasa Pascal sendiri mempunyai prinsip "Program itu adalah Struktur Data dengan Algoritma untuk memanipulasinya".

Contoh yang lain misalnya kita mempunyai data nilai seluruh mahasiswa dari semua mata kuliah yang diambil untuk beberapa semester lalu kita memutuskan apa yang bisa kita lakukan terhadap data tersebut (data centric) seperti mencari rata-rata kelas dan menghitung IPK setiap anak.

Semua library dari pemrograman prosedural adalah berupa fungsi yang menerima semua parameter masukan yang ia butuhkan dari luar fungsi. Akibatnya semua parameter masukan ataupun nilai kembalian menjadi variabel global.

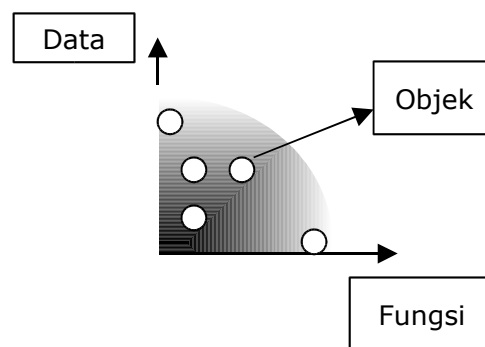
A.Pentingnya berfikir secara objek

Tidak ada yang salah dalam pemrograman prosedural, dan kita bisa membuat sebuah program yang baik (mudah dibaca dan dimengerti oleh manusia) dengan bahasa-bahasa prosedural. Untuk membuat class dengan baik pun kita juga harus tetap mengerti

pemrograman prosedural yang baik. Sedangkan komputer sendiri tidak akan ambil pusing apakah program yang ia jalankan ditulis dalam bahasa prosedural atau objek bahkan bahasa assembler sekalipun, toh komputer tidak bisa mengeluh.

Teknologi objek memang dimaksudkan untuk manusia, bukan untuk mesin. Dan seperti yang telah kita lihat pada contoh-contoh sebelumnya, apabila masalah yang kita hadapi mempunyai kompleksitas yang lebih tinggi, atau berubah-ubah dengan cepat maka paradigma berfikir prosedural menemui batasnya dan tidak cukup mampu lagi untuk menangani masalah. Karena cara berfikirnya sendiri tidak mampu untuk mengerti tentang masalah tersebut apalagi mencari pemecahan masalah.

Tidaklah cukup untuk membagi sebuah masalah besar hanya menjadi dua buah sisi: data dan fungsi saja. Dengan prinsip objek maka kita bisa melakukan zoom in atau zoom out dari suatu masalah sehingga kita bisa menemukan struktur atau hierarki permasalahan dalam tingkat kedetilan atau resolusi yang kita inginkan. Berfikir secara objek sebetulnya juga adalah cara yang lebih alami dalam memahami dan menyelesaikan suatu masalah.



B. Dekomposisi prosedural dan dekomposisi struktur objek.

Bagian terumit (untuk dibaca apalagi dimengerti) dari sebuah program prosedural adalah pernyataan kondisional atau seleksi dari sebuah fungsi dalam memproses suatu logic. Bayangkanlah sebuah kondisional yang bersarang dan bertingkat, seperti berikut ini:

```
//procedural pseudo-code
process(Event e) {
    if (e.getAgreement() equals (agreement1)) {
        if (e.type().equals(eventType1)) {
            if (e.date().laterThan(aug21)) {
                multiplyByRate(e, e.getCustomer().getRate());
                //do processing logic by subroutine
                else if ...
            }
            else if (e.type().equals(eventType1)) {
                ...
            }
            else {}
        }
    }
    if (e.getAgreement() equals (agreement2)) {
        ...
    }
}
```

Objek menggantikan fungsi-fungsi kondisional seperti di atas dengan struktur data yang mudah navigasinya dan sesuai dengan domain model. Ada lagi yang lebih bagus dari ini, apabila kita harus melakukan perubahan pada logic (dan ini tampaknya akan sering untuk dilakukan ☺), maka kita tidak perlu memeriksa keseluruhan pernyataan kondisional dan

melakukan banyak perubahan yang sangat rentan kesalahan, tetapi kita mungkin hanya perlu melakukan perubahan secara lokal atau sesederhana menginstansiasi sebuah objek dan menempelkannya dalam program kita (terutama apabila kita memrogram dengan interface).

Pokok perbedaan utama antara pendekatan objek dengan pendekatan prosedural yang terstruktur rapi adalah pada saat kita perlu untuk kerap mengganti logika kondisional atau seleksi. Dengan pendekatan objek, kita dapat melakukannya tanpa banyak merombak kode sumber.

Tapi ini bukanlah keuntungan yang terbesar. Keuntungannya yang terbesar adalah untuk membuat sebuah program prosedural yang terstruktur dengan baik dibutuhkan kedisiplinan yang sangat tinggi, padahal jarang pemrogram yang bisa disiplin terutama di saat dia bisa melakukan sesuatu dengan cara yang lebih mudah sehingga mengacaukan struktur programnya. Dengan struktur objek yang sudah ada maka programmer dipandu untuk menjadi disiplin tanpa disadarinya, sehingga struktur programnya tidak mudah untuk rusak. Ini juga yang menjadi dasar dari *polymorphism* yang menyebabkan semua berjalan dengan tampak lebih elegan.

Nah, sekarang pertanyaannya, kapan sebaiknya kita menggunakan pendekatan prosedural (*selection conditional clause*) dan kapan pendekatan objek. Apabila suatu logika proses seleksi sederhana dan kemungkinan berubahnya sedikit, maka disitulah digunakan prosedural, tapi kalau yang terjadi sebaliknya, objek memberikan fleksibilitas dan skalabilitas yang lebih baik.

C. Merancang Class Money

Pengembang perangkat lunak yang berpengalaman selain menggunakan alat-alat pemrograman (IDE, profiler, JUnit, UML tools) dengan efektif, ia juga mempunyai perpustakaan class sendiri baik hasil dari belajar maupun dari hasil kerja sebelumnya. Jadi, jangan menunda-nunda untuk membuat perpustakaan class sendiri, tidak perlu takut salah atau jelek hasilnya, toh nanti bisa disempurnakan dengan refactoring.

Software-software aplikasi yang paling banyak dipakai oleh orang selain aplikasi umum seperti Office/OpenOffice adalah aplikasi keuangan atau finansial, misal akuntansi, inventory, payroll, billing, bank, asuransi, dan sebagainya. Perkembangan teknologi terutama IT memang tidak bisa dilepaskan dari bisnis, dan bisnis tidak bisa dilepaskan dari akuntansi atau keuangan. Nah sekarang anda mengerti kan, mengapa banyak sekali lowongan pekerjaan yang membutuhkan keahlian seorang akuntan ☺. Dan mereka mau membayar mahal banyak orang untuk mengamati tingkah laku uang, dari mana ia datang, kemana ia pergi, dan apa yang terjadi dengannya. Perangkat lunak berperan penting untuk memudahkan pekerjaan catat-mencatat, mengelompokkan, dan meringkas seperti ini. Lagipula hitung-menghitung kan memang tugas "komputer".

Class-class yang akan kita buat sekarang digunakan untuk menyelesaikan masalah hitung-hitungan aritmetika dalam bermacam-macam mata uang. Aritmetika dalam satu mata uang yang sama tentu sangat mudah, kita tinggal menjumlah dua buah nilai dan mengabaikan satuan mata uang tersebut. Dan kita cukup menggunakan tipe primitif seperti int atau double untuk merepresentasikan objek uang.

Bagaimana jika kita ingin program yang bisa melibatkan berbagai jenis mata uang? Bahkan perusahaan software besar seperti SAP AG, pernah mengalami kesulitan dengan satuan mata uang karena memang belum memperhitungkan akan terjadinya perubahan mata uang Eropa menjadi Euro. Akibatnya mereka harus melakukan perbaikan besar-besaran pada program mereka tanpa mengganggu proses yang sedang berjalan di banyak perusahaan di Eropa dan SAP adalah program dengan kompleksitas yang besar. Kita tidak

bisa begitu saja mengkonversi satu mata uang ke dalam mata uang lain untuk melakukan perhitungan aritmatika karena tidak ada satu nilai tukar tunggal. Nilai tukar untuk hari ini boleh jadi berbeda dengan nilai kemarin, malah mungkin bisa berbeda hanya dalam hitungan jam.

Kesulitan dalam mengkonversi ke satu mata uang tidak akan terjadi apabila digunakan pendekatan objek dengan benar. Dengan perilaku seperti yang dilukiskan di atas, tipe primitif saja tidak cukup untuk merepresentasikan konsep uang. Ia layak mendapatkan objeknya sendiri. Uang adalah salah satu bentuk khusus dari kuantitas atau jumlah (**Quantity**). Jika anda mendapat sebuah pertanyaan "berapa jumlahnya?" dan kemudian dijawab 50, tentu masih tersisa di benak anda pertanyaan lanjutan "50 apa? 50 buah, 50 ekor, 50 pasang, 50 kg, atau 50 jam?" Sehari-hari kita merepresentasikan kuantitas lengkap dengan dimensi satuan dibelakangnya. Misal: tingginya 170 cm, berat badan 45 kg, menunggu selama 25 menit. Saya rasa cuma uang saja yang satuannya berada di depan jumlahnya, hal ini mungkin karena satuannya lebih penting dari pada nilainya ☺. Walaupun sama-sama berurusan dengan banyak angka-angka besar dan bekerja berdasarkan hukum konservasi, sangat jarang orang akuntansi yang mengerti fisika, begitu juga sebaliknya, anda tahu mengapa? Ini adalah juga akibat dari letak satuan tersebut ☺. Fisikawan mungkin biasa melihat kuantitas dari orde 10^{-14} m (ukuran elektron) sampai dengan orde 10^{40} m (ukuran galaksi), tapi dia bisa pusing kalau disuruh menghitung uang, padahalkan sama-sama angka.

Mari kita mulai dengan yang mudah, prinsip XP (*eXtreme Programming*): *build and do the simplest things that can possibly work*. Untuk diperhatikan, class yang kita buat sekarang bukanlah implementasi lengkap dan jelas bukan yang terbaik, tapi cukuplah untuk mengilustrasikan bagaimana cara membuat class yang baik. Limitasi dari class-class yang akan kita buat adalah: *Money* hanya direpresentasikan dengan int dan String sehingga tidak memperhitungkan pembulatan apalagi pembagian.

Kita definisikan class *Money* untuk merepresentasikan sebuah nilai dalam satu mata uang. Kita representasikan jumlah uang dengan sebuah integer *int*. Agar mendapatkan ketelitian penuh, anda mungkin perlu menggunakan `java.math.BigInteger` atau `java.math.BigDecimal` untuk menyimpan bilangan berpresisi sangat tinggi. Mata uang *tidak boleh* direpresentasikan dalam tipe `double` karena akan bermasalah dalam pembulatan dan ketelitiannya, kalau ingin simpel bisa menggunakan tipe data long dengan `BigDecimal` untuk pembulatannya. Selanjutnya mata uang direpresentasikan sebagai String yang menyimpan tiga huruf singkatan ISO dari mata uang (IDR, USD, CHF, etc.). Dalam implementasi yang lebih lanjut, mata uang (*currency*) layak untuk menjadi objek tersendiri seperti yang telah dilakukan pada Java 1.4 (`java.util.Currency`).

Salah satu kelebihan *Money* dari *Quantity* adalah kemampuannya dalam menangani pembulatan. *Money* sering dinyatakan bersama dengan bagian desimalnya. Untuk mata uang rupiah kita tidak pernah membaca Rp 345.345,45 sehari-hari kecuali dalam rekening di bank, tapi untuk mata uang lain ini adalah hal yang biasa. Apabila kita menggunakan `double` untuk merepresentasikan nilai uang, maka perilaku pembulatannya sulit untuk ditebak sehingga bisa menimbulkan bug yang sulit untuk ditemukan. Lain halnya jika kita menggunakan `BigInteger` maupun `BigDecimal` yang mempunyai sampai delapan mode pembulatan. Sebuah objek *Money*, tentunya dapat mempunyai aturan sendiri tentang pembulatan sehingga pemakai tidak perlu menyadarinya saat bekerja dengannya.

Pembulatan banyak dipakai pada pembagian. Jika anda membagi Rp 10.000 menjadi tiga, hasilnya tidak sesederhana Rp 3.333,33 tentu saja. Masalahnya jika anda kalikan tiga lagi akan didapatkan Rp 9.999,99 – berarti ada sen yang hilang. Para akuntan tidak menyukai jika ada sen yang hilang. Untuk itu perlu diketahui kebijakan apa yang berlaku untuk situasi seperti ini, biasanya yang berlaku adalah seseorang akan mendapatkan sen ekstra

tersebut. Akibatnya kita dapat menaruh method pada objek `Money` untuk mengembalikan kumpulan `Money` hasil dari pembagian. Kasus yang lebih menarik lagi misalnya apabila kita ingin menghitung bagi hasil antara beberapa pihak dengan rasio tertentu. Contohnya bagi hasil antara pihak bank dan nasabah dengan rasio 3:7, ini juga memerlukan penanganan yang lebih khusus.

`Money` tergolong dalam jenis *value object*, yang berarti ia dibedakan bukan dari identitas objeknya melainkan melalui nilainya dan diperiksa melalui method `equals(Object)`. Biasanya dalam implementasi *value object*, ia dibuat *immutable* sehingga nilainya tidak berubah sejak pertama kali ia diciptakan. Apabila kita ingin mengubah nilainya maka akan dihasilkan objek baru dengan nilai yang baru pula.

Limitasi dari class-class yang akan kita buat adalah:

`Money` hanya direpresentasikan dengan `int` dan `String`, dan operasi aritmatika hanya berupa penjumlahan. Untuk operasi yang lain silahkan dikerjakan sebagai latihan. Jangan lupa untuk pembagian pembulatan harus diperhitungkan

Kita coba lihat versi purba dari `Money`

```
class Money {
    private int amount;
    private String currency;
    public Money(int amount, String currency) {
        this.amount = amount;
        this.currency = currency;
    }

    public int amount() {
        return amount;
    }

    public String currency() {
        return currency;
    }
}
```

Kompilasi dan coba jalankan . Hasilnya adalah:

```
java.lang.NoSuchMethodError: main
Exception in thread "main"
```

Tentu saja karena kita memang belum menulis method mainnya. Untuk memastikan class kita bekerja dengan baik maka kita perlu melakukan tes. Dan tool yang baik untuk Java programmer adalah JUnit (www.junit.org), sebuah framewok untuk unit testing. JUnit mudah untuk digunakan dan sangat sederhana, tapi kita belum akan menggunakannya sekarang, karena hanya akan menambah beban pelajaran untuk pemula. Tapi, dan ini penting, jika anda benar-benar ingin menjadi seorang programmer yang profesional, jangan pernah ragu-ragu untuk menguasai JUnit. Jika diberi umur dan kesehatan, Insya Allah ini akan dibahas pada buku selanjutnya.

Catatan kegunaan unit testing selain automatic test:

1. Unit testing dapat menangkap user requirement secara nyata (berupa kode) dan memberikan arah bagi programmer apa yang diharapkannya dari class yang akan ia buat, terutama apabila ia berprinsip "*write test first*".
2. Unit testing juga memudahkan pencarian kegagalan (failure) maupun kesalahan (error)
3. Unit testing memberikan contoh bagaimana suatu class digunakan, dan meyakinkan client tentang betapa handalnya class yang dibuat.

Salah satu cara untuk menyiapkan tes yang mudah adalah dengan menuliskannya pada method `main`. Itulah yang telah kita lakukan pada program-program terdahulu walau bukan dimaksudkan untuk melakukan tes, melainkan untuk menjalankan aplikasi. Tes yang dituliskan diperiksa secara manual artinya kita menampilkan outputnya dan membandingkannya dengan hasil yang diharapkan. Cara yang paling mudah adalah dengan *override* method `toString` untuk memberi kita informasi tentang objek yang dimaksud.

Perhatian: walaupun cara di atas mudah, tapi saya tidak menganjurkannya. Cara yang terbaik, pastikan semua tes sepenuhnya otomatis dan dapat memeriksa hasilnya sendiri. Pelajaran yang bisa diambil dari sini adalah lebih baik punya tes yang jelek daripada tidak punya tes sama sekali.

Berikut adalah tes yang pertama, penjumlahan sederhana dari mata uang yang sama.

```
public Money add(Money m) {
    //assertSameCurrencyAs(m);
    return new Money(amount()+ m.amount(), currency());
}
public String toString() {
    StringBuffer buffer = new StringBuffer();
    buffer.append("[ "+amount()+" "+currency()+" ]");
    return buffer.toString();
}
public static void main(String[] args){
    Money m12CHF = new Money(12, "CHF");
    Money m14CHF = new Money(14, "CHF");
    Money result = m12CHF.add(m14CHF);
    System.out.println(m12CHF + " ditambah " + m14CHF + " adalah: " + result);
}
```

kemudian tambahkan tes lagi pada `main` untuk memeriksa apakah dua objek uang sama nilainya.

```
Money m26CHF = new Money(26, "CHF");
System.out.println(result.equals(m26CHF));
```

methodnya seperti ini:

```
public boolean equals(Object anObject) {
    if (anObject instanceof Money) {
        Money aMoney= (Money)anObject;
        return aMoney.currency().equals(currency())
            && amount() == aMoney.amount();
    }
    return false;
}
```

Untuk satu jenis mata uang tampaknya sudah cukup, sekarang kita akan coba untuk mata uang yang berbeda. Untuk mengatasi masalah nilai tukar yang berubah-ubah, kita perkenalkan `MoneyBag` yang menunda masalah konversi sampai saatnya diperlukan. Misal 12 Swiss Franc ditambah 7 US Dollar menghasilkan satu `MoneyBag` yang mengandung kedua mata uang tersebut, 12 CHF dan 7 USD. Apabila kita menambahkan 21 USD lagi ke dalamnya maka isi tas itu menjadi 12CHF dan 28USD.

Beginilah bentuk dasar dari `MoneyBag`

```
class MoneyBag {
    private Vector Monies= new Vector();

    MoneyBag(Money m1, Money m2) {
        appendMoney(m1);
        appendMoney(m2);
    }

    MoneyBag(Money bag[]) {
```

```
        for (int i= 0; i < bag.length; i++)
            appendMoney(bag[i]);
    }
}
```

method `appendMoney` adalah method internal yang akan menambahkan money ke dalam vector `Monies` dan akan membereskan masalah perbedaan mata uang.

Dengan `MoneyBag` yang sudah ada kita coba untuk merubah method `add` pada `Money`.

```
public Money add(Money m) {
    if (m.currency().equals(currency()) )
        return new Money(amount()+m.amount(), currency());
    return new MoneyBag(this, m);
}
```

Kalau kita coba untuk mengkompilasi method di atas maka kita tidak akan berhasil karena ia mengharapkan `Money` bukan `MoneyBag` sebagai nilai kembaliannya. Saatnya untuk mengenalkan objek baru sebagai indireksi atau penengah terhadap `Money` dan `MoneyBag`. Untuk itu kita buat sebuah interface `IMoney` yang diimplementasikan oleh keduanya.

Ini dia interface `IMoney`:

```
interface IMoney {
    IMoney add(IMoney i);
    IMoney addMoney(Money m);
    IMoney addMoneyBag(MoneyBag b);
}
```

Untuk menyembunyikan representasi yang berbeda dari kode klien maka harus diperhitungkan semua kombinasi penjumlahan antara `Money` dengan `MoneyBag`. Sebelum dilanjutkan, mari kita definisikan beberapa tes lagi.

```
Money m7USD = new Money( 7, "USD");
Money m21USD = new Money(21, "USD");
MoneyBag mB1 = new MoneyBag(m12CHF, m7USD);
MoneyBag mB2 = new MoneyBag(m14CHF, m21USD);

//money + money mixed currency
// [12 CHF] + [7 USD] == {[12 CHF][7 USD]}
System.out.println(m12CHF + " + " + m7USD + " = " + m12CHF.add(m7USD));

//money + moneyBag
// [14 CHF] + {[12 CHF][7 USD]} == {[26 CHF][7 USD]}
System.out.println(m14CHF + " + " + mB1 + " = " + m14CHF.add(mB1));

//moneyBag + money
// {[12 CHF][7 USD]} + [14 CHF] == {[26 CHF][7 USD]}
System.out.println(mB1 + " + " + m14CHF + " = " + mB1.add(m14CHF));

//moneyBag + moneyBag
// {[12 CHF][7 USD]} + {[14 CHF][21 USD]} == {[26 CHF][28 USD]}
System.out.println(mB1 + " + " + mB2 + " = " + mB1.add(mB2));
```

yang menyebabkan hal di atas adalah polymorphism dan mekanisme callback / double dispatch di bawah ini.

```
class Money implements IMoney {
    public IMoney add(IMoney m) {
        return m.addMoney(this);
    }
    //...
}
class MoneyBag implements IMoney {
    public IMoney add(IMoney m) {
        return m.addMoneyBag(this);
    }
    //...
}
```

Inilah hasil akhirnya:

```
package learn.object;

class Money implements IMoney{
    private int amount;
    private String currency;
    public Money(int amount, String currency) {
        this.amount = amount;
        this.currency = currency;
    }
    public int amount() {
        return amount;
    }
    public String currency() {
        return currency;
    }
    public boolean equals(Object anObject) {
        if (anObject instanceof Money) {
            Money aMoney= (Money)anObject;
            return aMoney.currency().equals(currency())
                && amount() == aMoney.amount();
        }
        return false;
    }
    public int hashCode() {
        return currency.hashCode()+ amount;
    }
    public IMoney add(IMoney m) {
        return m.addMoney(this);
    }
    public IMoney addMoney(Money m) {
        if (m.currency().equals(currency()) )
            return new Money(amount()+m.amount(), currency());
        return new MoneyBag(this, m);
    }
    public IMoney addMoneyBag(MoneyBag s) {
        return s.addMoney(this);
    }
    /*
    public Money add(Money m) {
        //assertSameCurrencyAs(m);
        return new Money(amount()+ m.amount(), currency());
    }
    */

    public boolean isZero() {
        return amount() == 0;
    }
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("[ "+amount()+" "+currency()+" ]");
        return buffer.toString();
    }
    public static void main(String[] args){
        Money m12CHF = new Money(12, "CHF");
        Money m14CHF = new Money(14, "CHF");
        Money result = m12CHF.add(m14CHF);
        System.out.println(m12CHF + " ditambah " + m14CHF + " adalah: " + result);

        Money m26CHF = new Money(26, "CHF");
        System.out.println(result.equals(m26CHF));

        Money m7USD = new Money( 7, "USD");
        Money m21USD = new Money(21, "USD");
        MoneyBag mB1 = new MoneyBag(m12CHF, m7USD);
        MoneyBag mB2 = new MoneyBag(m14CHF, m21USD);

        //money + money mixed currency
        // [12 CHF] + [7 USD] == {[12 CHF][7 USD]}
        System.out.println(m12CHF + " + " + m7USD + " = " + m12CHF.add(m7USD));

        //money + moneyBag
        // [14 CHF] + {[12 CHF][7 USD]} == {[26 CHF][7 USD]}
        System.out.println(m14CHF + " + " + mB1 + " = " + m14CHF.add(mB1));
    }
}
```

```

        //moneyBag + money
        // {[12 CHF][7 USD]} + [14 CHF] == {[26 CHF][7 USD]}
        System.out.println(mB1 + " + " + m14CHF + " = " + mB1.add(m14CHF));

        //moneyBag + moneyBag
        // {[12 CHF][7 USD]} + {[14 CHF][21 USD]} == {[26 CHF][28 USD]}
        System.out.println(mB1 + " + " + mB2 + " = " + mB1.add(mB2));
    }

}
package learn.object;
import java.util.*;
class MoneyBag implements IMoney {
    private Vector Monies= new Vector();
    MoneyBag(Money m1, Money m2) {
        appendMoney(m1);
        appendMoney(m2);
    }
    MoneyBag(Money bag[]) {
        for (int i= 0; i < bag.length; i++)
            appendMoney(bag[i]);
    }
    MoneyBag(Money m, MoneyBag bag) {
        appendMoney(m);
        appendBag(bag);
    }
    MoneyBag(MoneyBag m1, MoneyBag m2) {
        appendBag(m1);
        appendBag(m2);
    }
    private void appendMoney(Money aMoney) {
        IMoney old= findMoney(aMoney.currency());
        if (old == null) {
            Monies.addElement(aMoney);
            return;
        }
        Monies.removeElement(old);
        IMoney sum= old.add(aMoney);
        if (sum.isZero())
            return;
        Monies.addElement(sum);
    }
    private void appendBag(MoneyBag aBag) {
        for (Enumeration e= aBag.Monies.elements(); e.hasMoreElements(); )
            appendMoney((Money)e.nextElement());
    }
    private Money findMoney(String currency) {
        for (Enumeration e= Monies.elements(); e.hasMoreElements(); ) {
            Money m= (Money) e.nextElement();
            if (m.currency().equals(currency))
                return m;
        }
        return null;
    }
    public IMoney add(IMoney m) {
        return m.addMoneyBag(this);
    }
    public IMoney addMoney(Money m) {
        return (new MoneyBag(m, this)).simplify();
    }
    public IMoney addMoneyBag(MoneyBag s) {
        return (new MoneyBag(s, this)).simplify();
    }
    public boolean isZero() {
        return Monies.size() == 0;
    }
    private IMoney simplify() {
        if (Monies.size() == 1)
            return (IMoney)Monies.elements().nextElement();
        return this;
    }
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("{");
        for (Enumeration e= Monies.elements(); e.hasMoreElements(); )
            buffer.append((Money) e.nextElement());
        buffer.append("}");
        return buffer.toString();
    }
}

```

```
}  
  
public interface IMoney {  
    IMoney add(IMoney m);  
    IMoney addMoney(Money m);  
    IMoney addMoneyBag(MoneyBag s);  
    boolean isZero();  
}
```

Outputnya adalah:

```
[12 CHF] ditambah [14 CHF] adalah: [26 CHF]  
true  
[12 CHF] + [7 USD] = [12 CHF][7 USD]  
[14 CHF] + {[12 CHF][7 USD]} = {[26 CHF][7 USD]}  
{[12 CHF][7 USD]} + [14 CHF] = {[26 CHF][7 USD]}  
{[12 CHF][7 USD]} + {[14 CHF][21 USD]} = {[26 CHF][28 USD]}
```

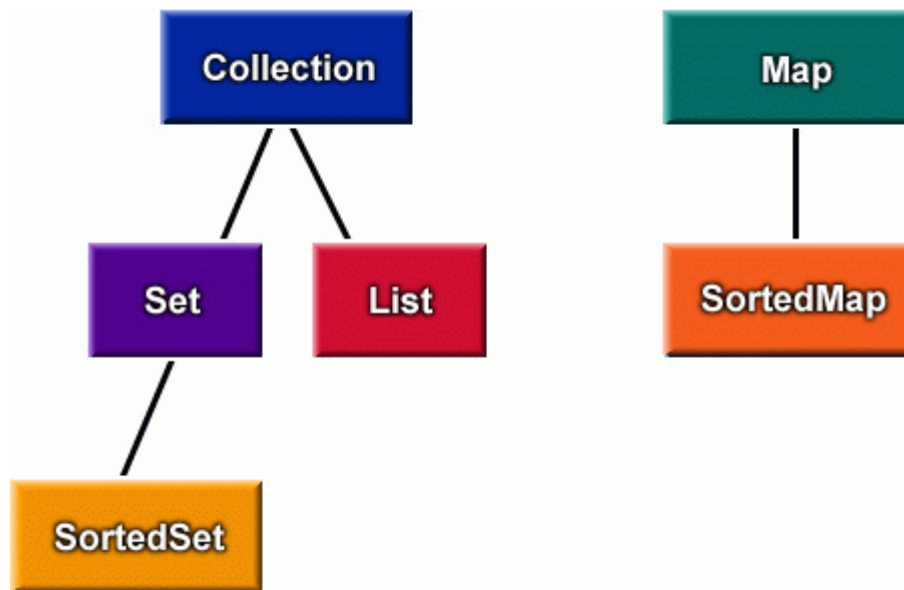
Selanjutnya silahkan anda membuat tes untuk operasi pengurangan dan perkalian dengan bilangan bulat serta implementasinya. (Petunjuk: pengurangan adalah penambahan dengan negasi suatu bilangan). Oh ya `Enumeration` adalah versi pendahulu dari `Iterator` yang akan dijelaskan pada pelajaran tentang `Collection`.

Sejarah tentang `MoneyBag`, ini pertama kali digunakan oleh `WardCunningham` dalam bahasa `SmallTalk`

Bab IX. Belajar Collections

Sering kali dalam suatu program kita tidak hanya berinteraksi dengan satu buah objek dengan tipe tertentu, tapi kita membutuhkan sekumpulan atau himpunan dari objek-objek baik dari tipe yang sama maupun tipe yang berbeda. Dalam Java 2 kumpulan ini diatur dalam Collections Framework.

Sebuah Collections adalah sebuah objek yang mengandung sekumpulan objek yang mempunyai kaitan tertentu. Sekumpulan objek ini biasa disebut elemen-elemen dari Collections. Collections mempunyai operasi untuk memanipulasi objek-objek yang berada di dalamnya. Collections mempunyai interface seperti gambar di bawah ini



A. Tipe-tipe Interface dari Collections

- **Collection:** kumpulan dari elemen-elemen
 - menambah dan menghapus elemen
 - memanggil method `size()`, `isEmpty()`, dan `contains()`
 - memperoleh `Iterator`
- **Set:** kumpulan dari elemen tanpa duplikasi
 - **SortedSet:** seperti `Set`, hanya saja elemen-elemennya tersusun terurut
- **List:** kumpulan terurut dan dibolehkan duplikasi elemen
- **Map:** objek yang memetakan kunci (*key*) ke nilainya dan menyimpan pasangan kunci dan nilai tersebut tanpa ada duplikasi kunci
 - **SortedMap:** seperti `Map`, dengan kunci-kunci dalam susunan terurut

- Ketika kita ingin mengambil objek kembali, tipe referensinya adalah Object
- Sebagian besar implementasinya tidak tersinkronisasi.

Inilah yang disebut dengan *core collections interfaces*.

		Implementasi			
		Hash Table	Resizable Array	Balanced Tree	Linked List
Interface	Set	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

B.Sets

- **ArraySet**: implementasi Set dengan resizable array
 - Digunakan untuk himpunan yang kecil dan sering dibuat ataupun dimusnahkan.
 - Cepat untuk dibuat dan dilintasi untuk himpunan yang kecil dan lambat untuk himpunan yang besar
- **HashSet**: a HashMap implementation of Set
 - For general use
- **TreeSet**: a TreeMap implementation of SortedSet
 - Sorted in natural ordering

C.Lists

- **List**: a Collection of ordered elements
 - get/set element at index
 - insert/delete element at index
 - get a `ListIterator` (traverse forward/backward, insert/remove in middle)
 - get a sublist
- **ArrayList**: resizable array implementation of List
 - random access yang cepat
 - Penyisipan dan penghapusan yang lamban
 - Kalau kita banyak menggunakan iterasi, gunakan `ListIterator`.
- **LinkedList**: linked list implementation of List
 - Slow random access

- Fast insertion and deletion
- Has add/get/remove of First() and Last() elements (stacks, queues, dequeues)
- Vector: retrofitted - use ArrayList

D. Maps

- Map: maintains key/value pairs
- Can look up a value with a key
- ArrayMap: ArrayList implementation of Map
 - Fast creation/iteration for small maps
 - Slow for large maps
- HashMap: implementasi Map dengan hash table
 - Untuk penggunaan secara umum
 - Dapat diubah load factornya sehingga meningkatkan unjuk kerja
- TreeMap: red-black tree implementation of Map
 - Sorted in *natural ordering*
 - Elemen-elemennya merupakan implementasi Comparable (mempunyai method `int compareTo(Object);`)
- Hashtable: telah disesuaikan menjadi implementasi dari Map yang tersinkronisasi dengan menggunakan HashMap

Agar lebih mudah memahami kita langsung mencobanya dengan contoh yang sederhana:

```
import java.util.*;

public class intro1 {

    public static void main(String args[]) {

        List list = new ArrayList();

        for (int i = 1; i <= 10; i++)

            list.add(i + " * " + i + " = " + i * i);

        Iterator iter = list.iterator();

        while (iter.hasNext())

            System.out.println(iter.next());

    }

}
```

Keluarannya adalah:

```
1 * 1 = 1
```

```
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
```

ArrayList disini adalah salah satu implementasi dari List tentu saja kita bisa menggantinya dengan yang lain seperti dengan Vector.

kita coba saja dan coba lihat dan bandingkan hasil keluarannya,

```
package learn.collection;

import java.util.*;

public class intro2 {

    public static void main(String args[]) {

        List list = new Vector();

        for (int i = 1; i <= 10; i++)

            list.add(i + " * " + i + " = " + i * i);

        Iterator iter = list.iterator();

        while (iter.hasNext())

            System.out.println(iter.next());

    }

}
```

Dengan menggunakan interface maka kita tidak terlalu terikat pada implementasi.

Selanjutnya contoh yang kedua

```
import java.util.*;

public class PrintingCollection {

    static Collection fill(Collection c) {

        c.add("dog");

        c.add("dog");

        c.add("cat");

        return c;

    }

    static Map mfill(Map m) {

        m.put("dog", "Bosco");

        m.put("dog", "Spot");

    }

}
```

```
        m.put("cat", "Rags");
        return m;
    }

    public static void main(String[] args) {
        System.out.println(fill(new ArrayList()));
        System.out.println(fill(new HashSet()));
        System.out.println(mfill(new HashMap()));
    }
}
```

Keluarannya adalah

[dog, dog, cat]

[cat, dog]

{cat=Rags, dog=Spot}

Collection framework mempunyai utility class yang baik yaitu Collections dan Arrays. Ini adalah salah satu contoh penggunaannya untuk mengurutkan:

```
import java.util.*;

public class sort1 implements Comparator {
    public int compare(Object o1, Object o2) {
        String s1 = (String)o1;
        String s2 = (String)o2;
        return s1.toLowerCase().compareTo(s2.toLowerCase());
    }
}

public static void main(String args[]) {
    List list = new ArrayList();
    list.add("abc");
    list.add("DEF");
    list.add("ghi");
    // standard sort
    Collections.sort(list);
    Iterator iter = list.iterator();
    while (iter.hasNext())
        System.out.println(iter.next());
    // sort, ignoring case
    Collections.sort(list, new sort1());
    iter = list.iterator();
    while (iter.hasNext())
        System.out.println(iter.next());
}
```

```
    }  
}
```

Keluarannya adalah:

```
DEF  
abc  
ghi  
abc  
DEF  
ghi
```

Jika implementasi di atas menggunakan ArrayList contoh berikut menggunakan HashMap:

```
import java.util.*;  
  
public class map1 {  
    public static void main(String args[]) {  
        Map hm = new HashMap();  
        hm.put("Mary", "123-4567");  
        hm.put("Larry", "234-5678");  
        hm.put("Mary", "456-7890");  
        hm.put("Felicia", "345-6789");  
        Iterator iter = hm.entrySet().iterator();  
        while (iter.hasNext()) {  
            Map.Entry e = (Map.Entry)iter.next();  
            System.out.println(e.getKey() + " "  
                + e.getValue());  
        }  
    }  
}
```

Perhatikan bahwa setiap entry dari map mempunyai dua nilai, yang satu adalah key (kunci) satunya lagi adalah nilai dari key tersebut. Sebuah map tidak boleh mengandung key yang terduplikasi, juga elemen-elemennya tidak terurut. Keluaran dari program di atas adalah:

```
Larry 234-5678  
Mary 456-7890  
Felicia 345-6789
```

Contoh untuk Set:

```
import java.util.*;  
  
public class set1 {  
    public static void main(String args[]) {  
        Set hs = new HashSet();
```

```
        hs.add("1");
        hs.add("2");
        hs.add("2");
        hs.add("3");

        Iterator iter = hs.iterator();

        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```

Keluarannya:

```
3
2
1
```

Contoh untuk SortedMap:

```
import java.util.*;

public class map2 {

    public static void main(String args[]) {

        SortedMap m = new TreeMap();

        m.put("Sneezy", "common cold");

        m.put("Sleepy", "narcolepsy");

        m.put("Grumpy", "seasonal affective disorder");

        System.out.println(m.keySet());

        System.out.println(m.values());

        System.out.println(m.entrySet());

    }

}
```

Perhatikan bagaimana urutan keluarannya:

```
[Grumpy, Sleepy, Sneezy]
[seasonal affective disorder, narcolepsy, common cold]
[Grumpy=seasonal affective disorder, Sleepy=narcolepsy, Sneezy=common cold]
```

Pada contoh-contoh sebelumnya kita telah melihat bagaimana melakukan iterasi dalam sebuah collection yaitu dengan menggunakan Iterator. Hanya saja dengan menggunakan iterator kita tidak dapat begitu saja mengubah collection pada saat kita melakukan iterasi sebab akan melontarkan `ConcurrentModificationException`. Dengan iterator kita juga belum bisa melakukan iterasi mundur. Untuk itu kita bisa menggunakan `ListIterator`

```
import java.util.*;

public class iterator1 {
```

```
public static void main(String args[]) {  
    List list = new ArrayList();  
    list.add("abc");  
    list.add("def");  
    list.add("ghi");  
    ListIterator iter1 = list.listIterator(list.size());  
    boolean first = true;  
    while (iter1.hasPrevious()) {  
        System.out.println(iter1.previous());  
        //list.add("xyz");  
        if (first) {  
            iter1.add("xyz");  
            first = false;  
        }  
    }  
}
```

Outputnya adalah seperti berikut ini:

```
ghi  
xyz  
def  
abc
```

1. Tips pemrograman dan perancangan API

Ada beberapa aturan yang membuat sebuah program berorientasi objek menjadi efektif. Hal-hal di bawah ini tidak hanya berlaku untuk collection. Aturan pertama adalah memprogram dalam interface bukan pada satu jenis implementasi. Contohnya:

```
List lst = new ArrayList();
```

lebih baik jika dibandingkan dengan:

```
ArrayList lst = new ArrayList();
```

karena menyebabkan tidak terlalu terikat pada implementasi tertentu. Suatu saat, `ArrayList` mungkin diganti menjadi `LinkedList`, atau sebuah program mungkin bergantung pada method yang hanya ada di `ArrayList` dan tidak ada di interface `List`.

Aturan lain, terutama apabila memungkinkan, ketika sebuah method memerlukan parameter masukan sebaiknya menggunakan tipe yang lebih umum atau interfacenya. Misal, `Collection` bukannya `List`, atau `List` bukannya `ArrayList`. Ini menyebabkan low-coupling antarobjek.

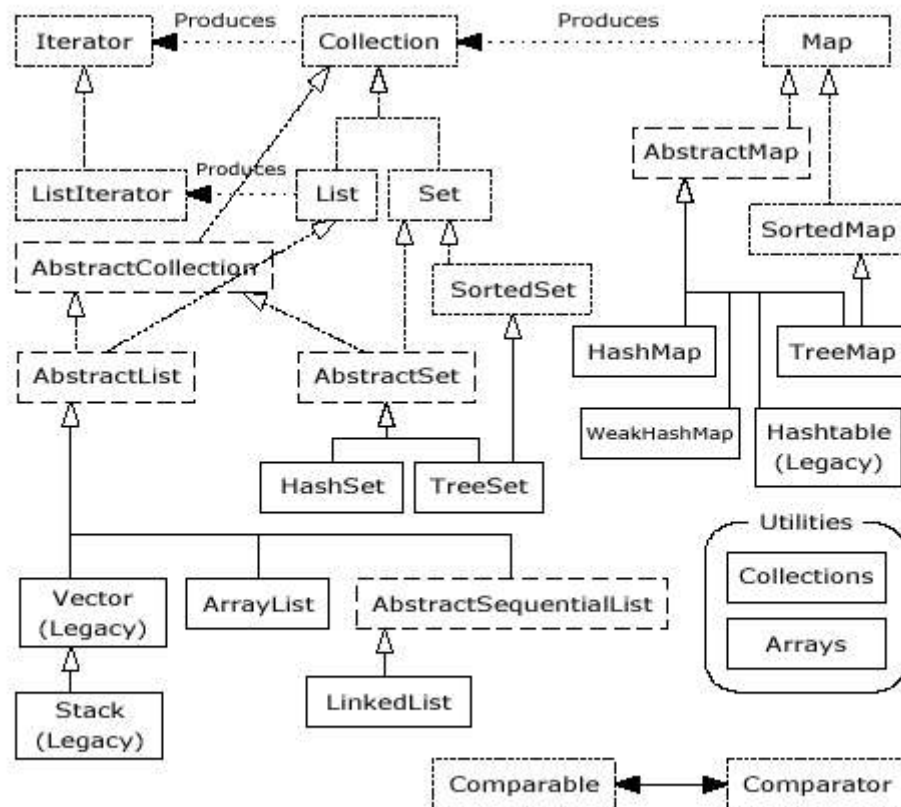
Untuk tipe nilai kembalian dari satu method, yang berlaku adalah sebaliknya, yaitu menggunakan tipe implementasi yang spesifik. Contohnya, `ArrayList` mempunyai performansi yang berbeda dengan `LinkedList` sementara `SortedMap` mempunyai

kemampuan yang lebih daripada `Map`, oleh karena itu pemakai memerlukan tipe struktur data jenis apa yang dikembalikan oleh method yang dia panggil. Tipe kembalian ini dapat di-*upcast* ke tipe yang lebih umum seperti contoh ini:

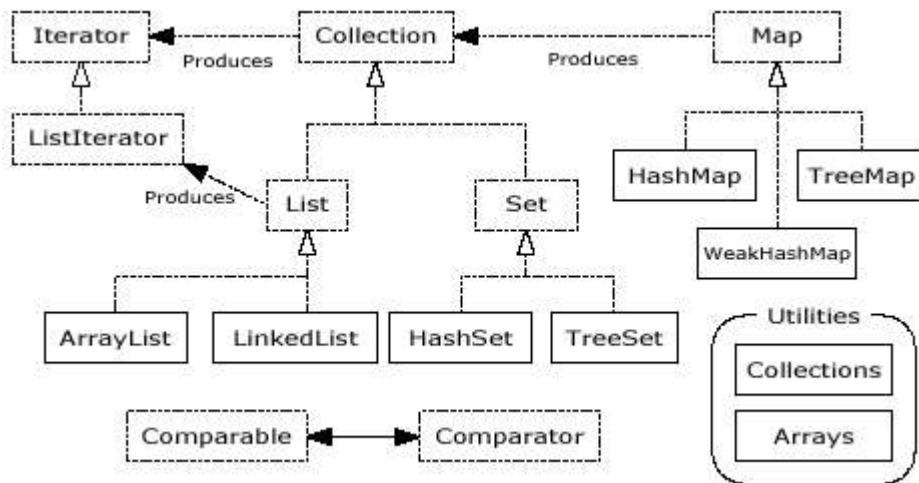
```
ArrayList f() {...}

void g() {
    List lst = f();
}

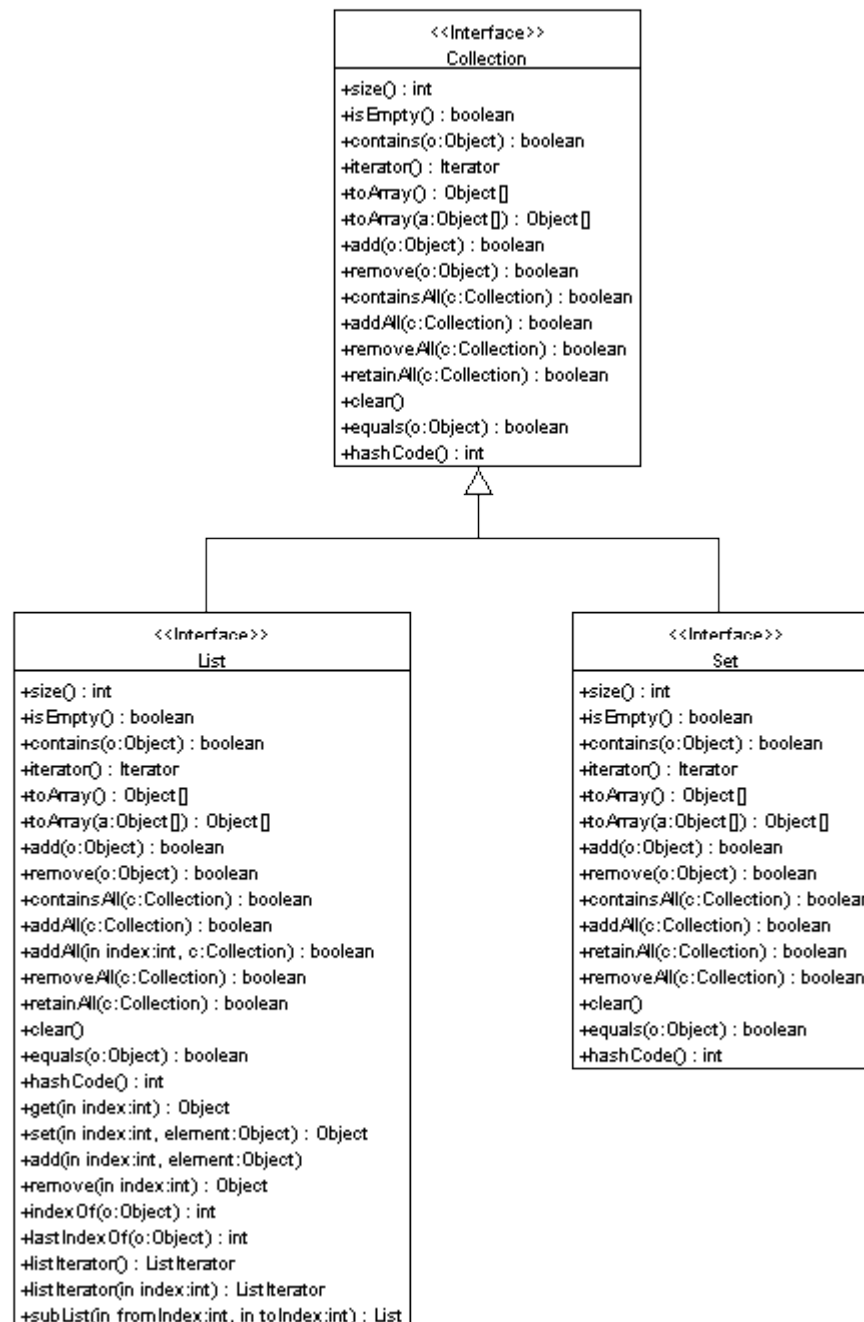
```

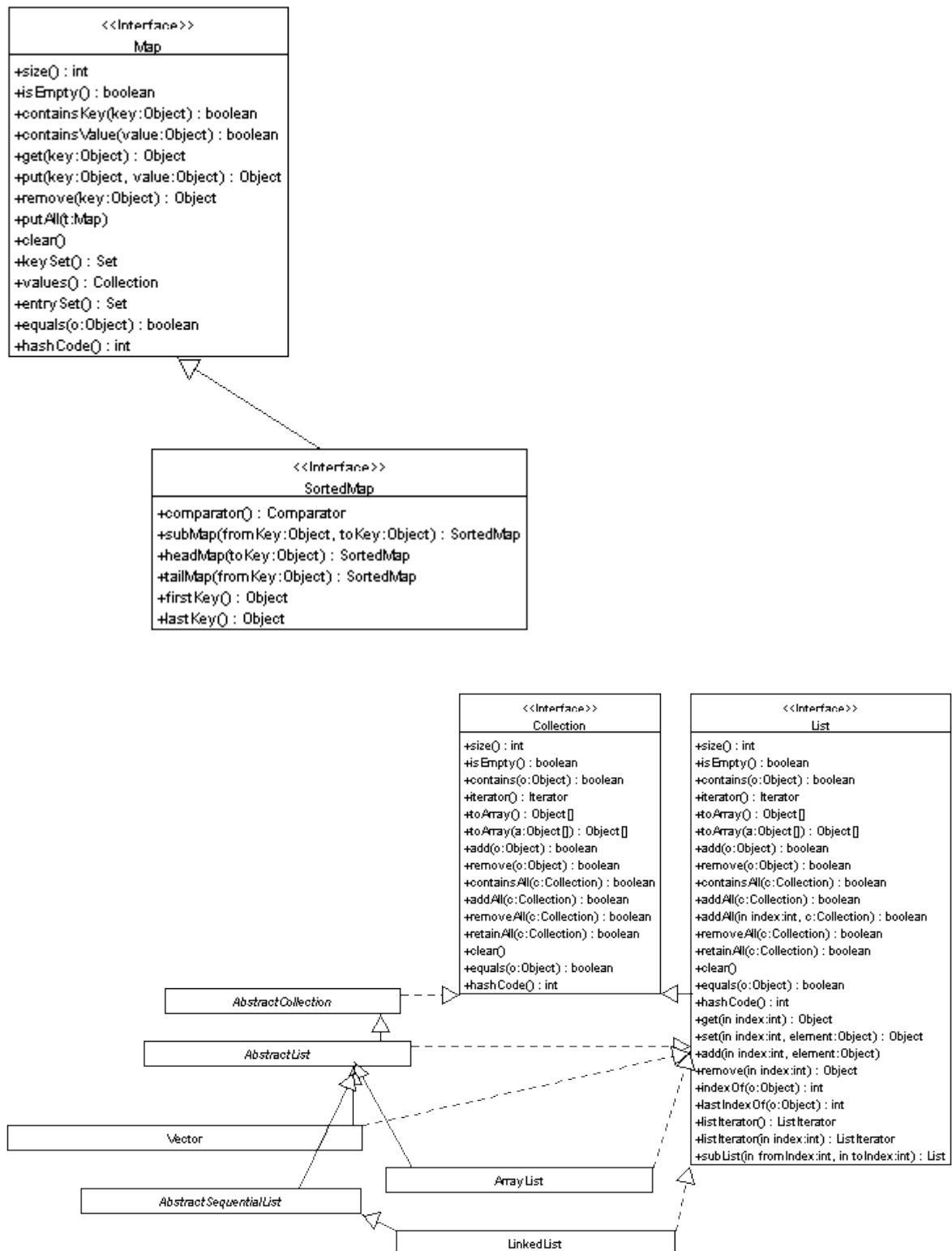


Gambar di atas adalah gambar collections framework secara umum. Mulai Java 2, Vector dan Hashtable sudah disesuaikan sehingga termasuk ke dalam framework dan merupakan implementasi yang tersinkronisasi walaupun masih mengandung beban operasi-operasi yang lama. Tapi selama kita memrogram ke dalam interface kita akan dengan mudah berganti implementasi apabila menemui kesulitan. Saran saya sebaiknya hanya gunakan framework yang standar seperti gambar di bawah ini dan nanti bisa mengeksplorasi lebih jauh lagi apabila menemui hal-hal baru.



UML Class Diagram dari Core Collection Interface



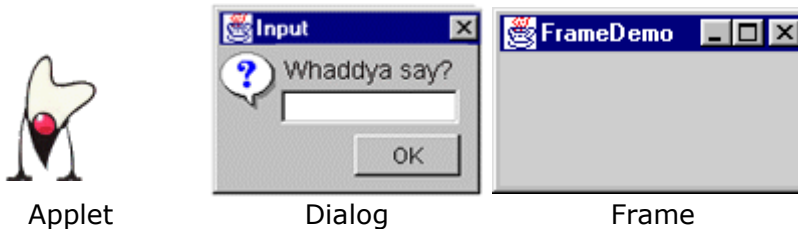


Bab X.Component Swing dan Layout Manager

A.Kontainer dan Komponen

B.Top-Level Containers

Komponen-komponen yang berada pada puncak setiap hirarki aplikasi Swing.

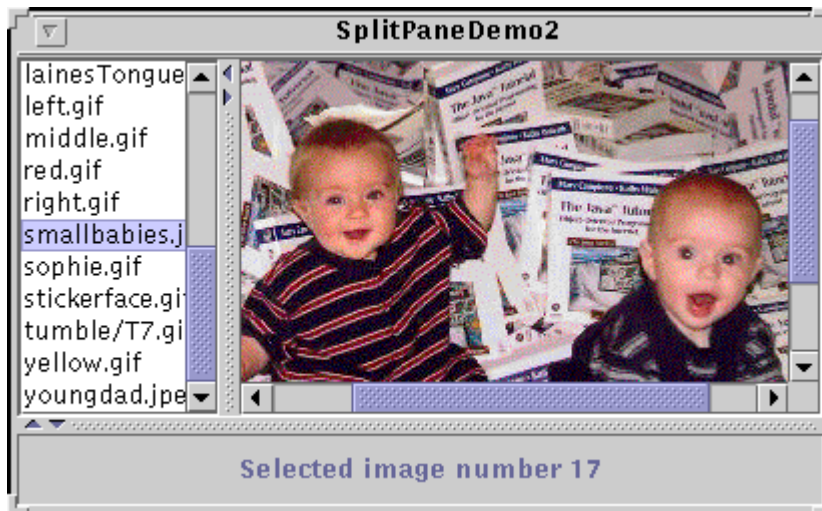
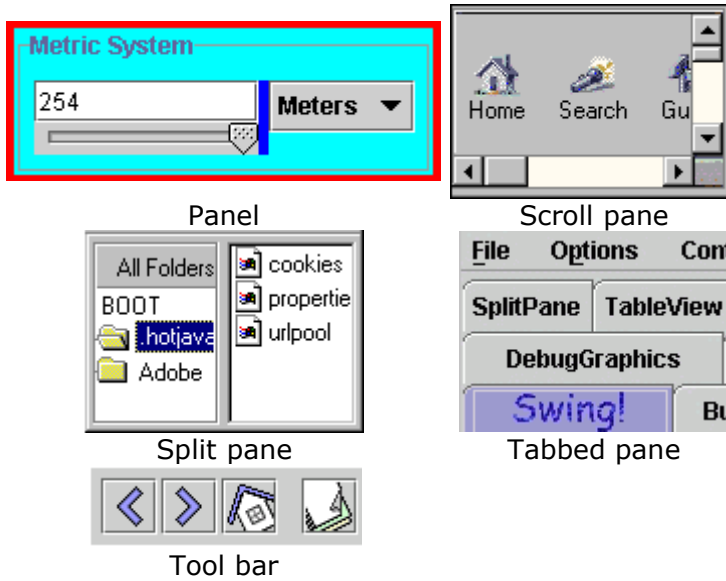


Tampilannya bergantung pada platform di mana dia berada sehingga disebut *heavyweight component*



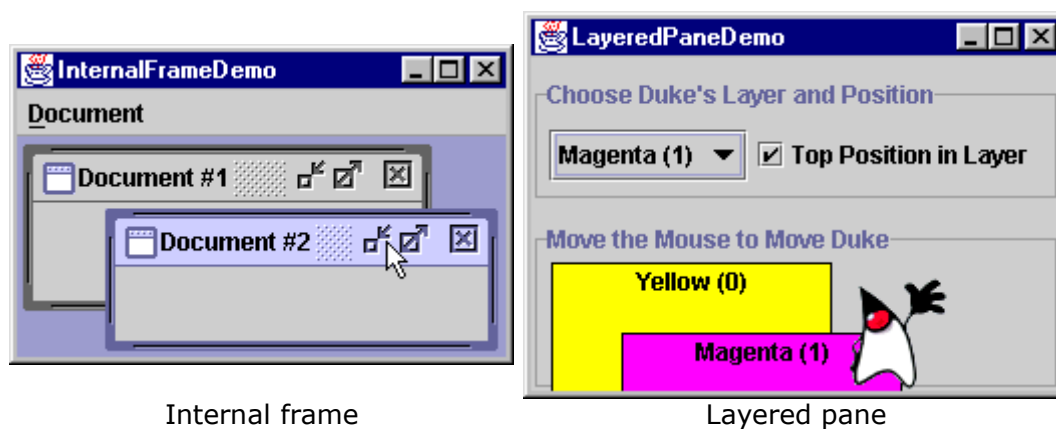
General-Purpose Containers

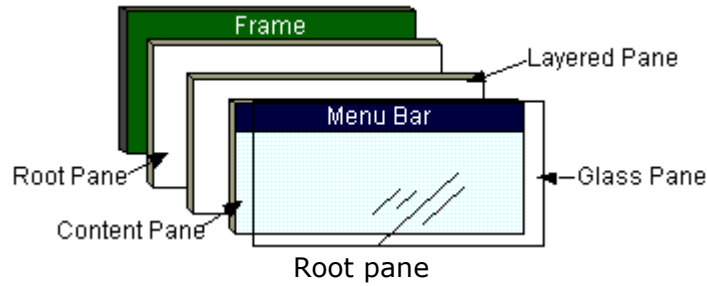
Kontainer tingkat menengah yang dapat digunakan pada banyak situasi dan keadaan yang berbeda.



Special-Purpose Containers

Kontainer antara yang mempunyai tugas khusus dalam User Interface.



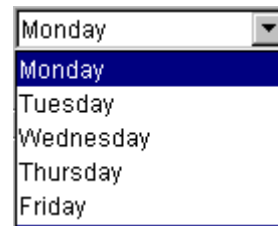


Basic Controls

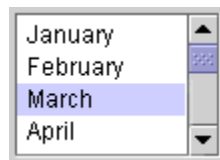
Komponen-komponen yang berguna untuk memperoleh masukan dari pemakai, biasanya menunjukkan keadaan tertentu yang sederhana.



Buttons



Combo box



List



Menu



Slider



Text fields

Uneditable Information Displays

Komponen-komponen yang berfungsi hanya untuk memberi informasi kepada pemakai.



Label



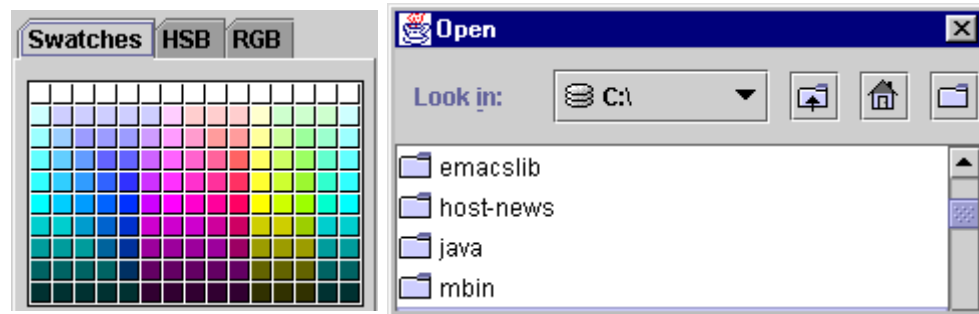
Progress bar



Tool tip

Editable Displays of Formatted Information

Komponen-komponen yang menampilkan informasi yang tersusun rapih dan bisa dipilih atau diedit oleh pemakai.



Color chooser

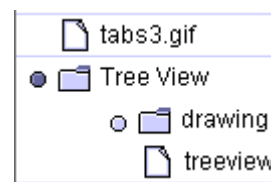
File chooser

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

Table

Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

Text



Tree

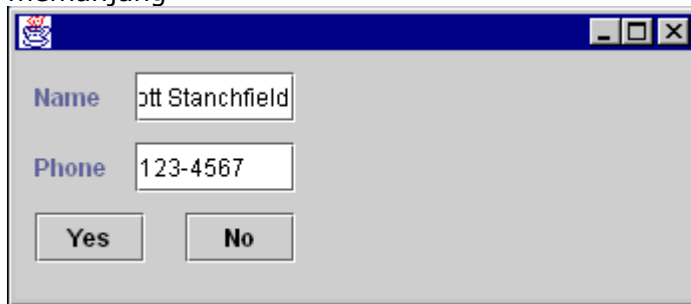
C.Layout Management yang Efektif

Mengapa kita perlu belajar mengelola tataletak?
Perhatikan baik-baik contoh kasus berikut:

Pembuat program tampaknya lupa memperhitungkan bahwa panjang nama seseorang bisa melebihi lebar kotak isian yang disediakan



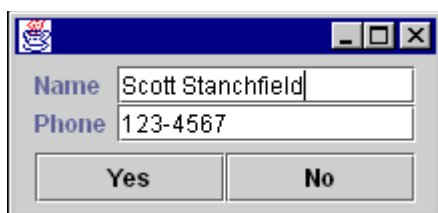
Ketika pemakai bermaksud untuk memperpanjang kotak isian ternyata dia tidak ikut memanjang



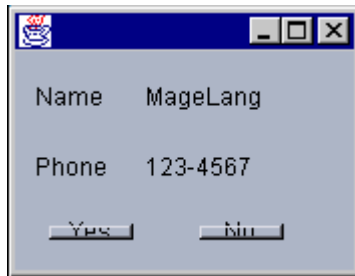
Seharusnya ketika pemakai mengalami kesulitan seperti ini,



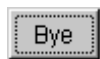
yang perlu dia lakukan hanyalah melebarkan jendelanya ke samping, sehingga tampak seperti



Komponen memang telah dirancang untuk Look-And-Feel atau ukuran Font tertentu
Dalam Look-and-Feel Motif (Solaris)



Komponen yang dirancang untuk i18n (menggunakan bahasa lain Jepang atau Arab misalnya ☺)



Tombol sederhana untuk mengakhiri sebuah aplikasi. Pemrogramnya tidak menyadari bagaimana translasinya ke bahasa lain



...seperti bahasa German. *Auf Wiedersehen* terpenggal menjadi lebih ringkas lagi yaitu *Wieder*.

Ini sangat disayangkan untuk dua alasan:

- Pengguna tidak melihat frase keseluruhan, bahkan lebih parah lagi...
- ..."wieder" berarti *again*(sekali lagi).

1.Apa itu Layout Manager?

Setiap container pasti mempunyai sebuah layout manager -- sebuah objek yang mengimplementasikan interface `LayoutManager`. Jika layout manager semula tidak memenuhi kebutuhan kita, ganti saja dengan yang lebih cocok. Java menyediakan layout manager dari yang sangat sederhana (`FlowLayout` dan `GridLayout`) ke yang lebih khusus (`BorderLayout` dan `CardLayout`) sampai dengan yang paling luwes (`GridBagLayout` dan `BoxLayout`).

- `BorderLayout`
- `FlowLayout`
- `BoxLayout`
- `GridLayout`
- `CardLayout`
- `GridBagLayout`
- `null` (tanpa layout)
- `AbsoluteLayout` (NetBeans/Forte)

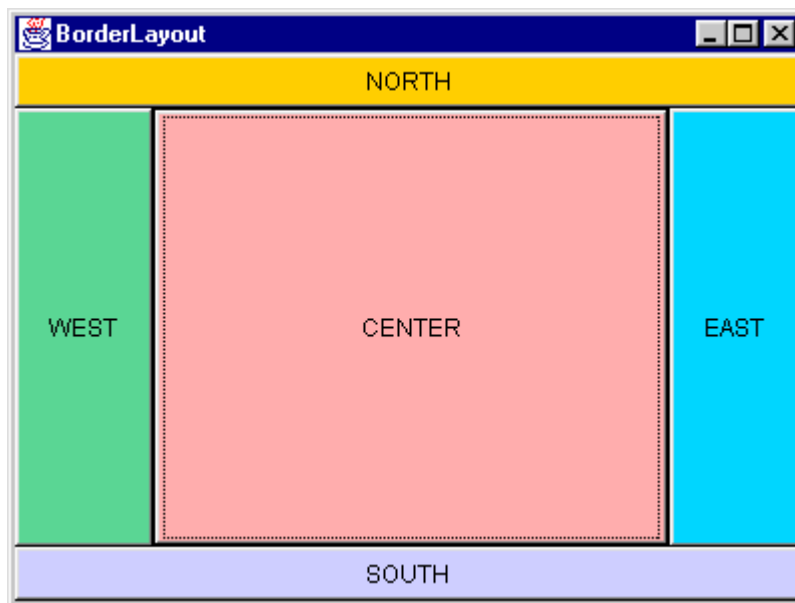
2.BorderLayout

`BorderLayout` menyusun komponen-komponen dari sebuah kontainer dalam wilayah-wilayah yang dinamai North, South, East, West, dan Center.

- Komponen-komponen di North dan South diberikan tinggi yang sesuai dan meregang sepanjang lebar seluruh kontainer.
- Komponen-komponen di East dan West diberikan lebar yang sesuai dan meregang secara vertikal untuk memenuhi ruang antara wilayah North dan South.
- Komponen-komponen yang terletak di Center mengembang untuk mengisi ruang yang tersisa.

Dalam Java AWT (Abstract Window Toolkit), semua jendela (termasuk frame dan dialog box) menggunakan `BorderLayout` secara default.

Misal

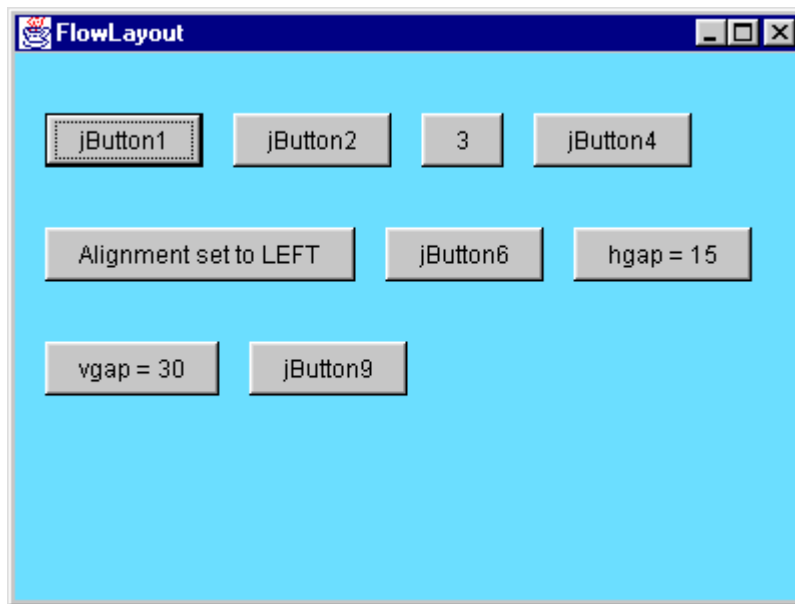


`BorderLayout` sangat baik untuk memaksa komponen-komponen menempel pada sisi-sisi kontainer dan untuk memenuhi bagian tengah dari kontainer dengan sebuah komponen.

3.FlowLayout

`FlowLayout` menyusun komponen-komponen dalam baris-baris dari kiri ke kanan dan kemudian dari atas ke bawah menggunakan ukuran alami setiap komponen, `preferredSize`. `FlowLayout` membariskan sebanyak mungkin komponen yang dia mampu ke dalam satu baris, baru kemudian berpindah ke baris baru. Biasanya, `FlowLayout` digunakan untuk menyusun tombol-tombol pada sebuah panel. Dalam Java AWT, semua all panel (termasuk applet) menggunakan `FlowLayout` secara default.

Contoh



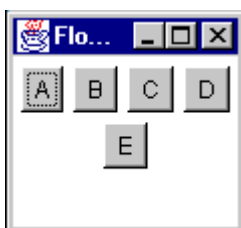
Anda dapat memilih apakah komponen-komponen dalam satu baris itu tersusun rata-kiri, rata-kanan, atau berada di tengah-tengah baris dengan menset alignment justification dari FlowLayout `left`, `right`, atau `center`. Anda juga dapat menentukan besar pemisahan (horizontal dan vertical spacing) antar komponen komponen dan baris. Gunakan Component Inspector untuk mengubah kedua properti `alignment` dan `gap` ketika anda berada pada GUIEditing workspace.

Visualisasi sifat-sifat FlowLayout

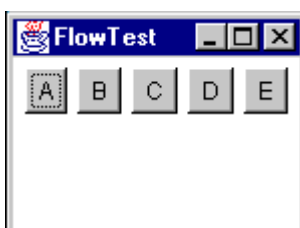
Ukuran awal



kemudian diperluas sedikit secara horisontal



kemudian lebih diperluas lagi



Perhatikan bagamaimana layout berusaha menempatkan sebanyak mungkin komponen per barisnya. Jika tidak ada ruang untuk seluruh komponen, mereka tidak diperlihatkan:



4.BoxLayout

Layout manager yang mengizinkan banyak komponen untuk diletakkan baik secara vertikal maupun horisontal. Komponen-komponen tersebut tidak akan tergulung sehingga, misalnya, sebuah susunan vertikal dari komponen-komponen akan tetap tersusun vertikal ketika ukuran frame diubah.

BoxLayout berusaha untuk menyusun komponen-komponen pada lebar alami mereka (dalam konfigurasi X axis) atau tingginya (dalam konfigurasi Y axis). Untuk konfigurasi X axis, jika tidak semua tinggi komponen sama, BoxLayout berusaha untuk membuat semua komponen mempunyai tinggi seperti komponen yang ukurannya tertinggi. Jika ini tidak mungkin untuk komponen tertentu, maka BoxLayout mensejajarkannya secara vertikal, menurut komponen alignment Y. Secara default, sebuah komponen mempunyai alignment Y sebesar 0.5, yang berarti mereka sejajar pada garis tengah komponen secara horisontal. Begitu juga untuk layout dalam Y axis, BoxLayout berusaha membuat semua komponen pada ukuran alaminya tetapi ukuran kontainer akan mengikuti ukuran komponen yang terlebar (pada konfigurasi Y axis, tingginya); jika itu tidak berhasil maka, komponen disejajarkan horisontal menurut alignment X -nya.

contoh BoxLayout dengan konfigurasi X axis dimana default pensejajaran vertikal alignment Y adalah 0.5



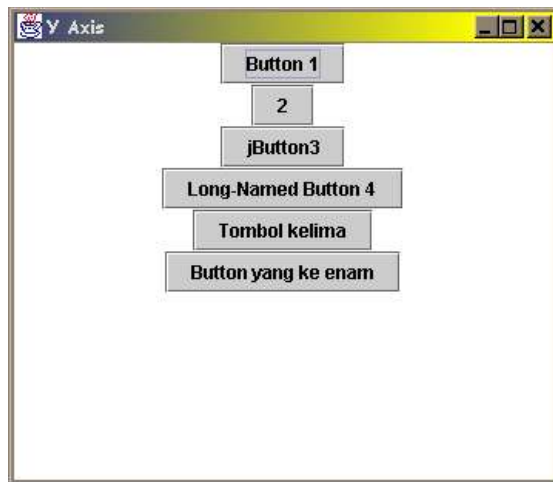
perhatikan apabila kita persempit secara horisontal maka komponen tidak akan pindah baris.



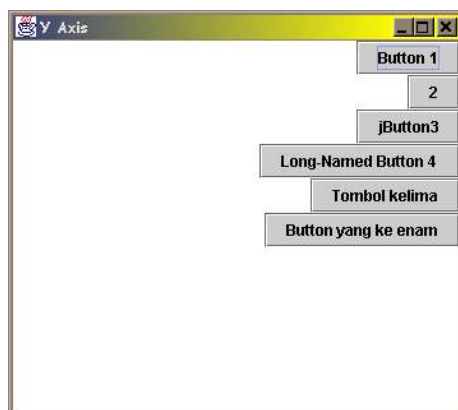
Untuk konfigurasi Y axis awalnya adalah rata kiri, alignment X setiap komponen adalah 0.0



Apabila alignment X setiap komponen diubah menjadi 0.5 hasilnya seperti di bawah ini



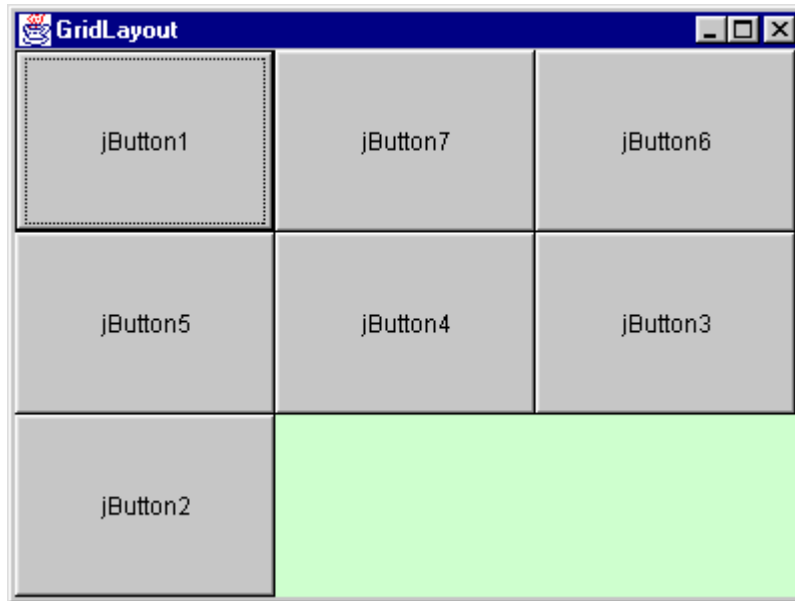
Untuk membuat rata kanan, alignment X setiap komponen tinggal diubah menjadi 1.



5.GridLayout

`GridLayout` menempatkan komponen ke dalam kotak-kotak sel dalam baris dan kolom. `GridLayout` memperbesar setiap komponen untuk memenuhi besar ruang yang tersedia di dalam satu sel. Setiap sel mempunyai ukuran yang tepat sama dan kotak-kotaknya seragam. Ketika kita mengubah ukuran kontainer `GridLayout`, `GridLayout` mengubah ukuran sel sebesar mungkin untuk memenuhi ruang yang tersedia pada kontainer.

contoh

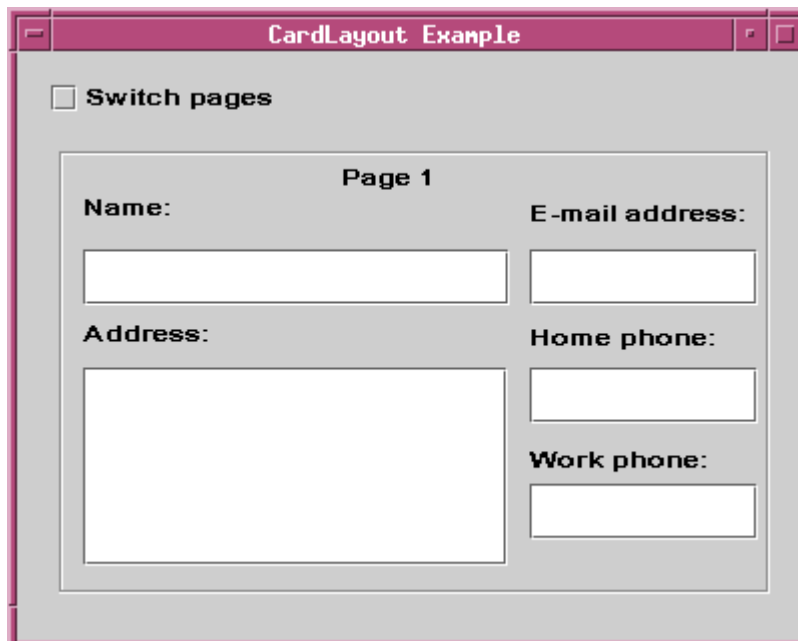


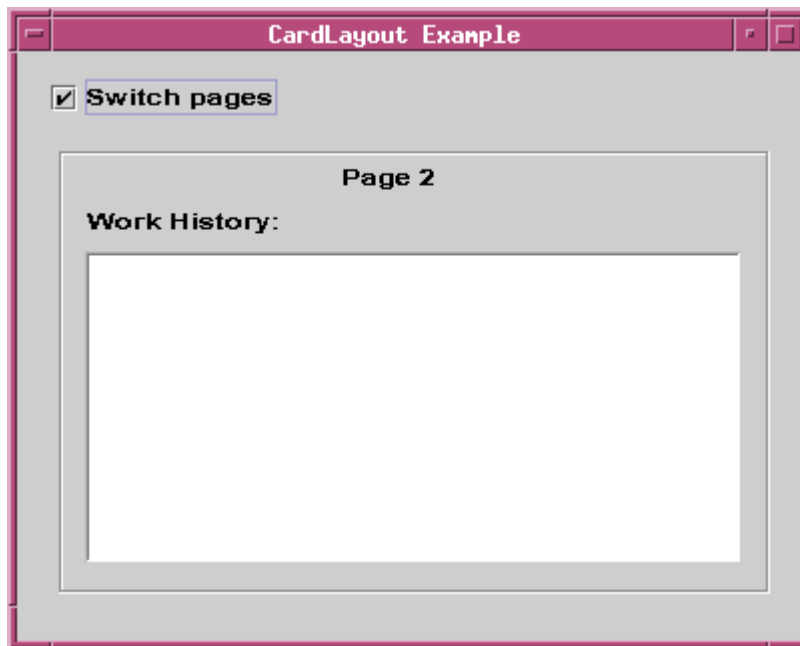
Gunakan `GridLayout` jika anda merancang kontainer dimana komponen-komponennya mempunyai ukuran yang sama seperti misalnya tombol-tombol kalkulator.

6. CardLayout

`CardLayout` menempatkan komponen-komponen (biasanya panel) satu di atas yang lainnya seperti tumpukan kartu. Kita hanya bisa melihat satu dalam satu waktu, dan anda dapat mengganti ke panel yang lain dengan menggunakan kontrol lain yang memilih panel mana yang berada di sebelah atas.

Contoh:





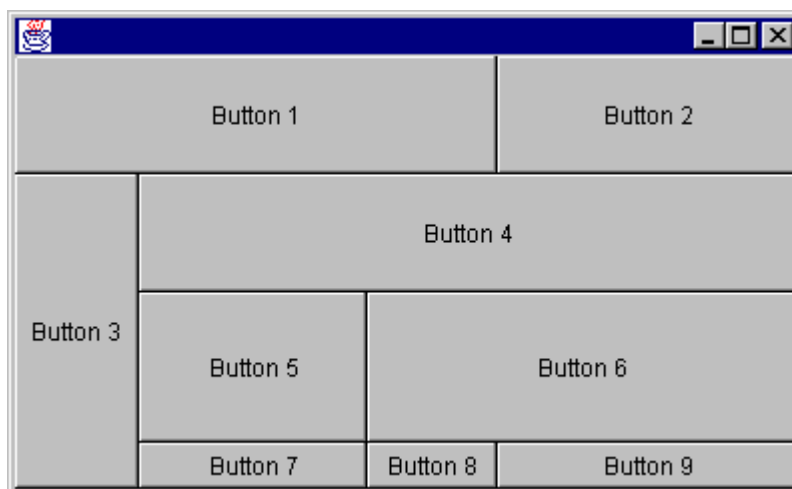
`CardLayout` merupakan layout sesuai untuk daerah yang mempunyai komponen berbeda pada saat yang berbeda pula. Hal ini membuat kita bisa mengatur dua atau lebih panel yang perlu menempati tempat yang sama.

`CardLayout` biasanya dihubungkan dengan komponen pengontrol seperti sebuah `checkbox` atau sebuah `list`. Keadaan dari komponen pengontrol menentukan komponen mana yang akan ditampilkan oleh `CardLayout`. Pengguna membuat pilihan dengan mengubah keadaan komponen pengontrol pada UI.

7.GridBagLayout

`GridBagLayout` adalah layout yang sangat luwes dan ampuh karena dengannya kita bisa mengatur dengan bebas tata letak komponen pada kotak-kotak sel dibandingkan dengan `GridLayout`. `GridBagLayout` menempatkan komponen-komponen secara horisontal dan vertikal pada kotak-kotak yang dinamis. Komponen-komponen tersebut tidak harus mempunyai ukuran yang sama, dan mereka dapat mengisi kotak lebih dari satu sel.

Contoh :



`GridBagLayout` menentukan penempatan komponen-komponennya berdasarkan batas-batas dan ukuran minimum dari setiap komponen, ditambah dengan ukuran alami kontainer.

Meskipun `GridBagLayout` dapat mengakomodasi grid yang kompleks, ia akan lebih baik (dan lebih mudah ditebak perilakunya) apabila kita mengatur komponen-komponen ke dalam panel-panel yang lebih kecil yang ditampung di dalam kontainer `GridBagLayout`. Panel-panel ini dapat menggunakan layout-layout yang lain atau bahkan mengandung panel-panel lagi. Metode ini mempunyai dua keuntungan:

- Kita dapat mengatur penempatan dan ukuran dari sebuah komponen dengan lebih natural karena menggunakan layout tertentu.
- Sel yang digunakan jadi lebih sedikit, dan membuat `GridBagLayout` menjadi lebih sederhana dan mudah dikontrol.

8.nullLayout

Layout `null` berarti tidak ada layout manager yang diset untuk kontainer yang kita pakai. `null layout` (Swing) serupa dengan `AbsoluteLayout` (NetBeans) dalam hal kita dapat meletakkan komponen pada container dengan koordinat tertentu dan mengubah ukuran komponen dengan mouse. Null Layout digunakan untuk merancang form tanpa layout manager sama sekali.

Kekuatan dan kelemahannya sama dengan `AbsoluteLayout`. Ia berguna untuk membuat prototipe GUI secara cepat tapi bukan untuk aplikasi akhir karena alasan-alasan yang telah dijelaskan di atas.

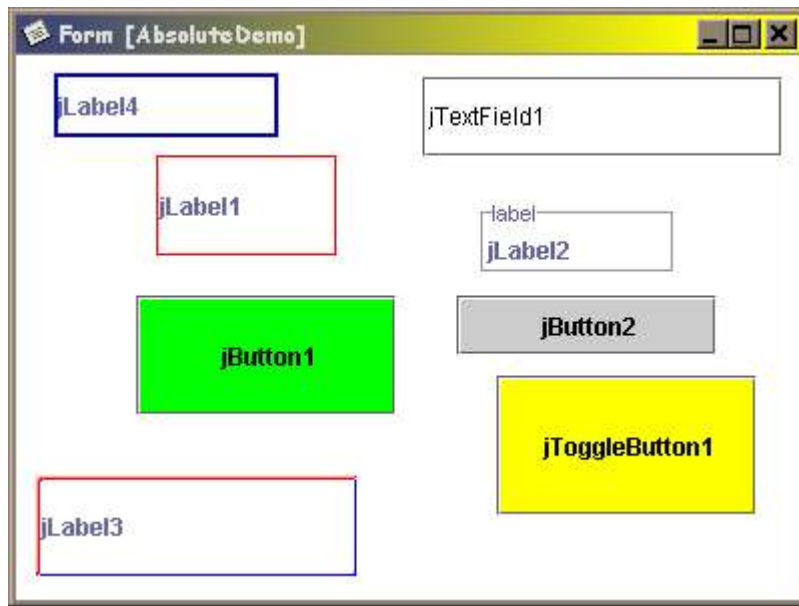
Jika anda menggunakan Null Layout untuk to rancangan akhir, pertimbangkan untuk pindah ke `GridBagLayout` dan memperhalusnya sebelum didistribusikan. Jika anda tetap menggunakan Null Layout, anda harus menset secara programatis ukuran kontainer. Apabila tidak diset, maka kontainer akan ditampilkan pada ukuran minimum pada awalnya. Salah satu cara untuk menetapkan ukuran kontainer adalah dengan mengeset atribut Form Size Code Generation dari form.

9.AbsoluteLayout

This is a feature of Netbeans yeah!!!.

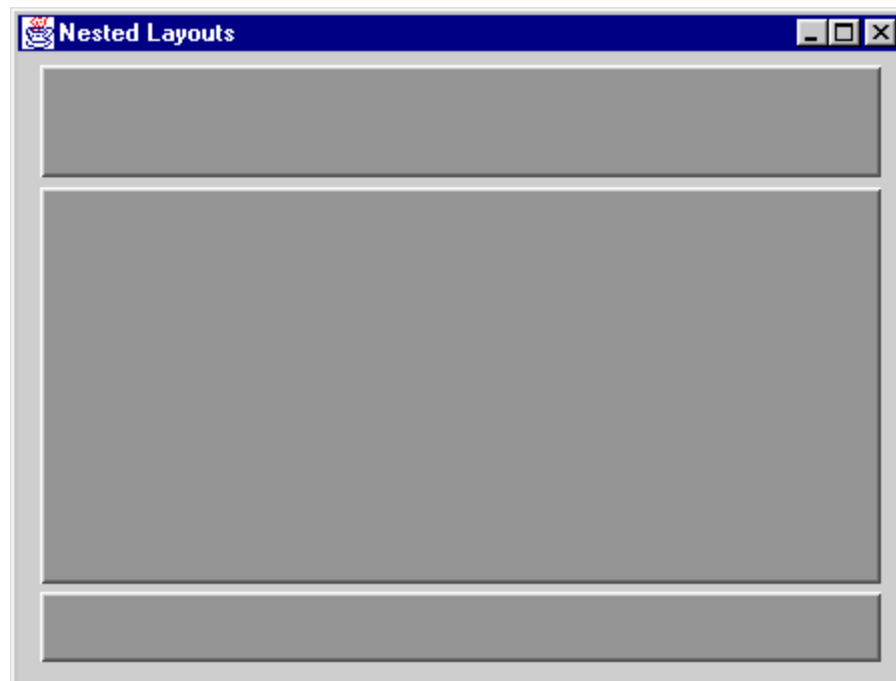
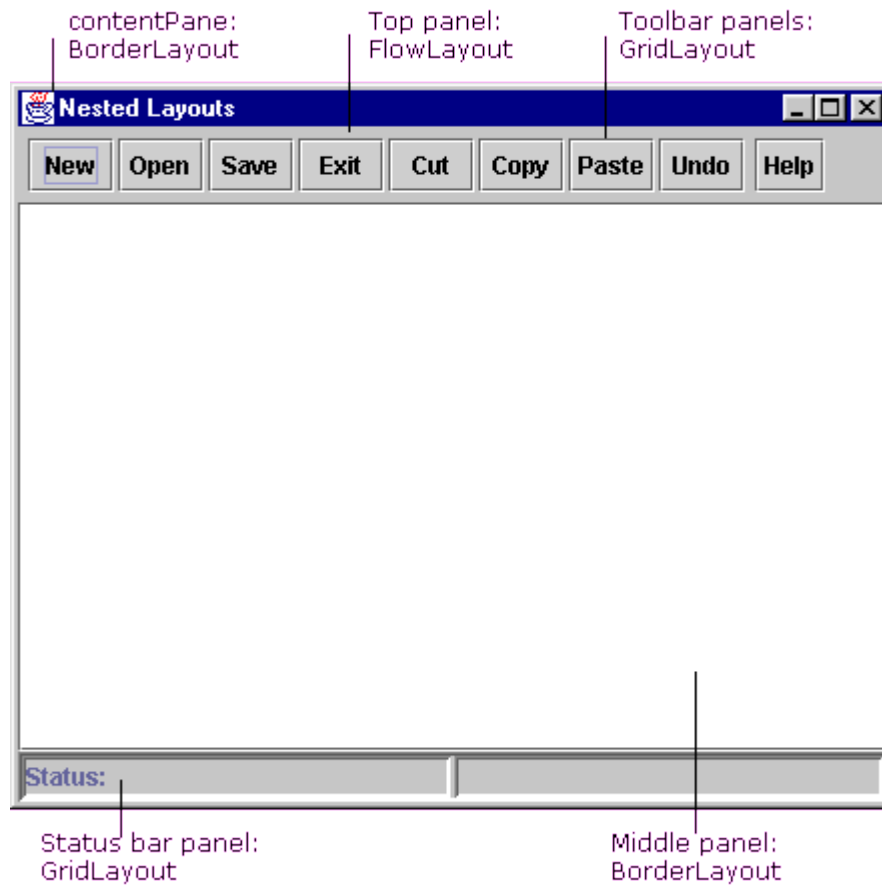
Kelebihan `AbsoluteLayout` manager jika dibandingkan dengan Null Layout adalah anda tidak perlu mengeset secara programatis ukuran dari kontainer. Dengan asumsi bahwa Form Size Policy dari kontainer (atribut Code Generation) diset pada defaultnya, `Generate pack()`. Jika dia diset pada `No Resize Code`, maka kontainer ditampilkan pada ukuran minimum apapun jenis layout managernya, dan tidak ada komponen yang ditampakkan sampai pemakai mengubah ukuran form.

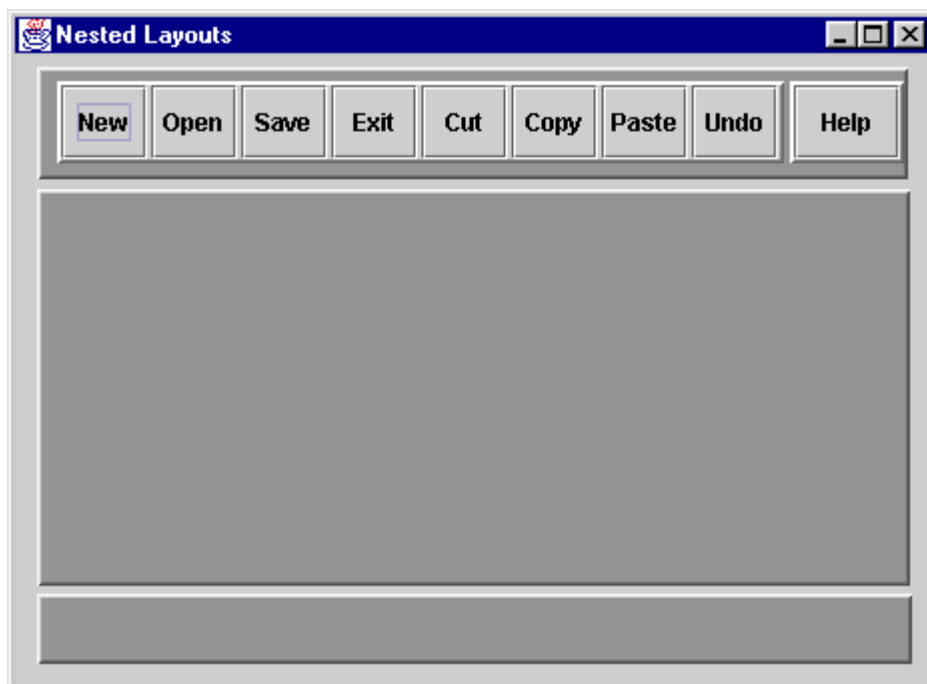
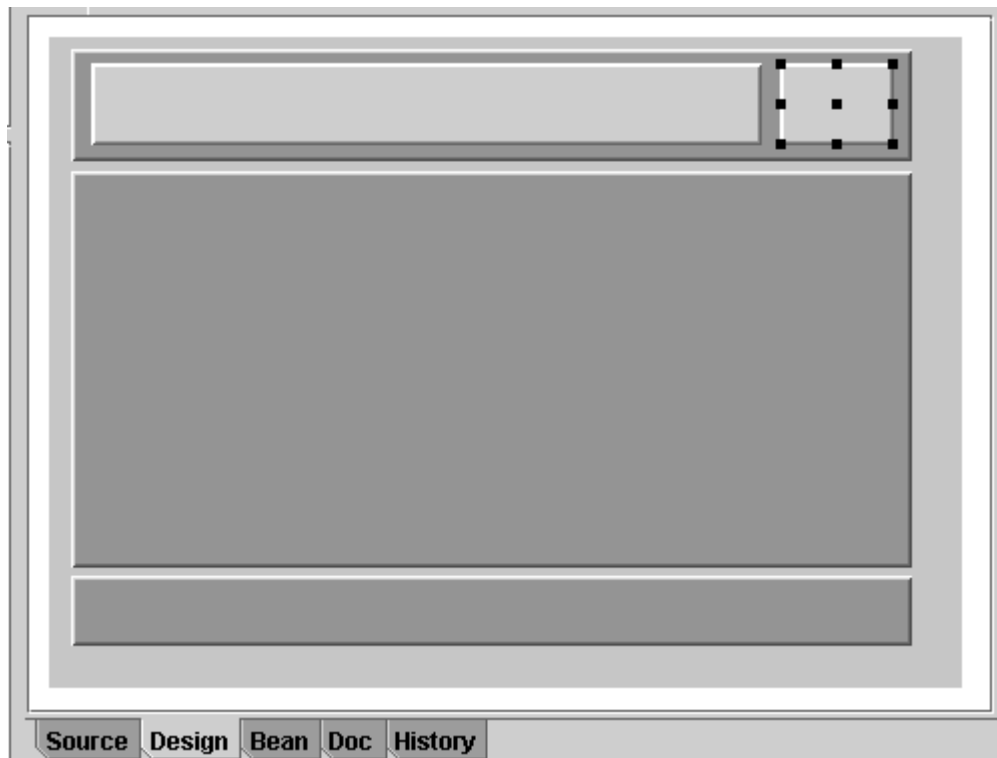
Contoh

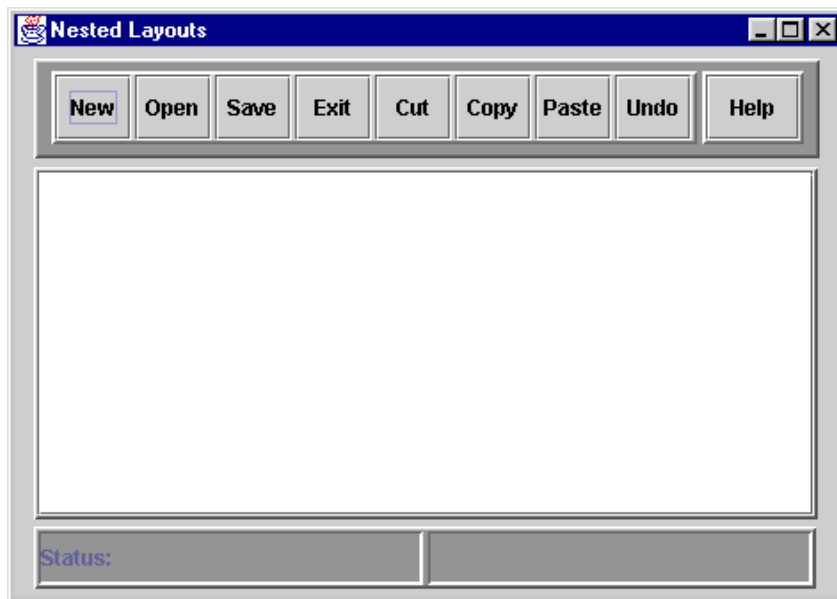
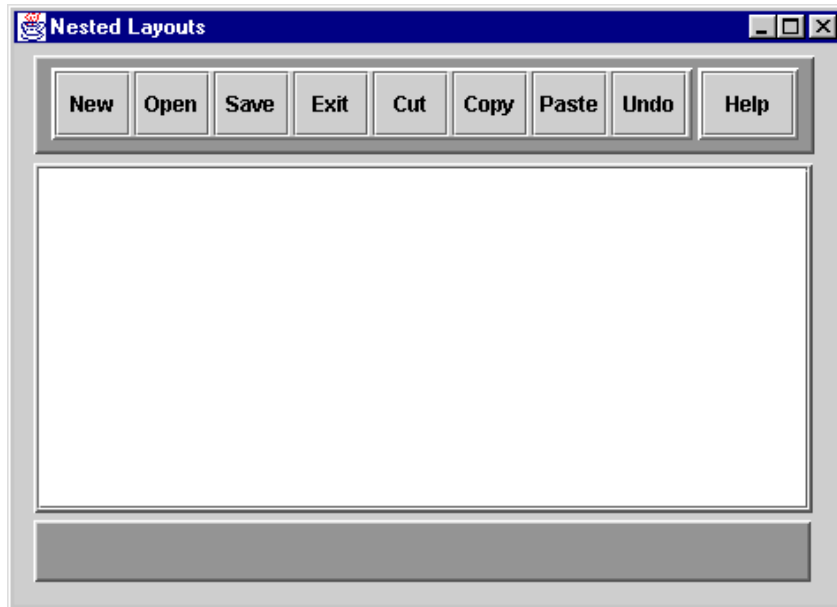


`AbsoluteLayout` sangat mudah untuk digunakan untuk membuat prototipe sebelum diubah ke dalam layout bersarang maupun `GridBagLayout`.

Contoh langkah-langkah pembuatan GUI

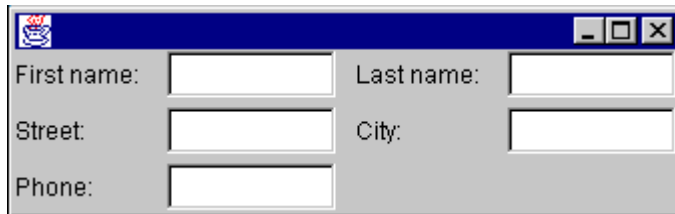






Tugas Kecil GUI

Kondisi awal

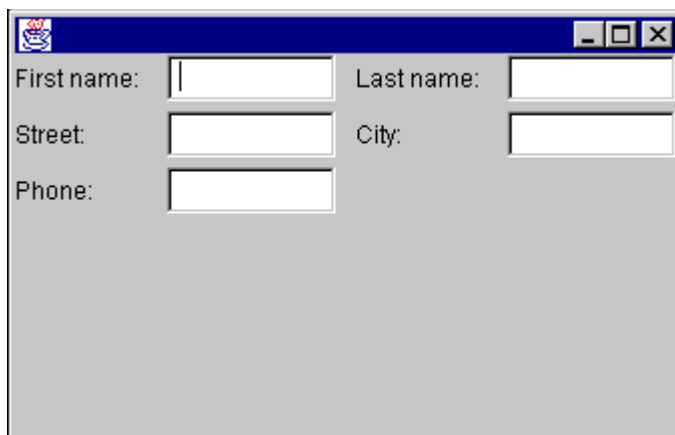


A screenshot of a GUI window with a blue title bar. The window contains a form with the following elements: 'First name:' followed by a text field, 'Last name:' followed by a text field, 'Street:' followed by a text field, 'City:' followed by a text field, and 'Phone:' followed by a text field. The labels and text fields are aligned horizontally.

Perhatikan bahwa label dan text field sejajar horisontal. Ketika GUI diperluas secara horisontal maka text field harus ikut memanjang tetapi label tidak boleh ikut memanjang.

Perhatikan juga apabila GUI diperluas vertikal

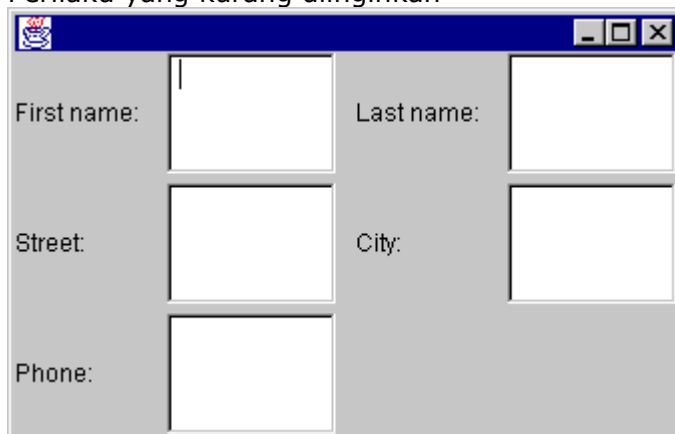
Perilaku yang diinginkan



A screenshot of a GUI window showing the form from the previous image. The window has been expanded horizontally. The text fields have become wider, but the labels ('First name:', 'Last name:', 'Street:', 'City:', 'Phone:') remain the same width and are not stretched.

Menjaga label dan text field pada tinggi yang diinginkan

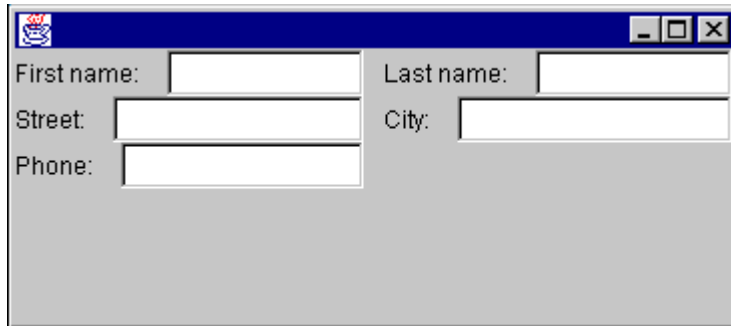
Perilaku yang kurang diinginkan



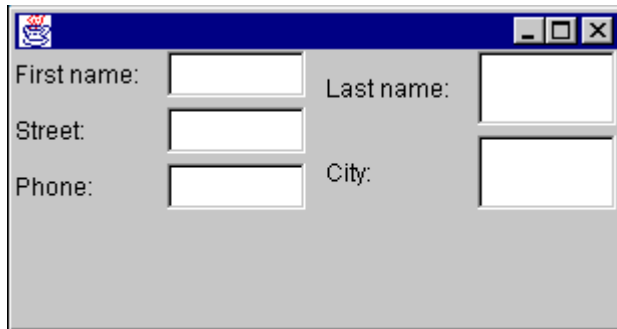
A screenshot of a GUI window showing the form from the previous image. The window has been expanded vertically. The text fields have become taller, and the labels ('First name:', 'Last name:', 'Street:', 'City:', 'Phone:') have also become taller, indicating that they are stretching along with the text fields.

label dan text field ikut meregang

Petunjuk:
kedua perilaku di bawah ini salah



A screenshot of a Java Swing window with a blue title bar and standard window controls. The window contains a form with five text input fields. The labels are arranged in two columns: 'First name:' and 'Last name:' on the top row, 'Street:' and 'City:' on the second row, and 'Phone:' on the third row. The 'Phone:' label is positioned to the left of its input field, while the other labels are to the left of their respective input fields.



A screenshot of a Java Swing window with a blue title bar and standard window controls. The window contains a form with five text input fields. The labels are arranged in two columns: 'First name:' and 'Last name:' on the top row, 'Street:' and 'City:' on the second row, and 'Phone:' on the third row. The 'Phone:' label is positioned to the left of its input field, while the other labels are to the left of their respective input fields.

HelloWorld Swing



```
package learn.swing;

import javax.swing.*;

public class HelloWorldSwing {

    public static void main(String[] args) {

        JFrame frame = new JFrame("HelloWorldSwing");

        final JLabel label = new JLabel("Hello World");

        frame.getContentPane().add(label);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.pack();

        frame.setVisible(true);

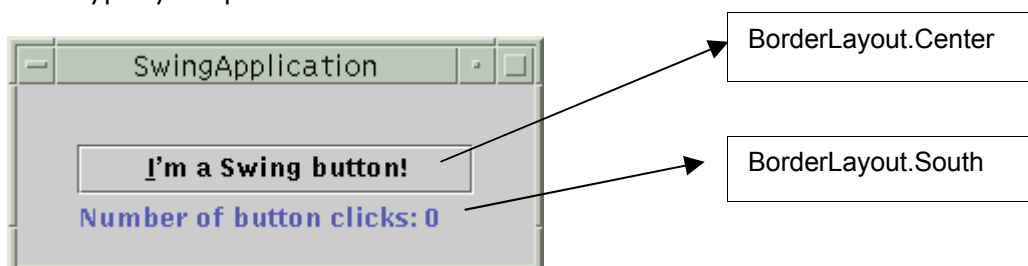
    }

}
```

MyFirstSwingApplication adalah program yang akan kita buat pertama kali dengan IDE NetBeans/Forte. Tujuannya untuk mengenal pola-pola program dengan GUI yang berbasis swing dengan event handling yang paling mudah. (Swing termasuk salah satu Java Foundation Class). Kerja dari program berikut adalah menampilkan banyaknya klik yang sudah dilakukan pada tombol.

Untuk membuat program yang baik kita harus banyak membaca kode yang baik pula oleh karena itu janganlah malas untuk membaca buku bagus.

Prototypenya seperti ini:

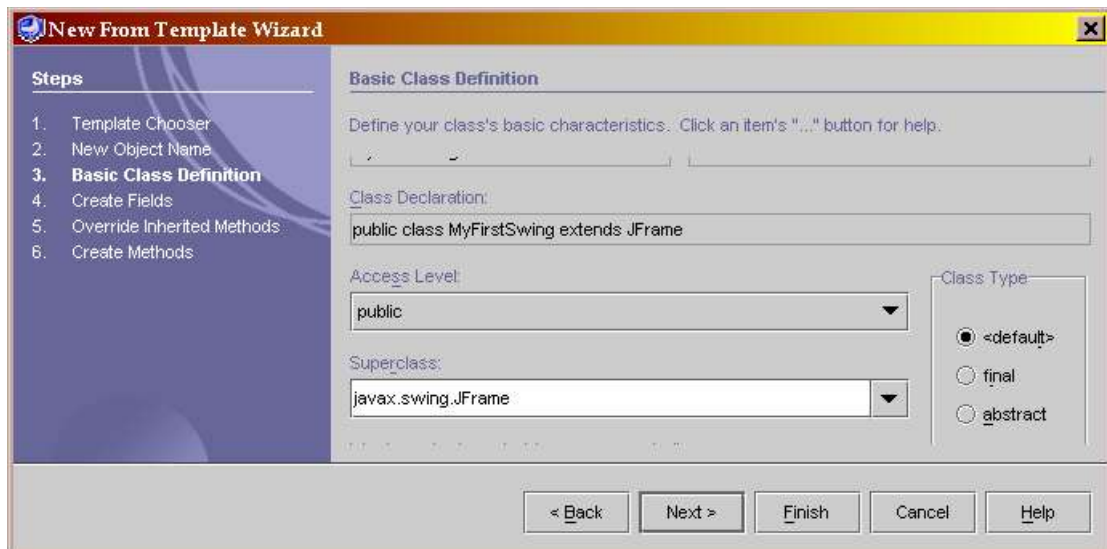


Kita akan mulai memanfaatkan kemampuan IDE yang telah kita pelajari.

Langkah-langkahnya adalah

Buka IDE dan buat objek baru dengan `New>SwingForms>Jframe`

Ketik `MyFirstSwing` pada field name klik Next lalu Finish



Lihat SourceEditor. Kode yang dibuat secara otomatis ini adalah kode yang paling sering muncul dalam inisialisasi GUI bahkan tanpa ada isi apapun

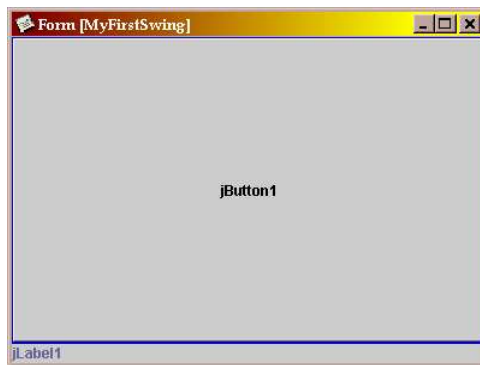
Perhatikan bahwa kita langsung dibawa ke workspace GUIEditing. Pada Component Inspector isilah atribut title dari `JFrame` dengan My first Swing Application, kemudian klik Test Form pada jendela utama



di layar akan tampak

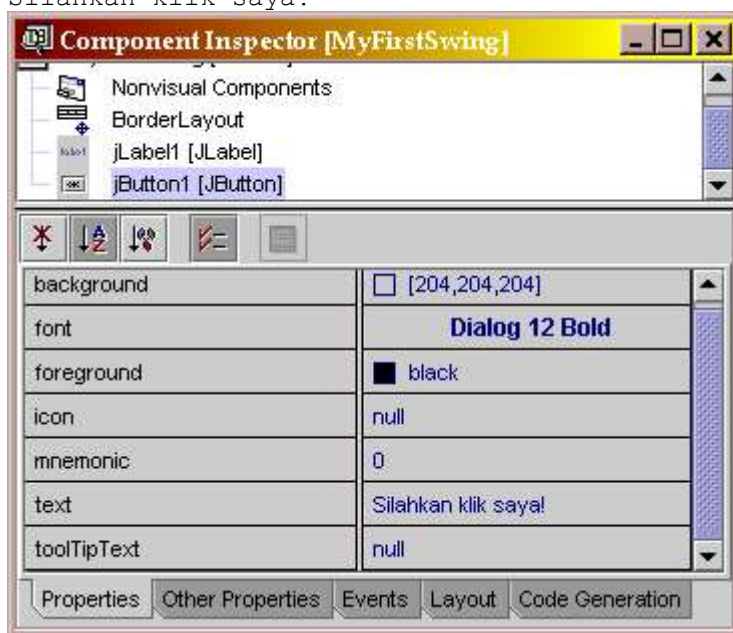


kemudian pada komponen palet klik `JLabel` dan tambahkan kedalam FromEditor di daerah sebelah batas bawah, kemudian klik `JButton` dan tambahkan ke tengah FormEditor



Periksa tab layout pada `jLabel1` (South) dan `jButton1` (Center), jika belum sesuaikanlah.

Pada component inspector ganti text pada tab Properties dari `JButton` menjadi
Silahkan klik saya!



Untuk `jLabel1` pada atribut text ketikkan Belum ada yang mengklik saya

Lihat kembali SourceEditor dan perhatikan perubahan-perubahan yang telah terjadi



apabila kita tarik perbesar perhatikan perubahannya



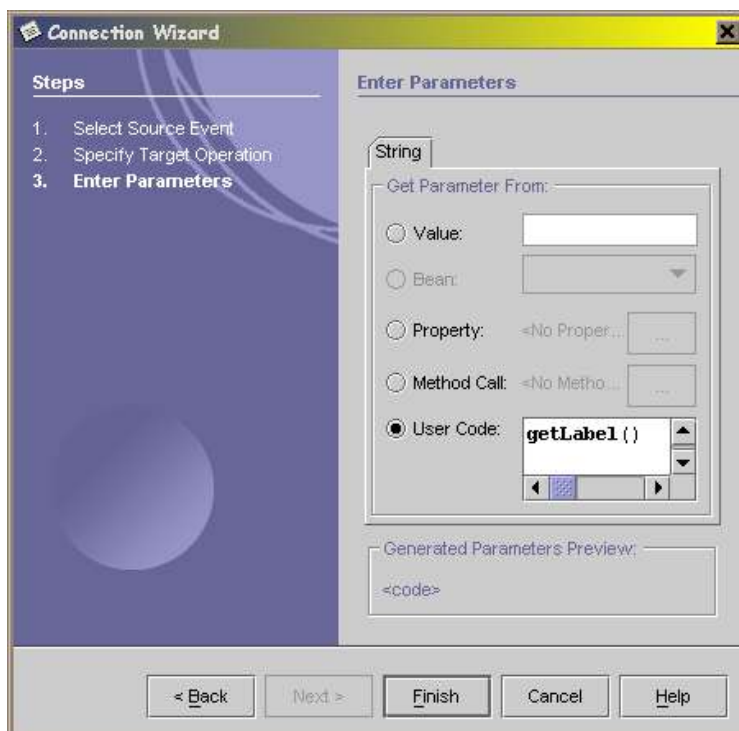
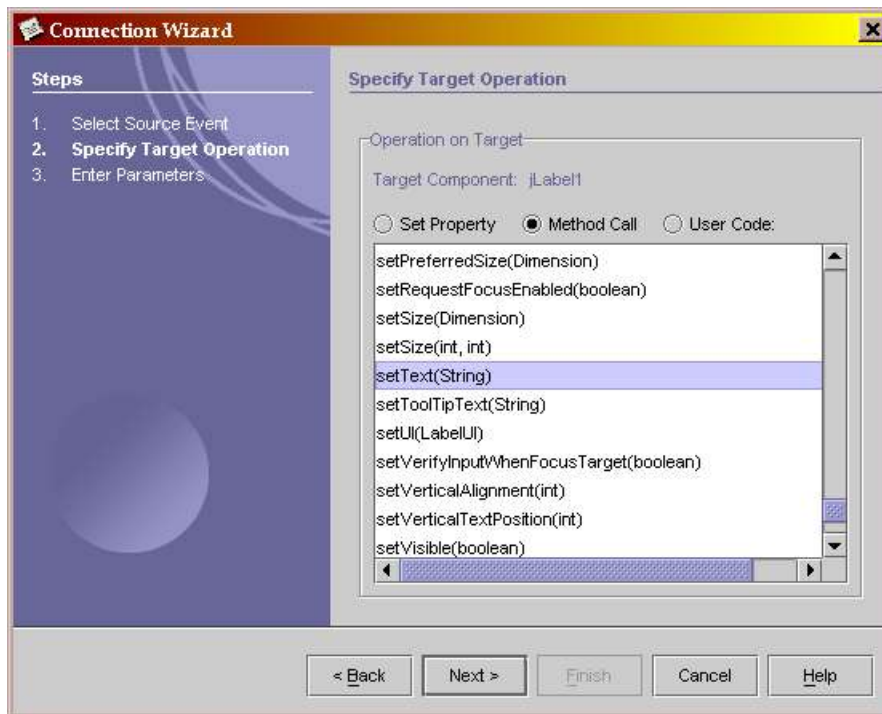
komponen `jButton1` memenuhi semua space yang tersedia sedangkan `jLabel1` hanya lebarnya saja yang mengikuti. Ini adalah sifat dari `BorderLayout` . Sekarang buatlah text pada `jLabel1` berada di tengah dengan mengganti atribut `horizontalAlignment` dari defaultnya `LEADING` menjadi `CENTER` .

Coba lagi Test Form

Sekarang kita selesaikan bagian eventhandling

Pada componen palet klik Connection mode terus..... klik `jButton1` dilanjutkan dengan `jLabel1` ini menunjukkan bahwa sumber dari event adalah `jButton1` dan yang menerimanya adalah `jLabel1` , kemudian tampilkan Connection Wizard. pilih event action performed dan anda bisa merubah nama methodnya jika anda mau, selanjutnya klik next



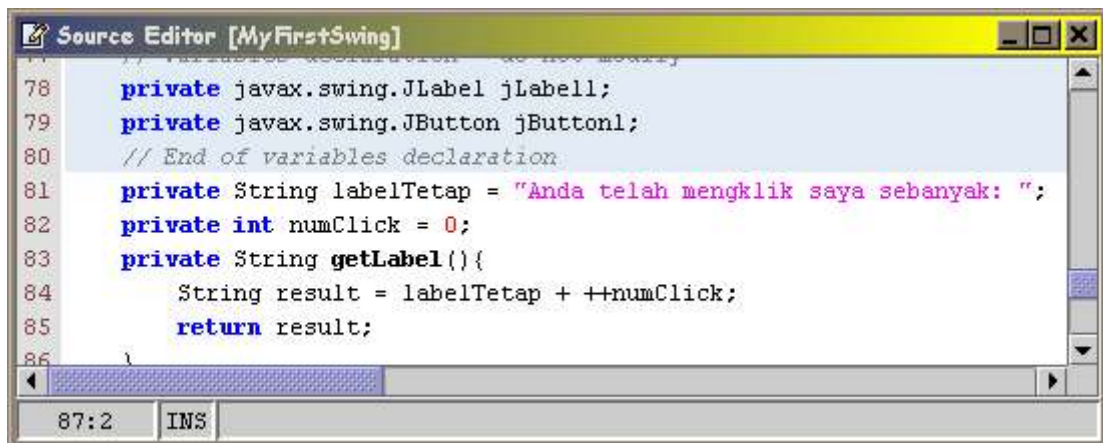


Selamat! anda telah berhasil menyelesaikan langkah event handling yang pertama, sekarang lihat lagi kode sumber anda.

Logika program ini sederhana saja, kita harus menyimpan jumlah klik dalam satu variabel dan setiap kali kita mengklik maka nilai variabelnya bertambah 1.

Kita juga harus membuat fungsi `getLabel()` yang harus mengembalikan objek bertipe `String`.

Saya akan melakukan penambahan kode di bagian bawah dari variabel deklarasi.



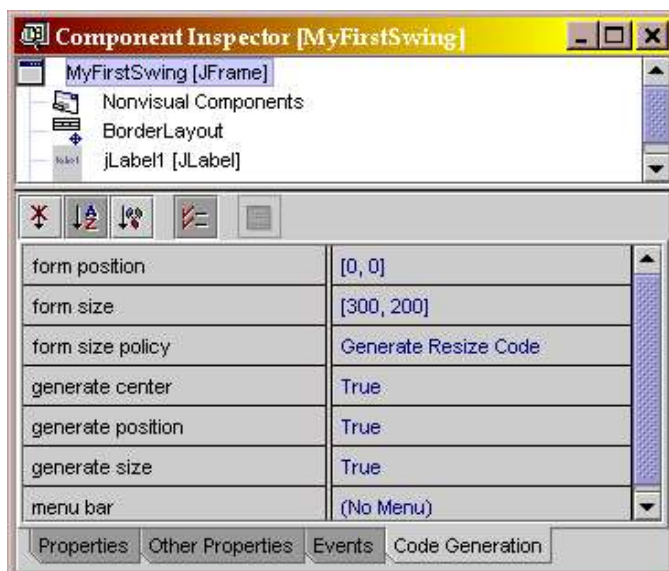
```
78 private javax.swing.JLabel jLabel1;  
79 private javax.swing.JButton jButton1;  
80 // End of variables declaration  
81 private String labelTetap = "Anda telah mengklik saya sebanyak: ";  
82 private int numClick = 0;  
83 private String getLabel(){  
84     String result = labelTetap + ++numClick;  
85     return result;  
86 }
```

Nah sekarang anda bisa mengkompilasinya dengan menekan F9 untuk menjalankannya tekan F6.

Lihat anda telah berhasil membuat program GUI yang baik untuk pemula perhatikan yang kode yang kita ketikkan betul-betul hanyalah tentang bagaimana program ini bekerja. Sederhana bukan?

Langkah selanjutnya adalah memperhalus tampilan GUI kita. Perhatikan bahwa pada saat kita mengeksekusi program ini, jendela program muncul di pojok kiri atas layar, hal ini disebabkan default posisinya adalah `[0,0]`. Ukuran dari jendela juga terlalu kecil untuk menampung komponen-komponen yang ada.

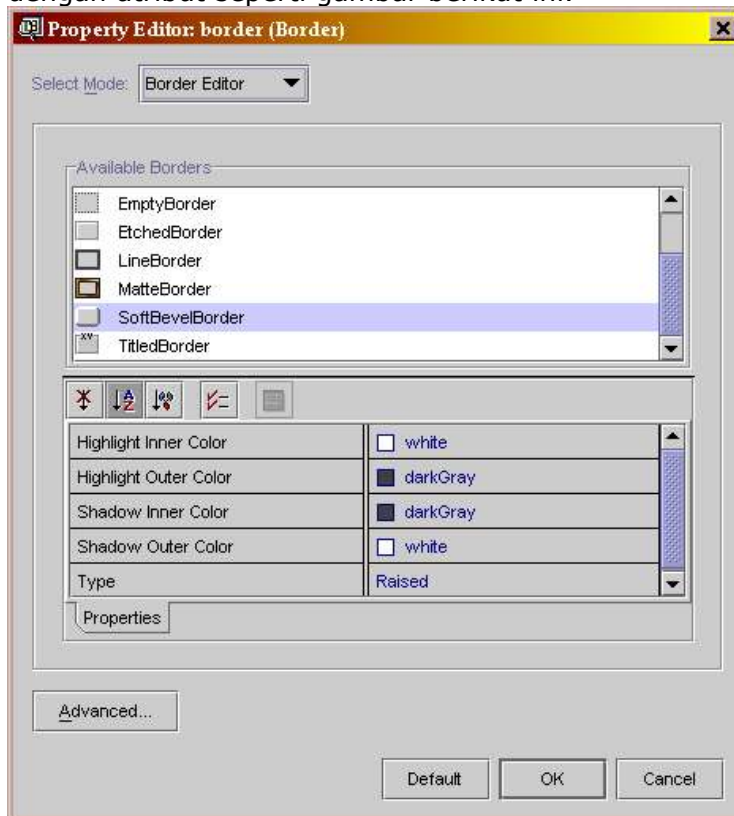
Pada komponen inspektor tab Code Generation ubah form size policynya menjadi Generate Resize Code



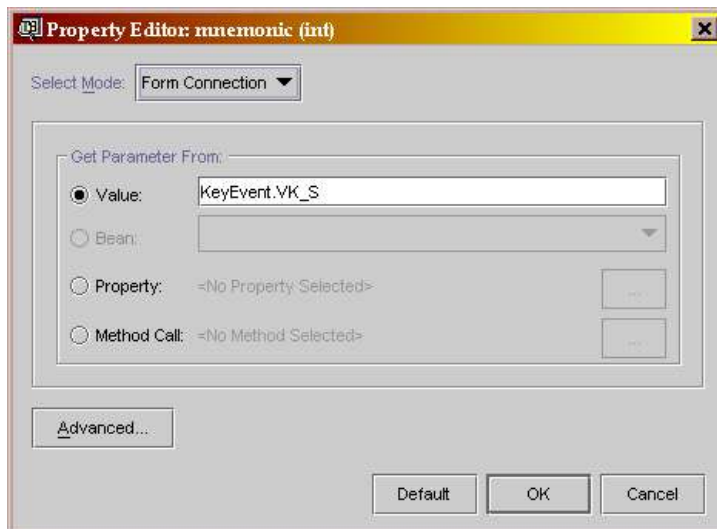
Kompilasi dan jalankan, akhirnya semua berjalan dengan lancar.

Selanjutnya kita akan memodifikasi `jButton1` dan `jLabel1` dengan memberinya border lalu menghilangkan garis biru yang mengelilingi text pada `jButton1`, selanjutnya kita juga memberikan tooltip pada `jButton1`

Pada `jLabel1`, ubah atribut border pada tab Other properties menjadi `BevelBorder` dan pilih type `Lowered`, sedangkan untuk `jButton` ubah menjadi `SoftBevelBorder` dengan atribut seperti gambar berikut ini.



Selanjutnya ubah tooltip pada tab properties menjadi `klik saja!` untuk menghilangkan garis biru disekitar text ubah atribut `focusPainted` menjadi `false`. Langkah terakhir adalah menambahkan mnemonic (keyboard shortcut) caranya dengan memasukkan nilai integer dari `KeyEvent`. Karena diinginkan apabila `Alt-S` ditekan sama dengan diklik maka kita masukkan nilai `KeyEvent.VK_S`. Jangan lupa untuk mengimpor `java.awt.event.KeyEvent` sesudah statement package



Hasil akhirnya seperti gambar di bawah ini. Perhatikan bahwa secara otomatis jvm akan menggarisbawahi dari huruf pertama mnemonic yang ditemukan.

(Dalam aplikasi nyata, biasanya JButton tidak akan mengikuti begitu saja perubahan ukuran dari kontainer yang ditempatinya)



Kode sumber akhir seperti ini:

```
/*
 * MyFirstSwing.java
 *
 * Created on March 5, 2002, 1:29 PM
 */

package learn.swing;

import java.awt.event.KeyEvent;

/**
 *
 */
```

```
* @author BUFFON
*/

public class MyFirstSwing extends javax.swing.JFrame {

    /** Creates new form MyFirstSwing */
    public MyFirstSwing() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();

        setTitle("My first Swing Application ");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        jLabel1.setText("Belum ada yang mengklik saya");
        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel1.setBorder(new javax.swing.border.BevelBorder(
            javax.swing.border.BevelBorder.LOWERED));

        getContentPane().add(jLabel1, java.awt.BorderLayout.SOUTH);

        jButton1.setToolTipText("klik saja!");
        jButton1.setMnemonic(KeyEvent.VK_S);
        jButton1.setText("Silahkan klik saya!");

        jButton1.setBorder(new javax.swing.border.SoftBevelBorder(
            javax.swing.border.BevelBorder.RAISED, java.awt.Color.darkGray, java.awt.Color.white,
            java.awt.Color.white, java.awt.Color.darkGray));

        jButton1.setFocusPainted(false);
    }
}
```

```
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1, java.awt.BorderLayout.CENTER);

        pack();

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().
        getScreenSize();
        setSize(new java.awt.Dimension(300, 200));
        setLocation((screenSize.width-300)/2, (screenSize.height-200)/2);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        jLabel1.setText(getLabel());
    }

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new MyFirstSwing().show();
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JButton jButton1;
    // End of variables declaration
    private String labelTetap = "Anda telah mengklik saya sebanyak: ";
```

```
private int numClick = 0;
private String getLabel(){
    String result = labelTetap + ++numClick;
    return result;
}
}
```


Bab XI.Observer dan Custom Event

A.Observer

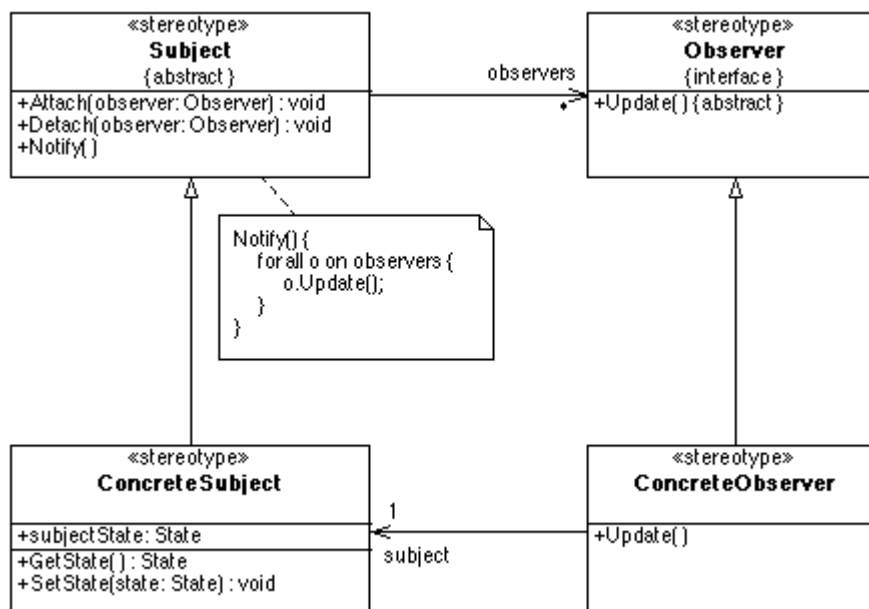
Kita sudah mempelajari bahwa objek-objek bekerja sama dengan cara saling berkomunikasi satu sama lainnya. Salah satu cara yang kita pelajari adalah dengan memanggil method pada objek yang ingin kita pakai. Cara ini tergolong dalam komunikasi synchronous, atau request-response model, atau command mode.

Selain itu ada juga yang disebut dengan asynchronous notification, seperti yang telah kita gunakan pada pelajaran tentang Swing, yaitu pada event handling. Dengan cara ini kita dapat mengimplementasikan hubungan satu-ke-banyak (*one-to-many*) antar objek sehingga perubahan pada satu objek bisa diketahui dan ditanggapi oleh yang lain tanpa peningkatan coupling yang tidak perlu.

Konsep ini juga dipakai untuk menghindari ketergantungan dua arah antar layer. Contoh layer presentasi/GUI mengetahui tentang layer yang ada di bawahnya yaitu layer domain model /business logic, sehingga ia bisa meminta informasi langsung dari sana. Tetapi apabila terjadi perubahan pada domain model, bagaimana cara memberitahu ke layer yang di atasnya, karena ia seharusnya tidak mengetahui tentang layer yang berada di atasnya. Di sinilah Observer berperan sebagai solusi.

Bagi yang ingin mengetahui lebih lanjut tentang Observer silahkan merujuk ke Design Pattern [GoF] atau buku-buku yang serupa.

Di bawah ini adalah class diagram dari Observer pattern. Perhatikan penggunaan interface yang menjadi point of coupling.



Dalam bahasa Java, Observer pattern diimplementasikan dalam class `java.util.Observable` dan interface `java.util.Observer`.

Contoh:

Class `IntegerDataBag` menjadi subject yang akan diamati, diturunkan dari `Observer` dan bertugas menyimpan kumpulan dari objek-objek `Integer`. Pengamatnya ada dua buah yaitu `IntegerAdder` yang bertugas untuk menjumlahkan semua bilangan yang ada pada `IntegerDataBag` dan `IntegerPrinter` yang bertanggungjawab untuk mencetak semua angka yang ada dalam `IntegerDataBag`.

Program clientnya adalah seperti ini:

```
package learn.design.observer;

public class Client {
    public static void main( String [] args ) {
        Integer i1 = new Integer( 1 );
        Integer i2 = new Integer( 2 );
        Integer i3 = new Integer( 3 );
        Integer i4 = new Integer( 4 );
        Integer i5 = new Integer( 5 );
        Integer i6 = new Integer( 6 );
        Integer i7 = new Integer( 7 );
        Integer i8 = new Integer( 8 );
        Integer i9 = new Integer( 9 );

        IntegerDataBag bag = new IntegerDataBag();
        bag.add( i1 );
        bag.add( i2 );
        bag.add( i3 );
        bag.add( i4 );
        bag.add( i5 );
        bag.add( i6 );
        bag.add( i7 );
        bag.add( i8 );

        IntegerAdder adder = new IntegerAdder( bag );
        IntegerPrinter printer = new IntegerPrinter( bag );

        // adder and printer add themselves to the bag

        System.out.println( "About to add another integer to the bag:" );
        bag.add( i9 );
        System.out.println("");
        System.out.println("About to remove an integer from the bag:");
        bag.remove( 0 );
    }
}
```

Apabila kita jalankan program yang sudah lengkap, hasilnya akan seperti ini:

```
About to add another integer to the bag:
The contents of the IntegerDataBag have changed.
The new contents of the IntegerDataBag contains:
1
2
3
4
5
6
7
8
9
The contents of the IntegerDataBag have changed.
```

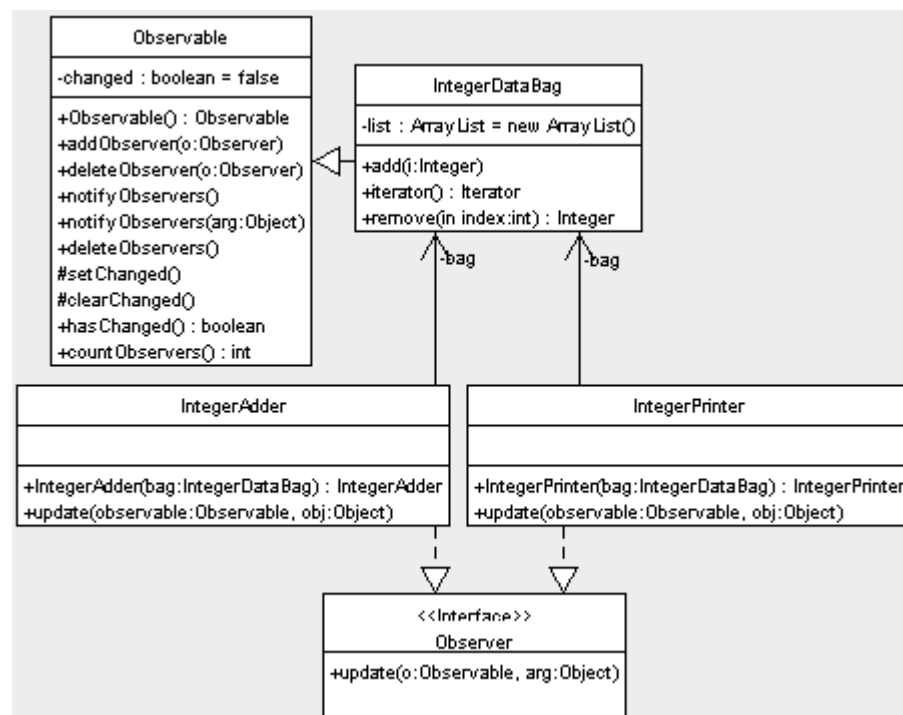
The new sum of the integers is: 45

About to remove an integer from the bag:
 The contents of the IntegerDataBag have changed.
 The new contents of the IntegerDataBag contains:

- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

The contents of the IntegerDataBag have changed.
 The new sum of the integers is: 44

Class diagramnya seperti di bawah ini. Perhatikan bahwa IntegerDataBag tidak mengetahui apa-apa tentang siapa saja yang mengamatinya, dan apabila terjadi perubahan pada dirinya, pihak-pihak yang berkepentingan langsung diberi tahu dan menanggapi sesuai dengan tanggung jawabnya masing-masing.



Program lengkapnya
 IntegerDataBag.java

```

package learn.design.observer;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Observable;

public class IntegerDataBag extends Observable {
    private ArrayList list = new ArrayList();

    public void add( Integer i ) {
        list.add( i );
        setChanged();
        notifyObservers();
    }
}
    
```

```
        clearChanged();
    }
    public Iterator iterator() {
        return list.iterator();
    }
    public Integer remove( int index ) {
        if( index < list.size() ) {
            Integer i = (Integer) list.remove( index );
            setChanged();
            notifyObservers();
            clearChanged();
            return i;
        }
        return null;
    }
}
```

IntegerPrinter.java

```
package learn.design.observer;
import java.util.Iterator;
import java.util.Observer;

public class IntegerPrinter implements Observer {
    private IntegerDataBag bag;

    public IntegerPrinter( IntegerDataBag bag ) {
        this.bag = bag;
        bag.addObserver( this );
    }
    public void update(java.util.Observable observable, java.lang.Object obj) {
        if( observable == bag ) {
            System.out.println( "The contents of the IntegerDataBag have changed." );
            System.out.println( "The new contents of the IntegerDataBag contains:" );
            Iterator i = bag.iterator();
            while( i.hasNext() ) {
                System.out.println( i.next() );
            }
        }
    }
}
```

IntegerAdder.java

```
package learn.design.observer;
import java.util.Iterator;
import java.util.Observer;

public class IntegerAdder implements Observer {
    private IntegerDataBag bag;

    public IntegerAdder( IntegerDataBag bag ) {
        this.bag = bag;
        bag.addObserver( this );
    }
    public void update(java.util.Observable observable, java.lang.Object obj) {
        if( observable == bag ) {
            System.out.println("The contents of the IntegerDataBag have changed." );
            int counter = 0;
            Iterator i = bag.iterator();
            while( i.hasNext() ) {
                Integer integer = ( Integer ) i.next();
                counter+=integer.intValue();
            }
            System.out.println( "The new sum of the integers is: " + counter );
        }
    }
}
```

Contoh dalam Swing yang menunjukkan pemisahan layer presentasi dengan layer domain model. Program ini menampilkan menghitung panjang dari suatu interval dan mengkalkulasi ulang setiap kita merubah isi field-fieldnya.

Interval.java adalah program yang berisi domain model dan berdiri sendiri.

```

package learn.swing;
public class Interval extends java.util.Observable {
    private int start = 0;
    private int end = 0;
    private int length = 0;
    /** Creates new Interval */
    public Interval() {
    }
    int getStart(){
        return start;
    }
    void setStart(int newStart){
        this.start = newStart;
        setChanged();
        notifyObservers();
    }
    int getEnd(){
        return end;
    }
    void setEnd(int newEnd){
        this.end = newEnd;
        setChanged();
        notifyObservers();
    }
    int getLength(){
        return length;
    }
    void setLength(int newLength){
        this.length = newLength;
        setChanged();
        notifyObservers();
    }
    public void calculateLength(){
        int newLength = end - start;
        setLength(newLength);
    }
    public void calculateEnd(){
        int newEnd = start + length;
        setEnd(newEnd);
    }
}

```

IntervalFrame.java dibuat dengan Forte sehingga kita tinggal menuliskan event handlingnya dan update() methodnya saja



```

package learn.swing;
import java.util.*;
public class IntervalFrame extends javax.swing.JFrame implements java.util.Observer {
    public IntervalFrame() {
        initComponents();
        subject = new Interval();
        subject.addObserver(this);
        update(subject, null);
    }
    private void initComponents() {
        jPanel3 = new javax.swing.JPanel();
        jPanel4 = new javax.swing.JPanel();
        jPanel1 = new javax.swing.JPanel();
        startField = new javax.swing.JTextField();
        endField = new javax.swing.JTextField();
        lengthField = new javax.swing.JTextField();
        jPanel2 = new javax.swing.JPanel();
        startLabel = new javax.swing.JLabel();
        endLabel = new javax.swing.JLabel();
        lengthLabel = new javax.swing.JLabel();
    }
}

```

```
setTitle("IntervalWindow");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

jPanel3.setLayout(new java.awt.BorderLayout());

jPanel4.setLayout(new java.awt.BorderLayout(5, 2));

jPanel1.setLayout(new java.awt.GridLayout(3, 0, 0, 10));

startField.addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(java.awt.event.FocusEvent evt) {
        startFieldFocusLost(evt);
    }
});

jPanel1.add(startField);

endField.addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(java.awt.event.FocusEvent evt) {
        endFieldFocusLost(evt);
    }
});

jPanel1.add(endField);

lengthField.addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(java.awt.event.FocusEvent evt) {
        lengthFieldFocusLost(evt);
    }
});

jPanel1.add(lengthField);

jPanel4.add(jPanel1, java.awt.BorderLayout.CENTER);

jPanel2.setLayout(new java.awt.GridLayout(3, 0, 0, 10));

startLabel.setText("Start");
jPanel2.add(startLabel);

endLabel.setText("End");
jPanel2.add(endLabel);

lengthLabel.setText("length");
jPanel2.add(lengthLabel);

jPanel4.add(jPanel2, java.awt.BorderLayout.WEST);

jPanel3.add(jPanel4, java.awt.BorderLayout.CENTER);

getContentPane().add(jPanel3, java.awt.BorderLayout.NORTH);

pack();
}

private void lengthFieldFocusLost(java.awt.event.FocusEvent evt) {
    // Add your handling code here:
    try {
        int newLength = Integer.parseInt(lengthField.getText());
        subject.setLength(newLength);
        subject.calculateEnd();
    } catch (NumberFormatException e) {
        throw new RuntimeException("Unexpected Number Format Error");
    }
}

private void endFieldFocusLost(java.awt.event.FocusEvent evt) {
    // Add your handling code here:
    try {
        int newEnd = Integer.parseInt(endField.getText());
        subject.setEnd(newEnd);
        subject.calculateLength();
    } catch (NumberFormatException e) {
```

```

        throw new RuntimeException("Unexpected Number Format Error");
    }
}

private void startFieldFocusLost(java.awt.event.FocusEvent evt) {
    // Add your handling code here:
    try {
        int newStart = Integer.parseInt(startField.getText());
        subject.setStart(newStart);
        subject.calculateLength();
    } catch (NumberFormatException e) {
        throw new RuntimeException("Unexpected Number Format Error");
    }
}

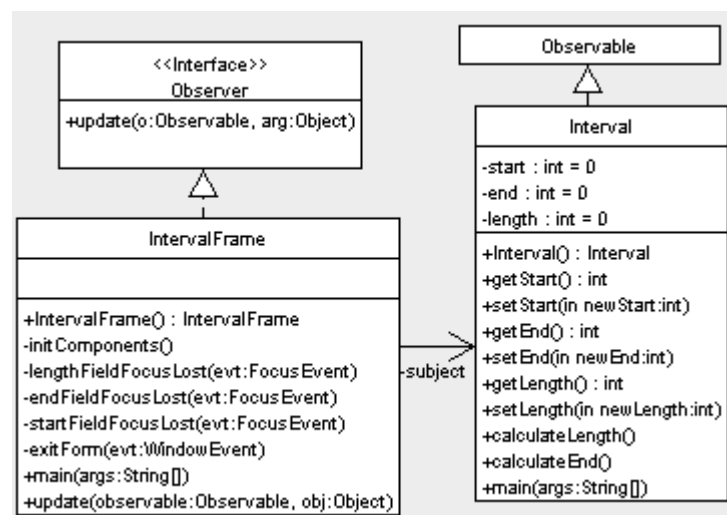
/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit(0);
}

public static void main(String args[]) {
    new IntervalFrame().show();
}

public void update(java.util.Observable observable, java.lang.Object obj) {
    startField.setText(Integer.toString(subject.getStart()));
    endField.setText(Integer.toString(subject.getEnd()));
    lengthField.setText(Integer.toString(subject.getLength()));
}

// Variables declaration - do not modify
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel1;
private javax.swing.JTextField startField;
private javax.swing.JTextField endField;
private javax.swing.JTextField lengthField;
private javax.swing.JPanel jPanel2;
private javax.swing.JLabel startLabel;
private javax.swing.JLabel endLabel;
private javax.swing.JLabel lengthLabel;
// End of variables declaration
private Interval subject;
}

```



B.Custom Event

Versi yang lain dari observer adalah custom event, dimana Observer digantikan dengan EventListener dan object yang dilewatkan ke dalam Observer bukan lagi Observable tetapi adalah EventObject

```
package learn.design.customevent;

public class ADayInTheLife {
    public static void main( String [] args ) {
        MrHappyObject happy = new MrHappyObject();
        MoodListener sky = new Sky();
        MoodListener birds = new FlockOfBirds();
        happy.addMoodListener( sky );
        happy.addMoodListener( birds );

        System.out.println( "Let's pinch MrHappyObject and find out what happens:" );
        happy.receivePinch();

        System.out.println("");
        System.out.println( "Let's hug MrHappyObject and find out what happens:" );
        happy.receiveHug();

        System.out.println("");
        System.out.println( "Now, let's make MrHappyObject angry and find out what happens:" );
        System.out.println("");
        System.out.println("one pinch:");
        happy.receivePinch();
        System.out.println("");
        System.out.println("second pinch, look out:");
        happy.receivePinch();
    }
}
```

Keluarannya adalah sebagai berikut:

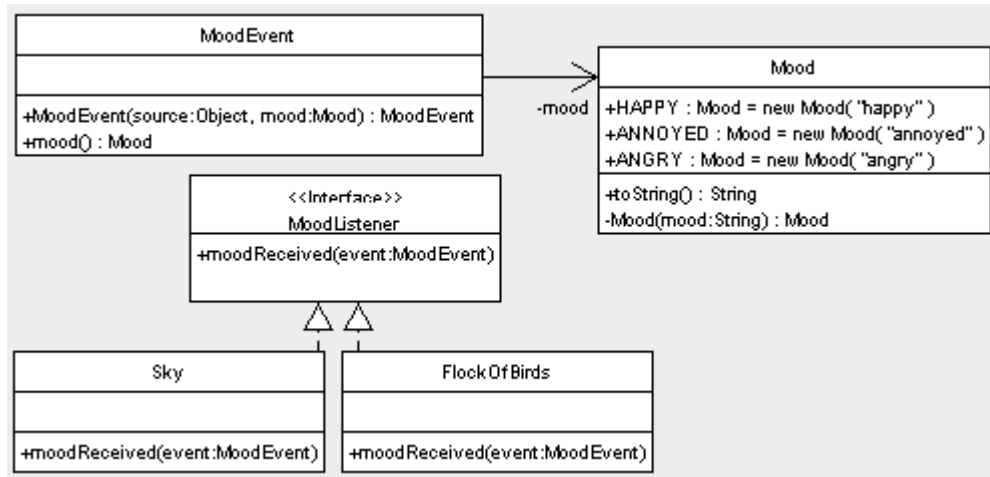
```
Let's pinch MrHappyObject and find out what happens:
Cloudy Skies!
Birds are silent!
```

```
Let's hug MrHappyObject and find out what happens:
Sun is shining!
Birds are singing!
```

```
Now, let's make MrHappyObject angry and find out what happens:
```

```
one pinch:
Cloudy Skies!
Birds are silent!
```

```
second pinch, look out:
Lightning rains from the heavens!
Birds are flying away!
```

Kode lengkapnya ada di bawah ini:

```

package learn.design.customevent;
import java.util.EventObject;

public class MoodEvent extends EventObject {
    private Mood mood;

    public MoodEvent( Object source, Mood mood ) {
        super( source );
        this.mood = mood;
    }

    public Mood mood() {
        return mood;
    }
}

package learn.design.customevent;
public class Mood {
    public static final Mood HAPPY = new Mood( "happy" );
    public static final Mood ANNOYED = new Mood( "annoyed" );
    public static final Mood ANGRY = new Mood( "angry" );
    private String mood;

    public String toString() {
        return mood;
    }
    private Mood( String mood ) {
        this.mood = mood;
    }
}

package learn.design.customevent;
import java.util.EventListener;
public interface MoodListener extends EventListener {
    public void moodReceived( MoodEvent event );
}

package learn.design.customevent;
public class FlockOfBirds implements MoodListener {

    public void moodReceived(MoodEvent event) {
        if( event.mood() == Mood.HAPPY ) {
            System.out.println( "Birds are singing!" );
        }
        else if( event.mood() == Mood.ANNOYED ) {
            System.out.println( "Birds are silent!" );
        }
        else {
            System.out.println( "Birds are flying away!" );
        }
    }
}

```

```

}

package learn.design.customevent;
public class Sky implements MoodListener {

    public void moodReceived(MoodEvent event) {
        if( event.mood() == Mood.HAPPY ) {
            System.out.println( "Sun is shining!" );
        }
        else if( event.mood() == Mood.ANNOYED ) {
            System.out.println( "Cloudy Skies!" );
        }
        else {
            System.out.println( "Lightning rains from the heavens!" );
        }
    }
}

package learn.design.customevent;
public class Sky implements MoodListener {

    public void moodReceived(MoodEvent event) {
        if( event.mood() == Mood.HAPPY ) {
            System.out.println( "Sun is shining!" );
        }
        else if( event.mood() == Mood.ANNOYED ) {
            System.out.println( "Cloudy Skies!" );
        }
        else {
            System.out.println( "Lightning rains from the heavens!" );
        }
    }
}

```



```

package learn.design.customevent;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MrHappyObject {
    private Mood mood = Mood.HAPPY;
    private List listeners = new ArrayList();

    public synchronized void receivePinch() {
        if( mood == Mood.HAPPY ) {
            mood = Mood.ANNOYED;
            fireMoodEvent();
        } else {
            mood = Mood.ANGRY;
            fireMoodEvent();
        }
    }

    public synchronized void receiveHug() {
        if( mood == Mood.ANGRY ) {
            mood = Mood.ANNOYED;
            fireMoodEvent();
        } else {
            mood = Mood.HAPPY;
            fireMoodEvent();
        }
    }

    public synchronized void addMoodListener( MoodListener l ) {
        listeners.add( l );
    }

    public synchronized void removeMoodListener( MoodListener l ) {
        listeners.remove( l );
    }
}

```

```
private synchronized void fireMoodEvent() {
    MoodEvent moodEvent = new MoodEvent( this, mood );
    Iterator it = listeners.iterator();
    while( it.hasNext() ) {
        ( (MoodListener) it.next() ).moodReceived( moodEvent );
    }
}
```

Bab XII.Ikhtisar aturan-aturan JavaBean

Sebuah JavaBean mempunyai karakteristik sebagai berikut:

- Ia adalah class public
- Ia mempunyai konstruktor publik tanpa parameter (tapi dia juga boleh mempunyai konstruktor bentuk lain)
- Ia mengimplementasikan interface Serializable (artinya dapat dibuat persisten, sehingga statenya dapat disimpan
- Ia mempunyai **properti-properti** dengan method "**getter**(aksesor)" dan "**setter** (mutator)" dengan penamaan yang mengikuti konvensi nama JavaBeans
- Ia memiliki event-event yang mengikuti **model event Java yang baku** dengan method **pendaftaran** yang penamaannya mengikuti konvensi nama JavaBeans
- Ia boleh mempunyai method-method lain yang tidak mengikuti konvensi nama JavaBeans, tapi method-method ini tidak akan diperlihatkan oleh IDE

Catatan:

Seorang **pemakai** dari komponen-komponen JavaBean dapat membangun sebuah **aplikasi** yang terdiri atas beberapa JavaBeans, dengan cara meng**konfigurasinya** dan **menghubungkan** sekian JavaBean bersama-sama pada saat **perencanaan**, dengan menggunakan **builder dalam IDE** (Integrated Development Environment), seperti Forte4Java, Netbeans dan JBuilder.

Builder menganalisa dan memperlihatkan **properti-properti** dan **event-event** dari JavaBean pada **saat perancangan melalui introspection**. Ia dapat melakukannya hanya jika ia mengikuti konvensi nama diterapkan pada properti dan event. (Dalam kelas ini tidak akan dibahas sampai beanInfo dan customizer).

1.Konvensi nama untuk atribut

Property name: <propertyName>
e.g. temperature

Property type: <Type>
e.g. double

Getter method: public <Type> get<PropertyName>() { //... }
e.g. double getTemperature() { // ... }

Boolean Property Getter Method: public boolean is<PropertyName>() { //... }
e.g. public boolean isFull() { // ... }

Setter method: public void set<PropertyName>(<Type> p) { // ... }
e.g. void setTemperature(double temp) { // ... }

B.Event-event dalam JavaBeans

Satu set Event terdiri atas:

- Sebuah Event Listener Interface:
interface ini merupakan ekstensi dari interface `java.util.EventListener`, dan mendefinisikan satu atau lebih method yang harus diimplementasikan oleh kelas yang ingin menerima event ini.
- Sebuah Event Object
Sebuah **event object** dilewatkan dari event source ke listener. Ia memperluas `java.util.EventObject` atau salah satu subclassnya.
- Method untuk Event Registration

Mereka adalah method **add-** atau **remove-listener**, yang diimplementasikan oleh komponen yang menjadi sumber event. Nama dari method-method ini harus memenuhi konvensi nama JavaBean.

1.Predefined Event Sets

Package `java.awt.event` menyediakan beberapa predefined event sets, misalnya, `KeyEvent` event set terdiri atas:

1. Interface `KeyListener`, method-method `keyPressed()`, `KeyReleased()`, dan `keyTyped()`
2. Class event object `KeyEvent`.
3. Komponen yang membangkitkan key event harus mendefinisikan method registrasi `addKeyListener(KeyListener kl)` dan `removeKeyListener(KeyListener kl)`.

Langkah-langkah untuk mengimplementasikan event-event multi-cast dalam sebuah JavaBean

- Pilih sebuah **nama** `<eventName>`, untuk event set, contoh: `Timer`
- Definisikan sebuah class **event object**, `<eventName>Event`, yang diturunkan dari `java.util.EventObject`. contoh: `TimerEvent`.
- Definisikan field-field yang dibutuhkan untuk menyimpan informasi keadaan (**state**) untuk event tersebut.
- Buat **constructor** yang mengambil sedikitnya satu argumen yang bertipe **object**, lalu memanggil parent constructor dan kemudian menginisialisasi field-field yang lain.

Contoh:

```
public class TimerEvent extends EventObject {
    private int tickCount;
    public TimerEvent(Object source) {
        super(source);
        tickCount = 0;
    }
    public int getTickCount() {
        return tickCount;
    }
    // ...
}
```

- Definisikan interface **event listener**, `<eventName>Listener`, contoh, `TimerListener` memperluas interface `java.util.EventListener`. Definisikan satu atau lebih operasi di dalam interface ini. Ke dalam operasi-operasi ini harus dilewatkan `<eventName>Event` pada argumennya, bersama dengan argumen lain sesuai dengan hasil rancangan. Operasi-operasi ini akan diimplementasikan oleh method-method event handling.

Biasanya, class-class event listening ini merupakan *local anonymous inner class*.

Dalam contoh kita, interface `TimerListener` dapat berupa seperti ini:

```
public interface TimerListener extends EventListener {
    public void timerTick(TimerEvent event);
    public void timerStart(TimerEvent event);
    public void timerStop(TimerEvent event);
}
```

- Implementasikan method **event registration** pada komponen sumber event.

method registrasi add-listener Multicast:

```
public void add<eventName >Listener (<eventName>Listener aListener);
```

contoh: `public synchronized void addTimerListener(TimerListener aListener);`

keyword `synchronized`, digunakan untuk melindungi dari masalah *race* dan *deadlock* problems pada sebuah lingkungan yang bersifat multithread.

method registrasi remove-listener:

`public void remove<eventName >Listener (<eventName>Listener aListener);`

contoh: `public synchronized void removeTimerListener(TimerListener aListener);`

keyword `synchronized`, digunakan untuk melindungi dari masalah *race* dan *deadlock* problems pada sebuah lingkungan yang bersifat multithread.

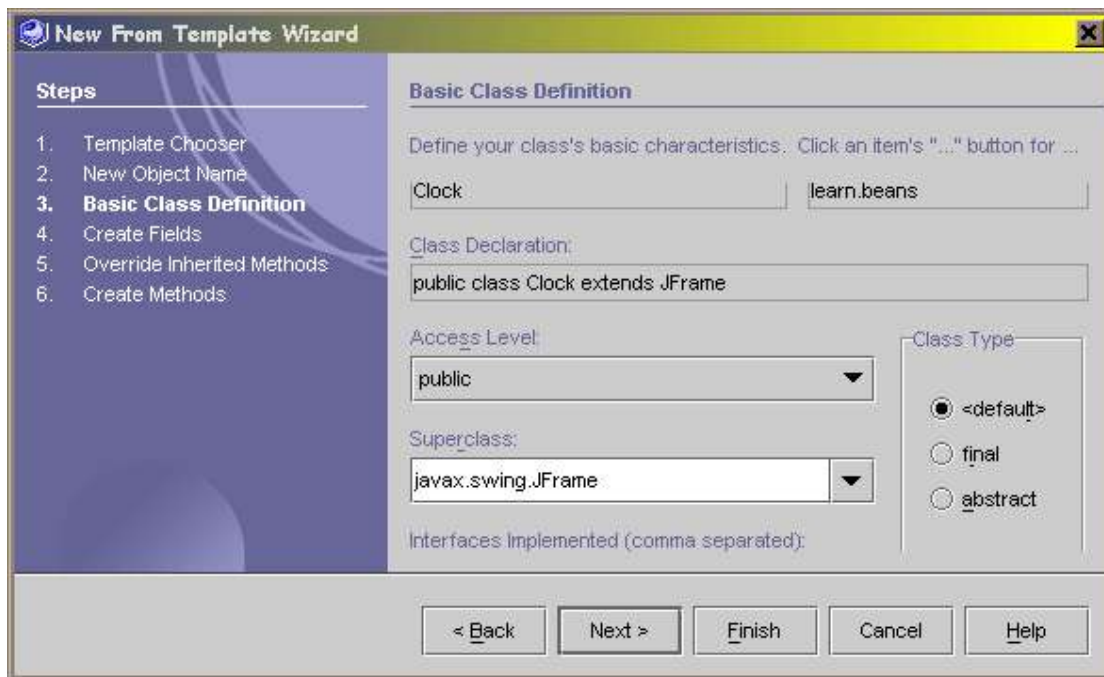
B.Mengirimkan event kepada listeners

Class yang membangkitkan event bertanggung jawab untuk menentukan kapan sebuah even sebenarnya terjadi. Ketika sebuah event tertentu terjadi, class yang menimbulkan event tersebut harus:

1. Ciptakan sebuah **event object**, dengan melewati referensi dirinya sendiri **"this"** ke dalam constructor dari event object tersebut. Tetapkan keadaan dari event object tersebut jika diperlukan.
2. Berkeliling ke seluruh objek yang telah mendaftarkan diri mereka untuk tertarik terhadap satu event tertentu, dan panggil satu atau lebih operasi yang telah didefinisikan pada interface event listening. Ia akan melewati event object yang telah dibuat sebagai argumen dari operasi-operasi ini. Misalnya, komponen `Timer`, yang memancarkan `TimerEvent` mungkin akan terlihat seperti ini:

```
public class Timer {
    private Vector listenerList = new Vector();
    public synchronized void addTimerListener(TimerListener tl) {
        // add tl to listenerList
    }
    public synchronized void removeTimerListener (TimerListener tl){
        // remove tl from listenerList
    }
    public processTimerTick() {
        TimerEvent ev = new TimerEvent(this);
        // set state of event object, ev, if needed
        For(int i = 0; i < listenerList.size(); ++i) {
            ((TimerListener)listenerList.elementAt(i)).timerTick(ev);
        }
    }
    public void run() {
        //
        // at each tick invoke this.processTimerTick()
        //
    }
}
```

Selanjutnya kita akan belajar bagaimana menggunakan JavaBeans Timer yang sudah ada dan terpasang pada Forte/Netbeans



1.Awal :

Klik kanan pada sebuah directory dalam jendela Explorer dan klik "New Packages". Sebuah dialog terbuka, tuliskan nama packagenya beans. kemudian klik kanan lagi pada package beans dan pilih New>SwingForms>JFrame. Ketikkan Clock pada field Object Name dan klik Next>Finish

Anda lihat bahwa pada baris status dari jendela utama terbaca "Opening Form : Clock". Workspacenya pindah ke GUIEditing, dan beberapa jendela terbuka - Source Editor, FormEditor dan Component Inspector. Perhatikan juga pada kode sumber ada bagian-bagian yang diberi warna. Ini dibuat oleh FormEditor dan tidak boleh dimodifikasi. Di dalam Component Inspector tampak komponen-komponen yang berada pada form dan propertinya. Pada awalnya tidak ada komponen yang berada di sana kecuali default layout (BorderLayout) dan simpel dari komponen Non-Visible

2.Menambah komponen :

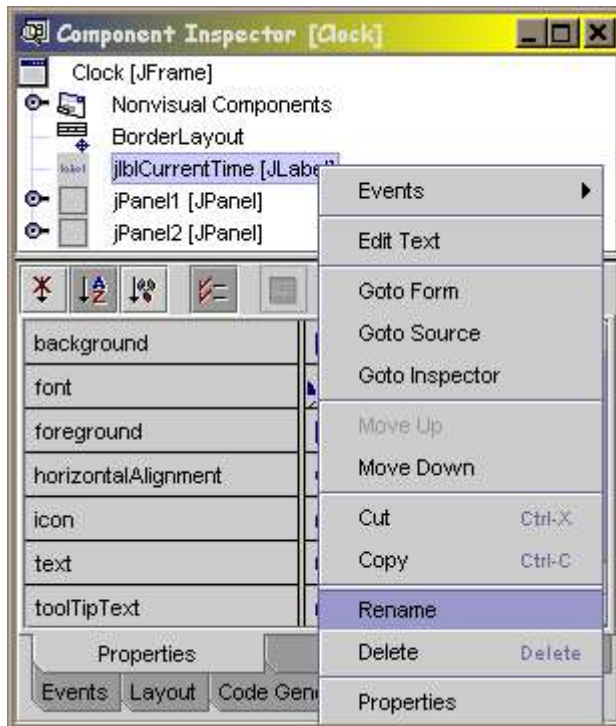
Kita akan menggunakan JLabel untuk menampilkan angka jam. Pindah ke tab Swing dari Component Palette, pilih JLabel dengan mengklik pada iconnya. Icon menjadi aktif.

Letakkan ia pada tengah panel dalam FormEditor dengan sekali klik. Anda akan melihat baris-baris baru muncul di Editor, dan sebuah komponen baru di dalam Component

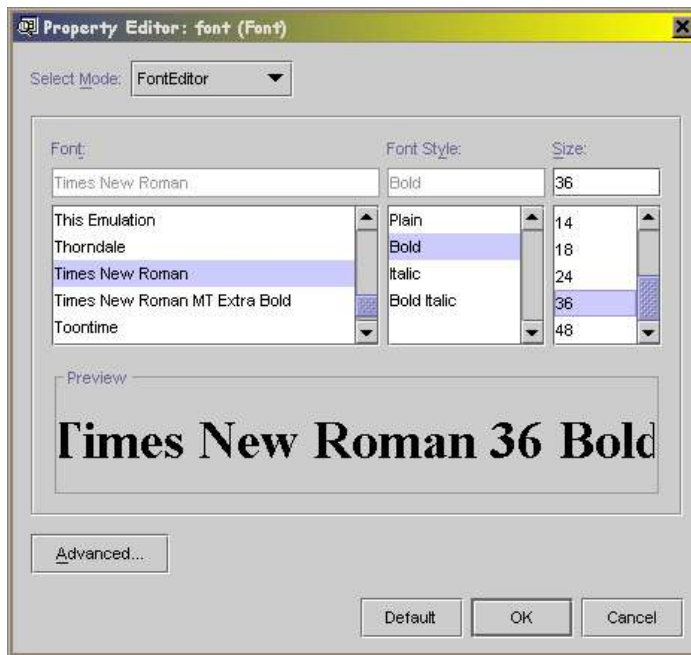
Inspector. Perhatikan bahwa komponen terpilih ditandai dengan garis biru di sekeliling komponen tersebut. Component Inspector juga menandai komponen tersebut.

3. Memodifikasi atribut dari komponen:

- Sekarang kita akan memodifikasi atribut dari JLabel. Pastikan JLabel pada Component Inspector telah dipilih lalu klik kanan.



- Kotak Dialog Renama terbuka. Hapus jLabel1 dan ketikkan jlblCurrentTime di sini lalu pilih Ok.
- Kemudian, pindah ke tab Properties dan temukan atribut text. Klik pada nilai atribut (defaultnya sekarang jLabel1) dan ketikkan teks yang akan muncul di Label: 00:00:00 lalu tekan Enter. Perhatikan perubahan pada FormEditor dan SourceEditor.
- JLabel adalah tampilan utama dari jam, jadi kita ubah saja font defaultnya. Klik atribut Font pada Component Inspector, dan pilih tombol "..." yang muncul. Kotak dialog Property Editor akan terbuka



- Ubah setting font menjadi Times Roman, Bold, 36 pt. FontEditor ikut berubah kemudian klik Ok untuk menutup dialog.
- Terakhir, untuk membuat teks berada di tengah-tengah jam. Ubah atribut `horizontalAlignment` dari nilai defaultnya (LEFT) menjadi CENTER.

4. Menambah kode untuk fungsionalitas :

- Pertama-tama, kita harus mengimpor class-class yang kita perlukan ke dalam kode. Pindah ke bagian atas Source Editor dan ketikkan di bawah "package beans;".

```
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Calendar;
import java.text.SimpleDateFormat;
```

Kita juga akan memasukkan `JOptionPane` untuk pesan kesalahan:

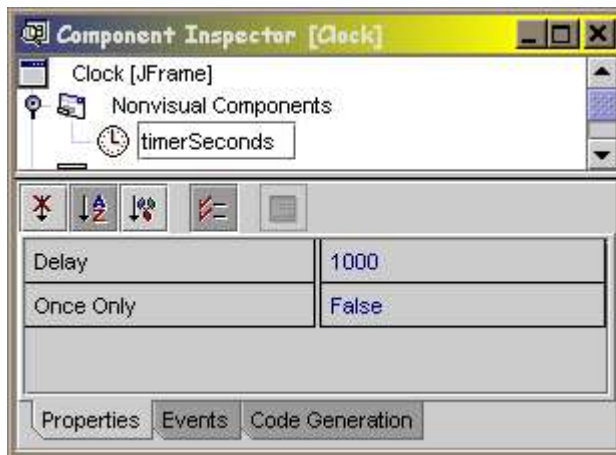
```
import javax.swing.JOptionPane;
```

- Tambahkan baris berikut di bawah blok "Variables Declaration". Kita sudah pernah membahas tentang class-class ini pada pelajaran-pelajaran sebelumnya (pengenalan terhadap Java), jadi seharusnya anda sudah mengerti apa yang sebenarnya terjadi bukan?

```
private GregorianCalendar gCal = new GregorianCalendar();
private String timeFormat = "hh:mm:ss";
private SimpleDateFormat formatter = new SimpleDateFormat(timeFormat);
```

5. Menambahkan TimerBean, dan menseset sebuah event handler :

- Pada tab Beans dari Component Palette, pilih `TimerBean`. Letakkan ia di mana saja pada Form Editor. `TimerBean` adalah komponen non-visual, anda tidak akan melihat perubahan apa pun pada Form Editor. `TimerBean` akan terlihat di dalam Component Inspector, di bawah simpul NonVisual Components.
- Pilih `TimerBean` pada Component Inspector, dan ubah nama variabelnya menjadi `timerSeconds`.



- Pindah ke dalam Events panel dari TimerBean. Apabila anda telah mengklik ganda Timer pada Componen Inspector, maka event onTime sudah diset menjadi timerSecondsOnTime. Dan anda akan melihat SourceEditor menunjukkan method yang baru dibuatnya. Jika anda melihat di sebelah atas kode maka akan terlihat kode listener yang memanggil event handler ini. Seandainya belum diset maka ubahlah menjadi timerSecondsOnTime



- Sekarang kita akan menambahkan isi dari method event handler ini. Tambahkan di bawah "// Add your handling code here" baris :

```

gCal.add(Calendar.SECOND,1);
String timeTxt = formatter.format(gCal.getTime());
if (lblCurrentTime != null)
    lblCurrentTime.setText(timeTxt);

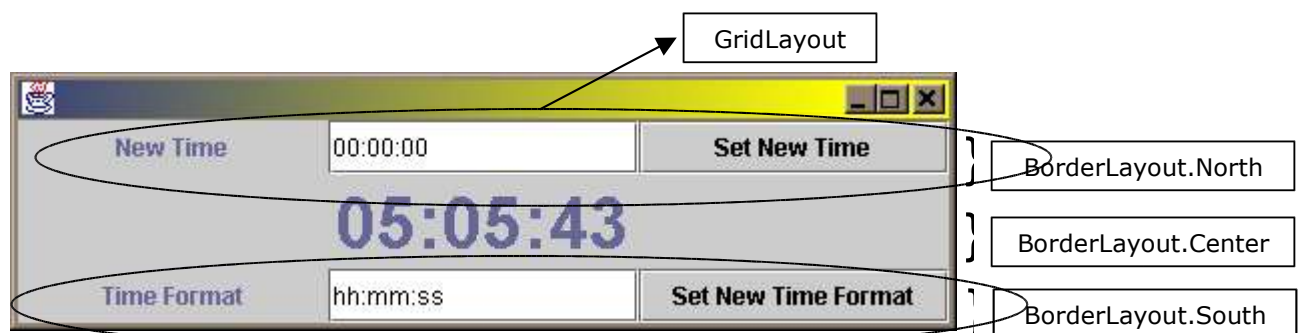
```

- Jam yang kita buat sekarang telah selesai. Jalankan programnya dengan mengklik icon Execute icon dari jendela utama. IDE akan menyimpan file sumber, mengkompilasi dan menjalankannya.
- Apabila tidak ada kesalahan, kompilasi akan sukses. IDE akan pindah ke workspace Running dan jendela Jam akan terbuka. Perhatikan Execution View, Clock ditampilkan sebagai proses yang sedang berjalan process.

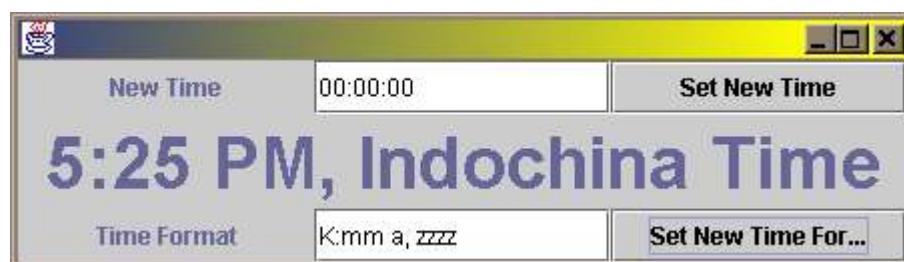
That's it, folk!

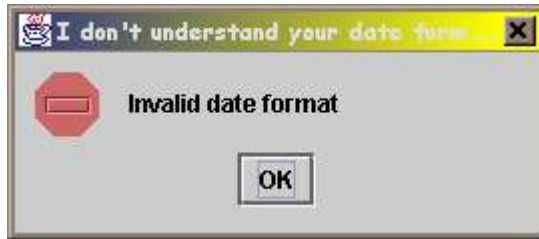


C. Tambahkan modifikasi (untuk JavaBeans sudah selesai)



Selanjutnya kita akan coba untuk memodifikasi jam yang telah kita buat. Pertama kita harus bisa menset jam tersebut, kedua kita ingin agar bisa mengubah-ubah tampilannya dengan mudah. Ketiga, apabila kita memberi masukan yang salah maka program kita harus menampilkan pesan kesalahan. Diilustrasikan seperti gambar di bawah ini

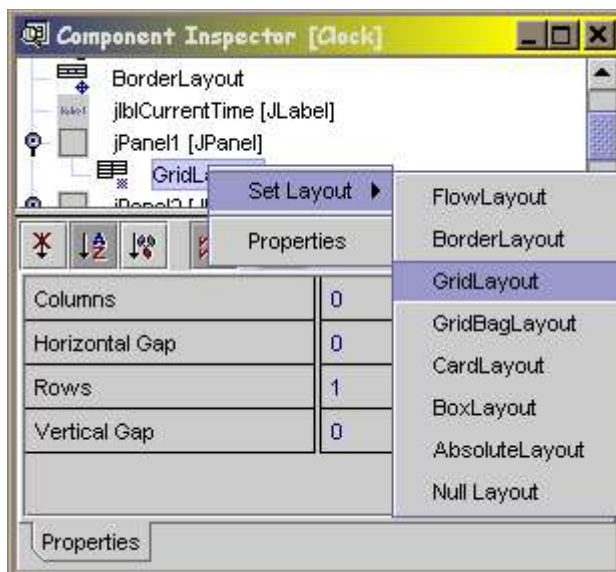




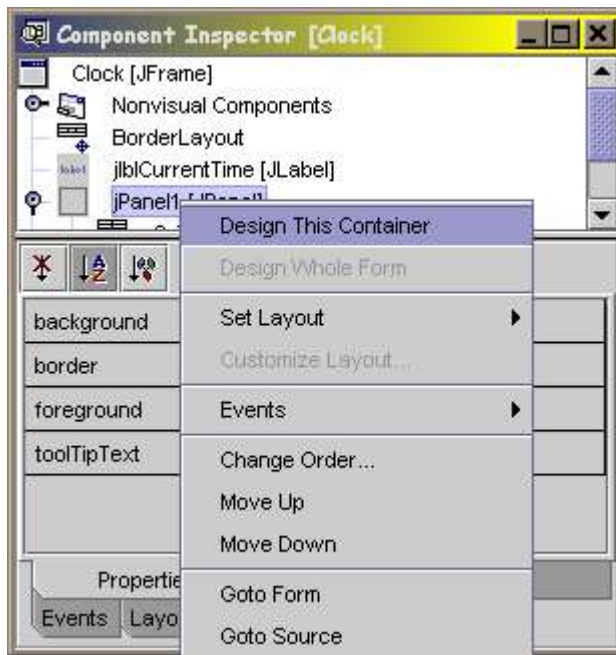
- Pertama kita harus menambahkan panel-panel kontainer di sebelah atas dan bawah jam. Pilih JPanel pada Component Palette dan letakkan di sebelah atas dari Form. Kemudian pilih lagi JPanel dan letakkan di sebelah bawah dari Form.



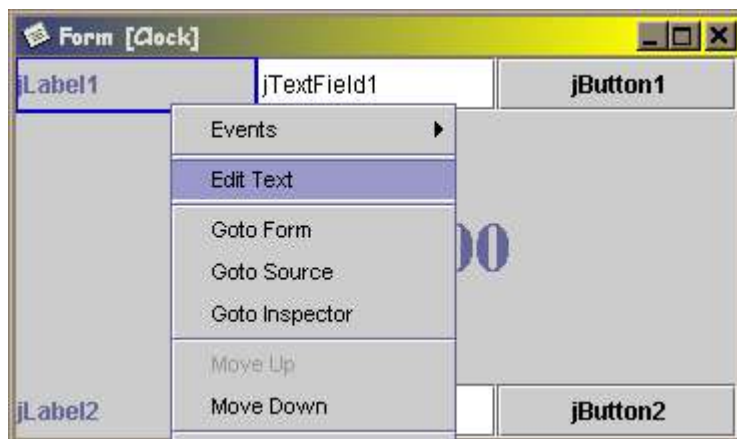
- Perhatikan perubahan pada Component Inspector. kemudian ubah layout kedua panel menjadi GridLayout



- Pindahkan perancangan Form dari JFrame ke jPanel1 dengan klik kanan dan memilih Design This Container. Kemudian tambahkan berturut-turut JLabel, JTextField dan JButton ke dalam Form. Lakukan hal yang sama untuk jPanel2.



- Selanjutnya ubah teks masing-masing komponen sesuai dengan ilustrasi gambar di atas -- misal jLabel1 menjadi New Time. Untuk JLabel set horizontal alignmentnya menjadi CENTER, sedangkan JButton itu memang nilai defaultnya.



- Sekarang coba jalankan programnya dan ubah waktu serta formatnya. Jika tidak terjadi apa-apa, itu memang karena kita belum memasang event handlernya ☺.

1. Menambah event handler.

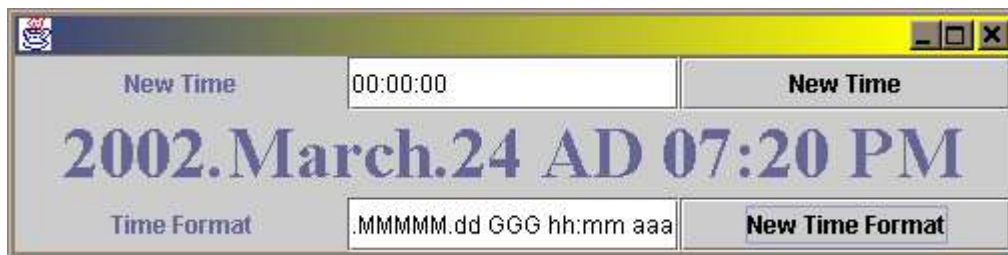
- Ganti kedua nama variabel JButton menjadi jbtnNewTime dan jbtnNewTimeFormat supaya mudah untuk dibedakan. Begitu juga dengan kedua JTextField menjadi jtfNewTime dan jtfNewTimeFormat
- Klik ganda kedua buah JButton tersebut pada Component Inspector. Kode untuk penanganan aksi secara otomatis dibuat pada Source Editor. Tambahkan kode seperti di bawah ini

```
private void jbtnNewTimeFormatActionPerformed(java.awt.event.ActionEvent evt) {  
    // Add your handling code here:  
    String timeFormat = jtfNewTimeFormat.getText();  
}
```

```
        formatter = new SimpleDateFormat(timeFormat);
    }

    private void jbtnNewTimeActionPerformed(java.awt.event.ActionEvent evt) {
        // Add your handling code here:
        try {
            String timeStr = jtfNewTime.getText();
            gCal.setTime(formatter.parse(timeStr));
        } catch (java.text.ParseException e) {
            JOptionPane.showMessageDialog(this, "Invalid date format", "I don't
understand your date format.", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

- Perhatikan betapa mudahnya menampilkan pesan kesalahan dengan menggunakan block try-catch dan JOptionPane
- Selesai sudah pekerjaan kita, jangan lupa untuk membaca dokumentasi API dari SimpleDateFormat supaya tidak mendapat `IllegalArgumentException` pada saat memasukkan format waktu yang baru.



Tugas kecil ☺:

Jika anda menggunakan jdk1.4.0 silahkan mencoba `SpringLayout` sehingga jendela otomatis mengembang atau mengempis mengikuti perubahan format waktu. Jika anda masih menggunakan jdk1.3.1 tugasnya adalah melokalisasi `DateFormat` untuk default locale menjadi format Indonesia misal: jam 12 malam hari Kamis bulan Februari tahun 2002 dengan menggunakan `DateFormatSymbols`. Petunjuk: lihat contohnya pada `IndonesianFormat`, kalau perlu tambahkan saja method staticnya di sana.

Bab XIII.Evolusi kode

Pada bagian ini kita akan melihat bagaimana suatu masalah diselesaikan dengan cara yang berbeda tergantung dari pemahaman dan pengalaman pemrogram terhadap pola pikir berbasis objek.

Contoh yang diambil adalah contoh trivial tentang pembayaran gaji yang bergantung kepada kepada jenis pekerjaan. Ini tentu saja sangat disederhanakan dan tidak ada di dunia nyata :). Outputnya adalah sama untuk setiap versi, yaitu:

```
This month we pay 10 to Adhia
This month we pay 9 to Vina
This month we pay 8 to Fauzan
```

Pemrogram yang baik tentu saja selalu berusaha membuat programnya sesederhana mungkin dalam menyelesaikan suatu masalah. Tetapi kesederhanaan ini adalah bukan kebetulan dan baru ia peroleh setelah mengetahui beberapa alternatif yang dapat dilakukan serta mengetahui konsekuensi dari alternatif-alternatif tersebut. Ia juga tahu kapan saat yang tepat untuk berhenti melakukan analisis atau perancangan dan mulai mengkodekan. Artinya ia tidak terjebak dalam analysis-paralysis maupun over-design. Beberapa pakar bahkan menganjurkan "Just In Time design". Konsep ini akan lebih mudah untuk dipahami apabila kita telah membaca buku Refactoring[Fowler] dan Design Pattern[GoF].

Solusi pertama pemrogram yang berorientasi objek mungkin seperti di bawah ini dan apabila ia tahu bahwa logika programnya (aturan penggajian) tidak akan berubah atau bertambah rumit dari yang ada sekarang ini maka iapun merasa cukup. Bagi yang belum berfikir secara objek ini kemungkinan besar ini adalah solusi pertama sekaligus terakhir karena ia tidak memiliki alternatif yang lebih sederhana lagi.

Employee
-type : int +ENGINEER : int = 0 +SALESMAN : int = 1 +MANAGER : int = 2 +monthlySalary : int = 8 +comission : int = 1 +bonus : int = 2
+Employee(in type:int, name:String) : Employee +payAmount() : int +toString() : String

```
package learn.refactoring.payment1;
import java.util.*;
class Employee {
    private int type;
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2 ;
    private String name;
    int monthlySalary = 8;
    int comission = 1;
    int bonus = 2;
    Employee(int type, String name){
        this.type = type;
        this.name = name;
    }
    int payAmount() {
        switch(type){
            case ENGINEER:
                return monthlySalary;
            case SALESMAN:
                return monthlySalary + comission;
        }
    }
}
```

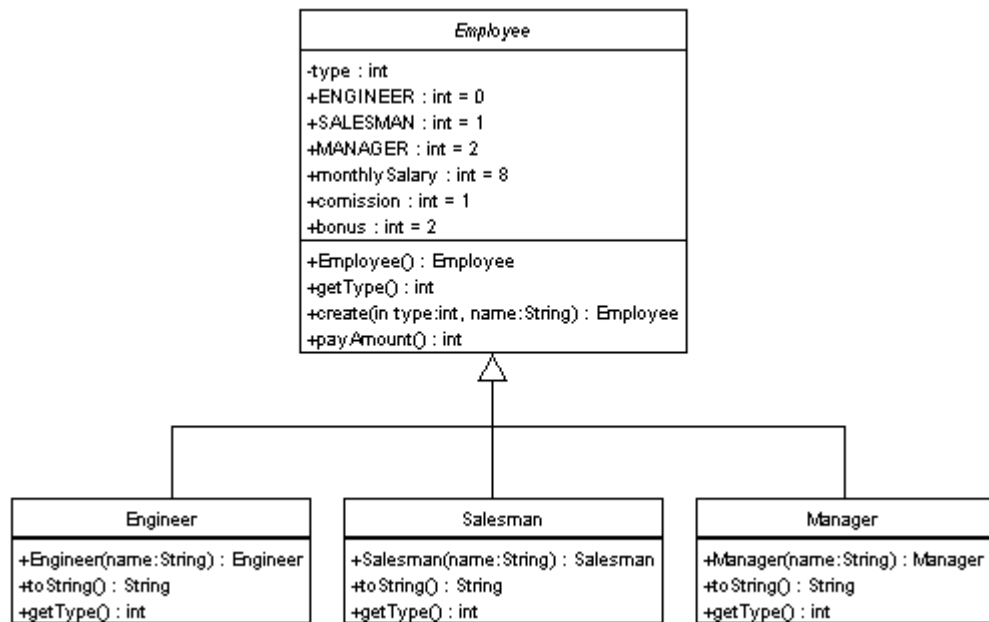
```

        case MANAGER:
            return monthlySalary + bonus;
        default:
            throw new RuntimeException("Incorrect Employee");
    }
}
public String toString(){
    return name;
}
}
}

class Payment1{
    public static void main(String[] args){
        ArrayList office = new ArrayList();
        Employee adhia = new Employee(Employee.MANAGER, "Adhia");
        Employee vina = new Employee(Employee.SALESMAN, "Vina");
        Employee fauzan = new Employee(Employee.ENGINEER, "Fauzan");
        office.add(adhia);
        office.add(vina);
        office.add(fauzan);
        Iterator it = office.iterator();
        while (it.hasNext()) {
            Employee each = (Employee)it.next();
            System.out.println("This month we pay " + each.payAmount() + " to " + each);
        }
    }
}

```

Selanjutnya dilakukan subclassing dari class Employee



```

package learn.refactoring.payment2;
import java.util.*;
abstract class Employee {
    private int type;
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2;

    int monthlySalary = 8;
    int comission = 1;
    int bonus = 2;
    Employee() {
    }
}

```



```
abstract int getType();
static Employee create(int type, String name){
    switch(type) {
        case ENGINEER:
            return new Engineer(name);
        case SALESMAN:
            return new Salesman(name);
        case MANAGER:
            return new Manager(name);
        default:
            throw new IllegalArgumentException("Incorrect type code value");
    }
}
int payAmount(){
    switch(getType()){
        case ENGINEER:
            return monthlySalary;
        case SALESMAN:
            return monthlySalary + comission;
        case MANAGER:
            return monthlySalary + bonus;
        default:
            throw new RuntimeException("Incorrect Employee");
    }
}
}

class Engineer extends Employee{
    private String name;
    public Engineer(String name){
        super();
        this.name = name;
    }
    public String toString(){
        return name;
    }
    int getType(){
        return Employee.ENGINEER;
    }
}

class Salesman extends Employee{
    private String name;
    public Salesman(String name){
        super();
        this.name = name;
    }
    public String toString(){
        return name;
    }
    int getType(){
        return Employee.SALESMAN;
    }
}

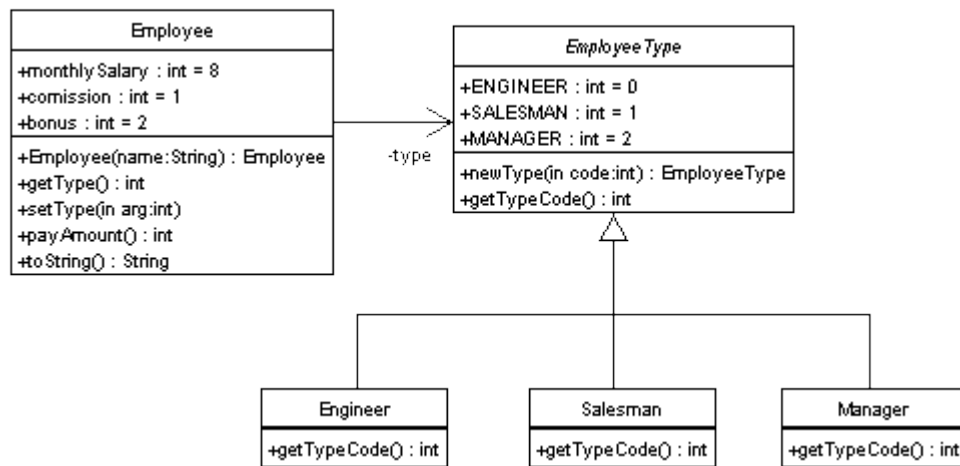
class Manager extends Employee{
    private String name;
    public Manager(String name){
        super();
        this.name = name;
    }
    public String toString(){
        return name;
    }
    int getType(){
        return Employee.MANAGER;
    }
}

class Payment2{
    public static void main(String[] args){
        ArrayList office = new ArrayList();
        Employee adhia = Employee.create(Employee.MANAGER, "Adhia");
        Employee vina = Employee.create(Employee.SALESMAN, "Vina");
        Employee fauzan = Employee.create(Employee.ENGINEER, "Fauzan");
        office.add(adhia);
        office.add(vina);
        office.add(fauzan);
        Iterator it = office.iterator();
        while (it.hasNext()) {
            Employee each = (Employee)it.next();
        }
    }
}
```

```

        System.out.println("This month we pay " + each.payAmount() + " to " + each);
    }
}
}

```



```

package learn.refactoring.payment3;
import java.util.*;

class Employee {
    private String name;
    private EmployeeType type;
    int monthlySalary = 8;
    int comission = 1;
    int bonus = 2;
    public Employee(String name){
        this.name = name;
    }
    int getType(){
        return type.getTypeCode();
    }
    void setType(int arg){
        type = EmployeeType.newType(arg);
    }
    int payAmount(){
        switch(getType()){
            case EmployeeType.ENGINEER:
                return monthlySalary;
            case EmployeeType.SALESMAN:
                return monthlySalary + comission;
            case EmployeeType.MANAGER:
                return monthlySalary + bonus;
            default:
                throw new RuntimeException("Incorrect Employee");
        }
    }
    public String toString() {
        return name;
    }
}

class EmployeeType {
    int ENGINEER = 0;
    int SALESMAN = 1;
    int MANAGER = 2;
    EmployeeType newType(int code){
        switch(code){
            case ENGINEER:
                return new EmployeeType(ENGINEER);
            case SALESMAN:
                return new EmployeeType(SALESMAN);
            case MANAGER:
                return new EmployeeType(MANAGER);
            default:
                throw new RuntimeException("Incorrect EmployeeType");
        }
    }
    int getTypeCode(){
        return this;
    }
}

class Engineer extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.ENGINEER;
    }
}

class Salesman extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.SALESMAN;
    }
}

class Manager extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.MANAGER;
    }
}

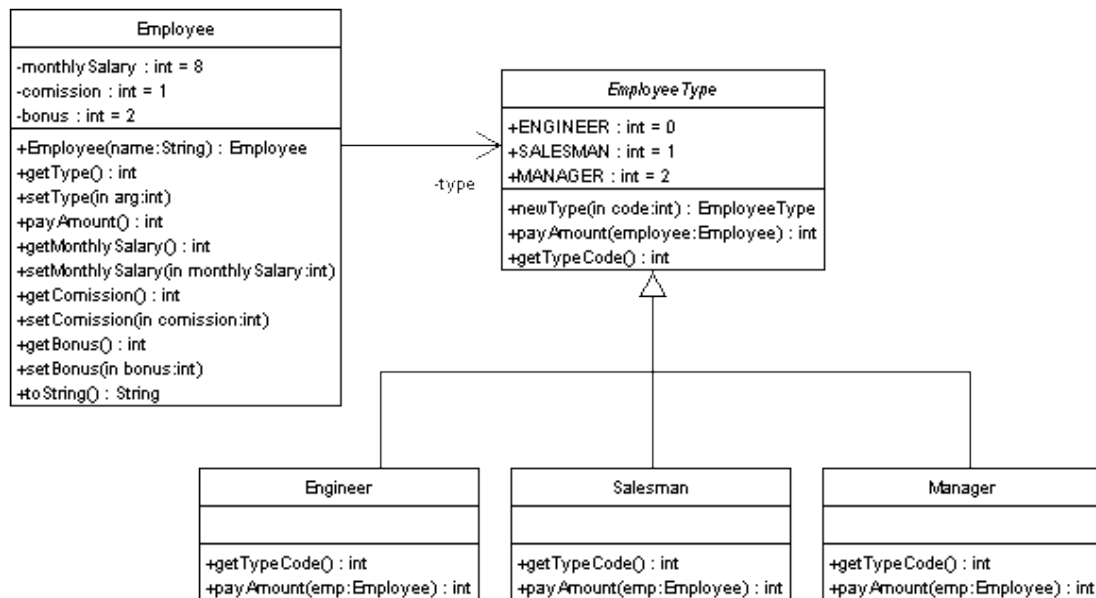
```

```

abstract class EmployeeType {
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2;

    static EmployeeType newType(int code){
        switch(code){
            case ENGINEER:
                return new Engineer();
            case SALESMAN:
                return new Salesman();
            case MANAGER:
                return new Manager();
            default:
                throw new IllegalArgumentException("Incorrect Employee Code");
        }
    }
    abstract int getTypeCode();
}
class Payment3{
    public static void main(String[] args){
        ArrayList office = new ArrayList();
        Employee adhia = new Employee("Adhia");
        adhia.setType(EmployeeType.MANAGER);
        Employee vina = new Employee("Vina");
        vina.setType(EmployeeType.SALESMAN);
        Employee fauzan = new Employee("Fauzan");
        fauzan.setType(EmployeeType.ENGINEER);
        office.add(adhia);
        office.add(vina);
        office.add(fauzan);
        Iterator it = office.iterator();
        while (it.hasNext()) {
            Employee each = (Employee)it.next();
            System.out.println("This month we pay " + each.payAmount() + " to " + each);
        }
    }
}

```



```

package learn.refactoring.payment4;
import java.util.*;
class Employee {
    private String name;
    private EmployeeType type;
    private int monthlySalary = 8;
    private int comission = 1;
    private int bonus = 2;
    public Employee(String name){

```

```
        this.name = name;
    }
    int getType(){
        return type.getTypeCode();
    }
    void setType(int arg){
        type = EmployeeType.newType(arg);
    }
    int payAmount(){
        return type.payAmount(this);
    }
    public int getMonthlySalary() {
        return monthlySalary;
    }
    public void setMonthlySalary(int monthlySalary) {
        this.monthlySalary = monthlySalary;
    }
    public int getComission() {
        return comission;
    }
    public void setComission(int comission) {
        this.comission = comission;
    }
    public int getBonus() {
        return bonus;
    }
    public void setBonus(int bonus) {
        this.bonus = bonus;
    }
    public String toString() {
        return name;
    }
}

class Engineer extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.ENGINEER;
    }
    int payAmount(Employee emp){
        return emp.getMonthlySalary();
    }
}

class Salesman extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.SALESMAN;
    }
    int payAmount(Employee emp){
        return emp.getMonthlySalary() + emp.getComission();
    }
}

class Manager extends EmployeeType {
    int getTypeCode(){
        return EmployeeType.MANAGER;
    }
    int payAmount(Employee emp){
        return emp.getMonthlySalary() + emp.getBonus();
    }
}

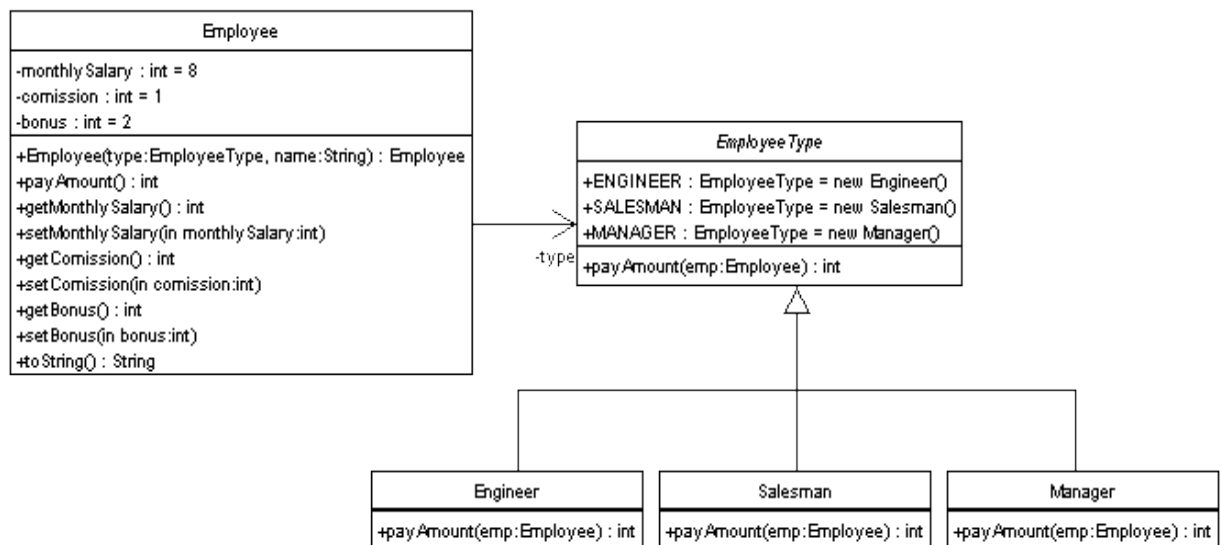
abstract class EmployeeType {
    static final int ENGINEER = 0;
    static final int SALESMAN = 1;
    static final int MANAGER = 2;

    static EmployeeType newType(int code){
        switch(code){
            case ENGINEER:
                return new Engineer();
            case SALESMAN:
                return new Salesman();
            case MANAGER:
                return new Manager();
            default:
                throw new IllegalArgumentException("Incorrect Employee Code");
        }
    }
    abstract int payAmount(Employee employee);
    abstract int getTypeCode();
}
```

```

class Payment4{
    public static void main(String[] args){
        ArrayList office = new ArrayList();
        Employee adhia = new Employee("Adhia");
        adhia.setType(EmployeeType.MANAGER);
        Employee vina = new Employee("Vina");
        vina.setType(EmployeeType.SALESMAN);
        Employee fauzan = new Employee("Fauzan");
        fauzan.setType(EmployeeType.ENGINEER);
        office.add(adhia);
        office.add(vina);
        office.add(fauzan);
        Iterator it = office.iterator();
        while (it.hasNext()) {
            Employee each = (Employee)it.next();
            System.out.println("This month we pay " + each.payAmount() + " to " + each);
        }
    }
}

```



```

package learn.refactoring.payment5;
import java.util.*;
class Employee {
    private String name;
    private EmployeeType type;
    private int monthlySalary = 8;
    private int comission = 1;
    private int bonus = 2;
    public Employee(EmployeeType type, String name){
        this.type = type;
        this.name = name;
    }
    int payAmount(){
        return type.payAmount(this);
    }
    public int getMonthlySalary() {
        return monthlySalary;
    }
    public void setMonthlySalary(int monthlySalary) {
        this.monthlySalary = monthlySalary;
    }
    public int getComission() {
        return comission;
    }
    public void setComission(int comission) {
        this.comission = comission;
    }
    public int getBonus() {
        return bonus;
    }
}

```

```
        public void setBonus(int bonus) {
            this.bonus = bonus;
        }
        public String toString() {
            return name;
        }
    }

class Engineer extends EmployeeType {
    int payAmount(Employee emp){
        return emp.getMonthlySalary();
    }
}

class Salesman extends EmployeeType {
    int payAmount(Employee emp){
        return emp.getMonthlySalary() + emp.getComission();
    }
}

class Manager extends EmployeeType {
    int payAmount(Employee emp){
        return emp.getMonthlySalary() + emp.getBonus();
    }
}

abstract class EmployeeType {
    static EmployeeType ENGINEER = new Engineer();
    static EmployeeType SALESMAN = new Salesman();
    static EmployeeType MANAGER = new Manager();
    abstract int payAmount(Employee emp);
}

class Payment5{
    public static void main(String[] args){
        ArrayList office = new ArrayList();
        Employee adhia = new Employee(EmployeeType.MANAGER, "Adhia");
        Employee vina = new Employee(EmployeeType.SALESMAN, "Vina");
        Employee fauzan = new Employee(EmployeeType.ENGINEER, "Fauzan");
        office.add(adhia);
        office.add(vina);
        office.add(fauzan);
        Iterator it = office.iterator();
        while (it.hasNext()) {
            Employee each = (Employee)it.next();
            System.out.println("This month we pay " + each.payAmount() + " to " + each);
        }
    }
}
```

Bab XIV.Service Oriented Object

A.Data Oriented Matrix

```

package learn.design.dataoriented;

class DataExample {
    public static void main(String[] args) {

        int[][] init1 = { {2, 2}, {2, 2} };
        int[][] init2 = { {1, 2}, {3, 4} };

        Matrix m1 = new Matrix(init1);
        Matrix m2 = new Matrix(init2);

        // Add m1 & m2, store result in a new Matrix object
        Matrix sum = new Matrix(2, 2);
        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                int addend1 = m1.get(i, j);
                int addend2 = m2.get(i, j);
                sum.set(i, j, addend1 + addend2);
            }
        }

        // Print out the sum
        System.out.print("Sum: {");
        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                int val = sum.get(i, j);
                System.out.print(val);
                if (i == 0 || j == 0) {
                    System.out.print(", ");
                }
            }
        }
        System.out.println("}");
    }
}

```

Outputnya:

Sum: {3, 4, 5, 6}

Matrix
-rowCount : int
-colCount : int
+Matrix(in rows:int) : Matrix
+Matrix(in rows:int, in cols:int) : Matrix
+Matrix(in init:int[][]) : Matrix
+get(in row:int, in col:int) : int
+set(in row:int, in col:int, in value:int)
+getRows() : int
+getCols() : int
+clone() : Object
+equals(o:Object) : boolean
+hashCode() : int
-checkValidity(in val:int[][])
-checkIndices(in row:int, in col:int)

Kode sumbernya:

```

package learn.design.dataoriented;

```

```
import java.io.Serializable;
/**
 * Represents a matrix each of whose elements is an <CODE>int</CODE>.
 *
 * @author Bill Venners
 */
public class Matrix implements Serializable, Cloneable {

    private int[][] elements;
    private int rowCount;
    private int colCount;

    public Matrix(int rows) {

        if (rows < 1) {
            throw new IllegalArgumentException();
        }

        elements = new int[rows][rows];
        rowCount = rows;
        colCount = rows;
    }

    public Matrix(int rows, int cols) {

        if (rows < 1 || cols < 1) {
            throw new IllegalArgumentException();
        }

        elements = new int[rows][cols];
        rowCount = rows;
        colCount = cols;
    }

    public Matrix(int[][] init) {

        checkValidity(init);

        elements = (int[][]) init.clone();
        rowCount = init.length;
        colCount = init[0].length;
    }

    public int get(int row, int col) {
        checkIndices(row, col);
        return elements[row][col];
    }

    public void set(int row, int col, int value) {
        checkIndices(row, col);
        elements[row][col] = value;
    }

    public int getRows() {
        return rowCount;
    }
    public int getCols() {
        return colCount;
    }
    public Object clone() {
        try {
            Matrix clone = (Matrix) super.clone();
            clone.elements = (int[][]) elements.clone();
            return clone;
        }
        catch (CloneNotSupportedException e) {
            // Can't happen
            throw new InternalError();
        }
    }

    public boolean equals(Object o) {

        if ((o == null) || (getClass() != o.getClass())) {
            return false;
        }
        Matrix m = (Matrix) o;

        if ((rowCount != m.rowCount) || (colCount != m.colCount)) {
```



```
        return false;
    }
    for (int row = 0; row < rowCount; ++row) {
        for (int col = 0; col < colCount; ++col) {
            if (elements[row][col] != m.elements[row][col]) {
                return false;
            }
        }
    }
    return true;
}

public int hashCode() {
    int retVal = rowCount * colCount;

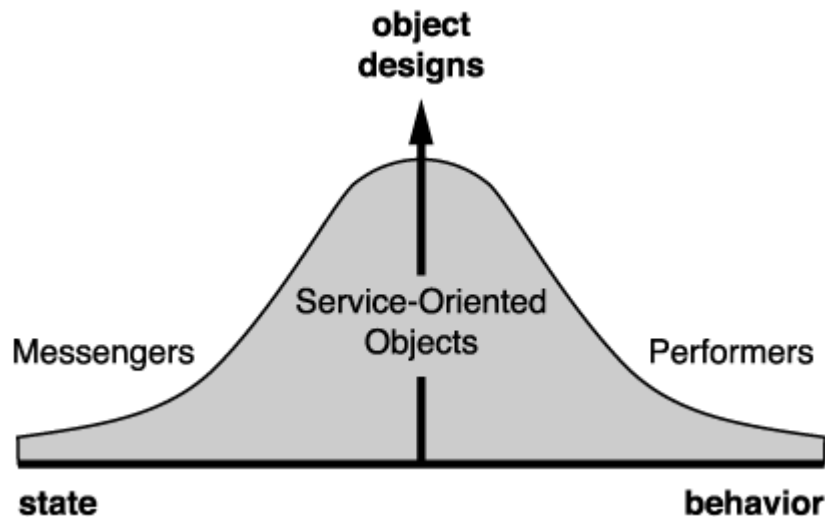
    for (int row = 0; row < rowCount; ++row) {
        for (int col = 0; col < colCount; ++col) {
            retVal ^= elements[row][col];
        }
    }

    return retVal;
}

private static void checkValidity(int[][] val) {
    try {
        int rows = val.length;
        if (rows == 0) {
            throw new IllegalArgumentException();
        }
        int cols = val[0].length;
        if (cols == 0) {
            throw new IllegalArgumentException();
        }
        for (int i = 1; i < rows; ++i) {
            if (val[i].length != cols) {
                throw new IllegalArgumentException();
            }
        }
    }
    catch (NullPointerException e) {
        throw new IllegalArgumentException();
    }
}

private void checkIndices(int row, int col) {
    if (row >= rowCount || row < 0 || col >= colCount || col < 0) {
        throw new IndexOutOfBoundsException();
    }
}
}
```

B.Service Oriented Matrix



```
package learn.design.serviceoriented;
class ServiceExample {
    public static void main(String[] args) {

        int[][] init1 = { {2, 2}, {2, 2} };
        int[][] init2 = { {1, 2}, {3, 4} };

        Matrix m1 = new Matrix(init1);
        Matrix m2 = new Matrix(init2);

        // Add m1 & m2, store result in a new matrix object
        Matrix sum = m1.add(m2);

        // Print out the sum
        System.out.println("Sum: " + sum.toString());
    }
}
```

Outputnya adalah:
Sum: ((3, 4), (5, 6))

Perhatikan bedanya dengan diagram yang di atas

Matrix
-rowCount : int
-colCount : int
+Matrix(in rows:int) : Matrix
+Matrix(in rows:int, in cols:int) : Matrix
+Matrix(in init:int[][]) : Matrix
+get(in row:int, in col:int) : int
+getRows() : int
+getCols() : int
+add(addend:Matrix) : Matrix
+subtract(subtrahend:Matrix) : Matrix
+multiply(in scalar:int) : Matrix
+multiply(multiplier:Matrix) : Matrix
+toString() : String
+clone() : Object
+equals(o:Object) : boolean
+hashCode() : int
-checkValidity(in val:int[][])

Matrix.java

```
package learn.design.serviceoriented;
import java.io.Serializable;

/**
```

```
* A two-dimensional matrix of <CODE>int</CODE>s.
*
* <P>
* The <em>order</em> of the matrix is its number of rows and columns. For
* example, the order of a matrix with 5 rows and 4 columns is "5 by 4." A
* matrix with the same number of rows and columns, such as a 3 by 3 matrix,
* is a <em>square matrix</em>. A matrix all of whose elements is zero is a
* <em>zero matrix</em>.
*
* <P>
* Instances of <CODE>Matrix</CODE> are immutable.
*
* @author Bill Venners
*/
public class Matrix implements Serializable, Cloneable {

    private int[][] elements;
    private int rowCount;
    private int colCount;

    public Matrix(int rows) {
        if (rows < 1) {
            throw new IllegalArgumentException();
        }
        elements = new int[rows][rows];
        rowCount = rows;
        colCount = rows;
    }

    public Matrix(int rows, int cols) {
        if (rows < 1 || cols < 1) {
            throw new IllegalArgumentException();
        }
        elements = new int[rows][cols];
        rowCount = rows;
        colCount = cols;
    }

    public Matrix(int[][] init) {
        checkValidity(init);
        elements = (int[][]) init.clone();
        rowCount = init.length;
        colCount = init[0].length;
    }

    public int get(int row, int col) {
        if (row >= rowCount || row < 0 || col >= colCount || col < 0) {
            throw new IndexOutOfBoundsException();
        }
        return elements[row][col];
    }

    public int getRows() {
        return rowCount;
    }

    public int getCols() {
        return colCount;
    }

    public Matrix add(Matrix addend) {
        if ((addend.rowCount != rowCount) || (addend.colCount != colCount)) {
            throw new IllegalArgumentException();
        }

        Matrix retVal = new Matrix(elements);
        for (int row = 0; row < rowCount; ++row) {
            for (int col = 0; col < colCount; ++col) {
                retVal.elements[row][col] += addend.elements[row][col];
            }
        }
        return retVal;
    }

    public Matrix subtract(Matrix subtrahend) {
        if ((subtrahend.rowCount != rowCount) || (subtrahend.colCount != colCount)) {
            throw new IllegalArgumentException();
        }

        Matrix retVal = new Matrix(elements);
        for (int row = 0; row < rowCount; ++row) {
            for (int col = 0; col < colCount; ++col) {
```

```

        retVal.elements[row][col] -= subtrahend.elements[row][col];
    }
}
return retVal;
}
public Matrix multiply(int scalar) {
    Matrix retVal = new Matrix(elements);
    for (int row = 0; row < rowCount; ++row) {
        for (int col = 0; col < colCount; ++col) {
            retVal.elements[row][col] *= scalar;
        }
    }
    return retVal;
}

public Matrix multiply(Matrix multiplier) {
    if (colCount != multiplier.rowCount) {
        throw new IllegalArgumentException();
    }
    int resultRows = rowCount;
    int resultCols = multiplier.colCount;
    int[][] resultArray = new int[resultRows][resultCols];

    Matrix retVal = new Matrix(elements);

    for (int row = 0; row < resultRows; ++row) {
        for (int col = 0; col < resultCols; ++col) {
            for (int i = 0; i < colCount; ++i) {

                resultArray[row][col] += elements[row][i]
                    * multiplier.elements[i][col];
            }
        }
    }
    return retVal;
}

public String toString() {
    StringBuffer retVal = new StringBuffer("(");
    for (int row = 0; row < rowCount; ++row) {
        retVal.append("(");
        for (int col = 0; col < colCount; ++col) {
            retVal.append(elements[row][col]);
            if (col != colCount - 1) {
                retVal.append(", ");
            }
        }
        retVal.append(")");
        if (row != rowCount - 1) {
            retVal.append(", ");
        }
    }
    retVal.append(")");
    return retVal.toString();
}

public Object clone() {
    try {
        Matrix clone = (Matrix) super.clone();
        clone.elements = (int[][]) elements.clone();
        return clone;
    }
    catch (CloneNotSupportedException e) {
        // Can't happen
        throw new InternalError();
    }
}

public boolean equals(Object o) {
    if ((o == null) || (getClass() != o.getClass())) {
        return false;
    }
    Matrix m = (Matrix) o;
    if ((rowCount != m.rowCount) || (colCount != m.colCount)) {
        return false;
    }
    for (int row = 0; row < rowCount; ++row) {
        for (int col = 0; col < colCount; ++col) {

            if (elements[row][col] != m.elements[row][col]) {

```

```

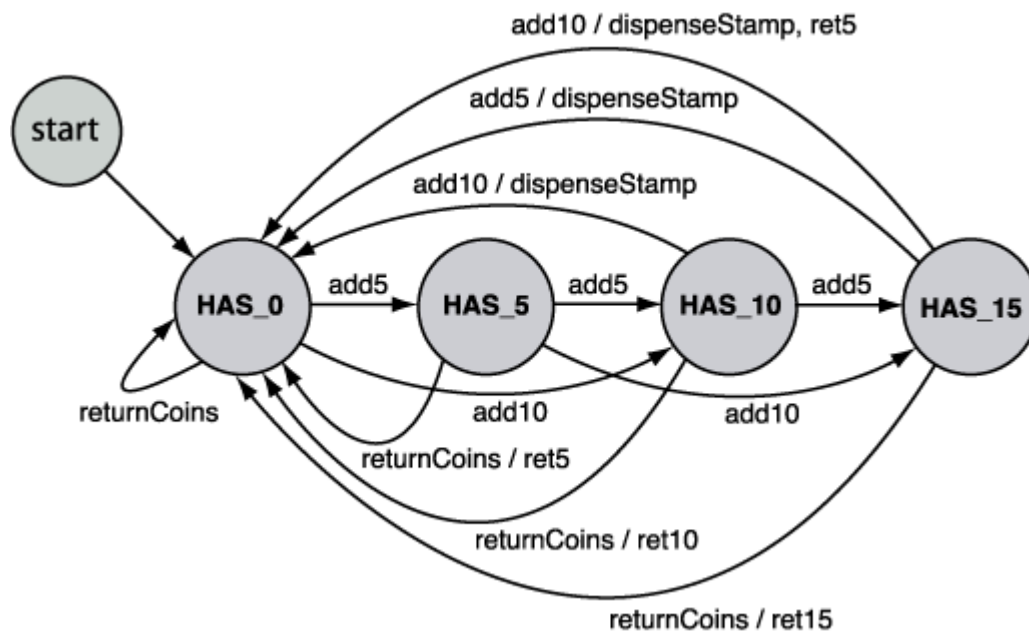
        return false;
    }
}
return true;
}

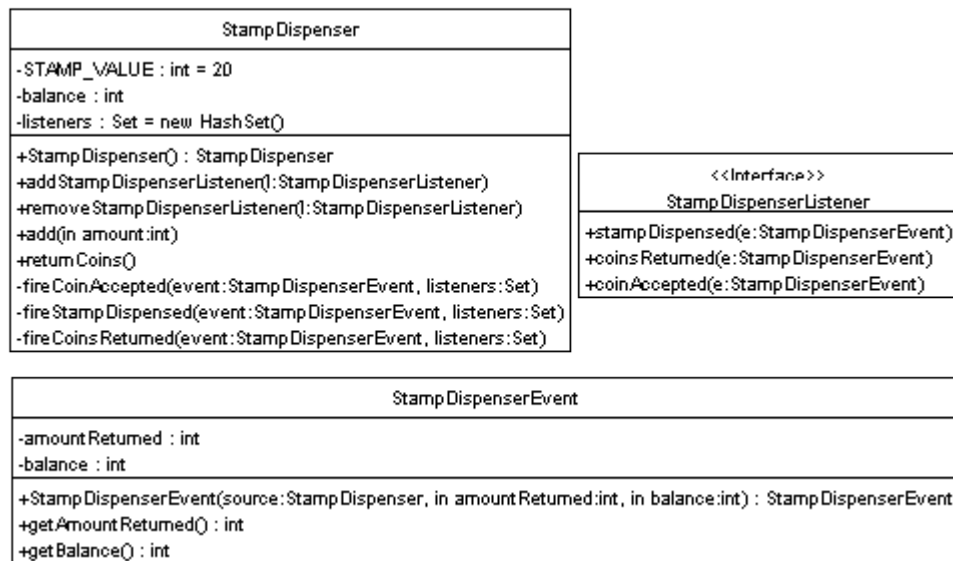
public int hashCode() {
    int retVal = rowCount * colCount;
    for (int row = 0; row < rowCount; ++row) {
        for (int col = 0; col < colCount; ++col) {
            retVal *= elements[row][col];
        }
    }
    return retVal;
}

private static void checkValidity(int[][] val) {
    try {
        int rows = val.length;
        if (rows == 0) {
            throw new IllegalArgumentException();
        }
        int cols = val[0].length;
        if (cols == 0) {
            throw new IllegalArgumentException();
        }
        for (int i = 1; i < rows; ++i) {
            if (val[i].length != cols) {
                throw new IllegalArgumentException();
            }
        }
    }
    catch (NullPointerException e) {
        throw new IllegalArgumentException();
    }
}
}
}

```

C. Service Oriented Stamp Dispenser





1.Versi light dengan pendekatan objek

```

package learn.design.stampdispenser.light;

import java.util.Set;
import java.util.Iterator;
import java.util.HashSet;

/**
 * A stamp dispenser that accepts nickels and dimes and dispenses
 * twenty cent stamps.
 *
 * @author Bill Venners
 */
public class StampDispenser {

    private final static int STAMP_VALUE = 20;
    private int balance;
    private Set listeners = new HashSet();
    public StampDispenser() {
    }

    public synchronized void addStampDispenserListener(StampDispenserListener l) {
        listeners.add(l);
    }

    public synchronized void removeStampDispenserListener(StampDispenserListener l) {
        listeners.remove(l);
    }

    public synchronized void add(int amount) {
        if ((amount != 5) && (amount != 10)) {
            throw new IllegalArgumentException();
        }

        balance += amount;

        if (balance >= STAMP_VALUE) {
            StampDispenserEvent event = new StampDispenserEvent(this, balance - STAMP_VALUE,
0);

```

```
        balance = 0;
        fireStampDispensed(event, listeners);
    }
    else {

        StampDispenserEvent event = new StampDispenserEvent(this, amount, balance);
        fireCoinAccepted(event, listeners);
    }
}

public synchronized void returnCoins() {

    if (balance > 0) {

        StampDispenserEvent event = new StampDispenserEvent(this, balance, 0);
        balance = 0;
        fireCoinsReturned(event, listeners);
    }
}

private static void fireCoinAccepted(StampDispenserEvent event,
Set listeners) {

    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.coinAccepted(event);
    }
}

private static void fireStampDispensed(StampDispenserEvent event,
Set listeners) {

    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.stampDispensed(event);
    }
}

private static void fireCoinsReturned(StampDispenserEvent event,
Set listeners) {

    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.coinsReturned(event);
    }
}
}

package learn.design.stampdispenser.light;
import java.util.EventObject;
public class StampDispenserEvent extends java.util.EventObject {

    private int amountReturned;
    private int balance;
    public StampDispenserEvent(StampDispenser source, int amountReturned,
int balance) {

        super(source);

        if (balance != 0 && balance != 5 && balance != 10
&& balance != 15) {

            throw new IllegalArgumentException();
        }

        if (amountReturned != 0 && amountReturned != 5 && amountReturned != 10
&& amountReturned != 15) {

            throw new IllegalArgumentException();
        }

        this.amountReturned = amountReturned;
        this.balance = balance;
    }

    public int getAmountReturned() {
        return amountReturned;
    }
}
```

```

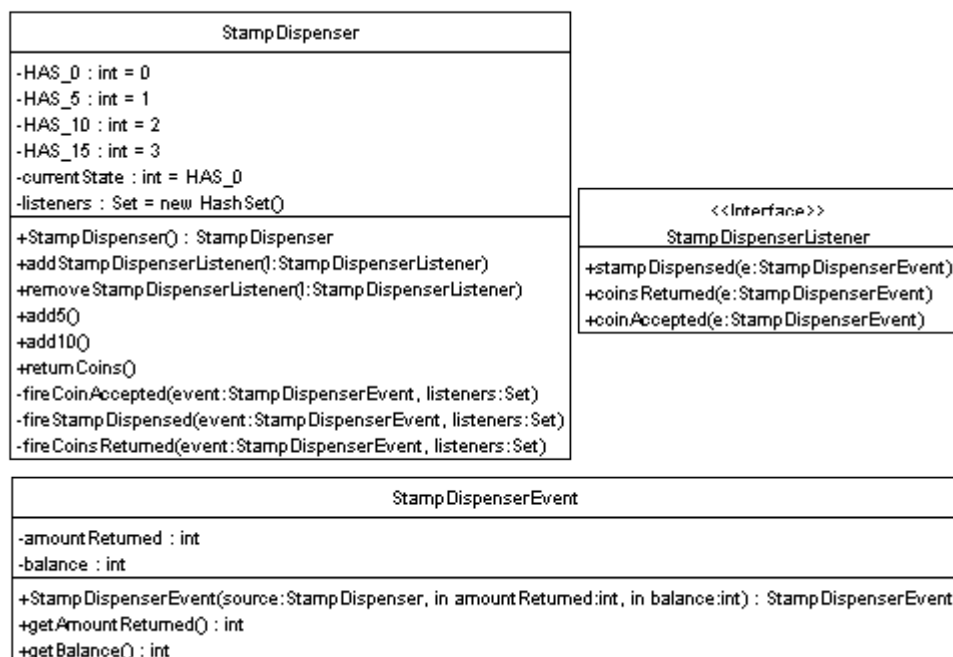
        public int getBalance() {
            return balance;
        }
    }

package learn.design.stampdispenser.light;
public interface StampDispenserListener {

    void stampDispensed(StampDispenserEvent e);
    void coinsReturned(StampDispenserEvent e);
    void coinAccepted(StampDispenserEvent e);
}
    
```

2. Dengan pendekatan prosedural

Current State	Message	Action	Next State
HAS_0	add5		HAS_5
HAS_0	add10		HAS_10
HAS_0	returnCoins		HAS_0
HAS_5	add5		HAS_10
HAS_5	add10		HAS_15
HAS_5	returnCoins	ret5	HAS_0
HAS_10	add5		HAS_15
HAS_10	add10	dispenseStamp	HAS_0
HAS_10	returnCoins	ret10	HAS_0
HAS_15	add5	dispenseStamp	HAS_0
HAS_15	add10	dispenseStamp,ret5	HAS_0
HAS_15	returnCoins	ret15	HAS_0
Initial State: HAS_0			




```
package learn.design.stampdispenser.heavy;

import java.util.Set;
import java.util.Iterator;
import java.util.HashSet;

/**
 * A stamp dispenser that accepts nickels and dimes and dispenses twenty cent
 * stamps.
 *
 * @author Bill Venners
 */
public class StampDispenser {

    private static final int HAS_0 = 0;
    private static final int HAS_5 = 1;
    private static final int HAS_10 = 2;
    private static final int HAS_15 = 3;

    private int currentState = HAS_0;
    private Set listeners = new HashSet();
    public StampDispenser() {
    }

    public synchronized void addStampDispenserListener(
        StampDispenserListener l) {

        listeners.add(l);
    }
    public synchronized void removeStampDispenserListener(
        StampDispenserListener l) {

        listeners.remove(l);
    }
    public synchronized void add5() {

        switch (currentState) {

            case HAS_0:
                StampDispenserEvent event = new StampDispenserEvent(this, 0, 5);
                fireCoinAccepted(event, listeners);
                currentState = HAS_5;
                break;

            case HAS_5:
                event = new StampDispenserEvent(this, 0, 10);
                fireCoinAccepted(event, listeners);
                currentState = HAS_10;
                break;

            case HAS_10:
                event = new StampDispenserEvent(this, 0, 15);
                fireCoinAccepted(event, listeners);
                currentState = HAS_15;
                break;

            case HAS_15:

                event = new StampDispenserEvent(this, 0, 0);
                fireStampDispensed(event, listeners);
                currentState = HAS_0;
                break;

        }
    }
    public synchronized void add10() {

        switch (currentState) {

            case HAS_0:
                StampDispenserEvent event = new StampDispenserEvent(this,
                    0, 10);
                fireCoinAccepted(event, listeners);
        }
    }
}
```

```
        currentState = HAS_10;
        break;

    case HAS_5:
        event = new StampDispenserEvent(this, 0, 15);
        fireCoinAccepted(event, listeners);
        currentState = HAS_15;
        break;

    case HAS_10:
        event = new StampDispenserEvent(this, 0, 0);
        fireStampDispensed(event, listeners);
        currentState = HAS_0;
        break;

    case HAS_15:

        event = new StampDispenserEvent(this, 5, 0);
        fireStampDispensed(event, listeners);
        currentState = HAS_0;
        break;
    }
}

public synchronized void returnCoins() {

    switch (currentState) {

        case HAS_0:
            currentState = HAS_0;
            break;

        case HAS_5:
            StampDispenserEvent event = new StampDispenserEvent(this, 5, 0);
            fireCoinsReturned(event, listeners);
            currentState = HAS_0;
            break;

        case HAS_10:
            event = new StampDispenserEvent(this, 10, 0);
            fireCoinsReturned(event, listeners);
            currentState = HAS_0;
            break;

        case HAS_15:

            event = new StampDispenserEvent(this, 15, 0);
            fireCoinsReturned(event, listeners);
            currentState = HAS_0;
            break;
    }
}

private static void fireCoinAccepted(StampDispenserEvent event,
Set listeners) {

    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.coinAccepted(event);
    }
}

private static void fireStampDispensed(StampDispenserEvent event,
Set listeners) {

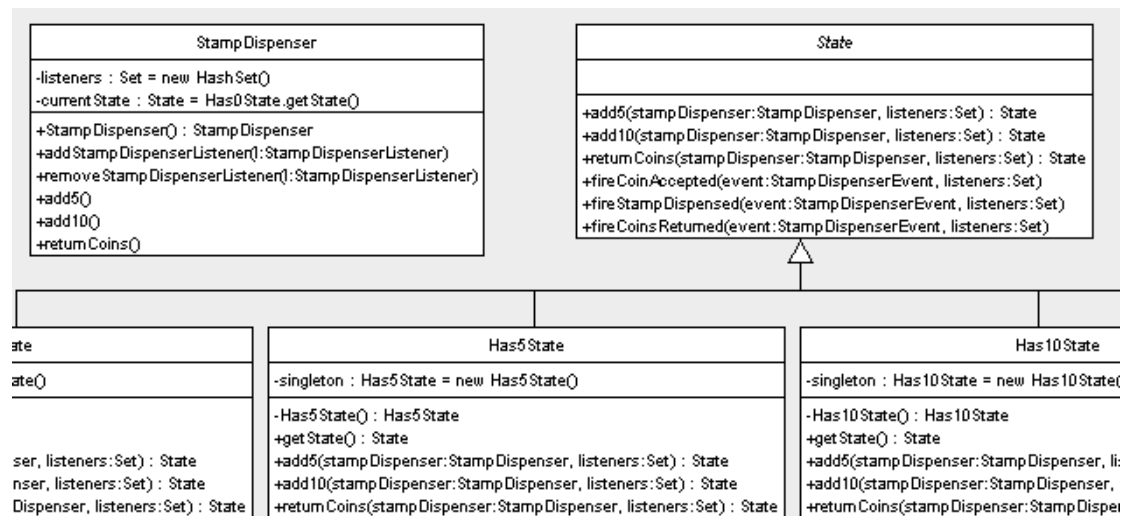
    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.stampDispensed(event);
    }
}

private static void fireCoinsReturned(StampDispenserEvent event,
Set listeners) {

    Iterator it = listeners.iterator();
    while (it.hasNext()) {
        StampDispenserListener l = (StampDispenserListener) it.next();
        l.coinsReturned(event);
    }
}
}
```

}

3.Implementasi yang berbeda dengan State pattern



```

package learn.design.stampdispenser.medium;

import java.util.Iterator;
import java.util.Set;

abstract class State {
    abstract State add5(StampDispenser stampDispenser, Set listeners);
    abstract State add10(StampDispenser stampDispenser, Set listeners);
    abstract State returnCoins(StampDispenser stampDispenser, Set listeners);

    static void fireCoinAccepted(StampDispenserEvent event, Set listeners) {
        Iterator it = listeners.iterator();
        while (it.hasNext()) {
            StampDispenserListener l = (StampDispenserListener) it.next();
            l.coinAccepted(event);
        }
    }

    static void fireStampDispensed(StampDispenserEvent event, Set listeners) {
        Iterator it = listeners.iterator();
        while (it.hasNext()) {
            StampDispenserListener l = (StampDispenserListener) it.next();
            l.stampDispensed(event);
        }
    }

    static void fireCoinsReturned(StampDispenserEvent event, Set listeners) {
        Iterator it = listeners.iterator();
        while (it.hasNext()) {
            StampDispenserListener l = (StampDispenserListener) it.next();
            l.coinsReturned(event);
        }
    }
}

```

```

package learn.design.stampdispenser.medium;
import java.util.Set;

class Has0State extends State {
    private static Has0State singleton = new Has0State();
    private Has0State() {
    }
    static State getState() {
    }
}

```

```
        return singleton;
    }
    State add5(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 5);
        fireCoinAccepted(event, listeners);
        return Has5State.getState();
    }
    State add10(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 10);
        fireCoinAccepted(event, listeners);
        return Has10State.getState();
    }
    State returnCoins(StampDispenser stampDispenser, Set listeners) {
        return this;
    }
}
```

```
package learn.design.stampdispenser.medium;
import java.util.Set;

class Has5State extends State {

    private static Has5State singleton = new Has5State();
    private Has5State() {
    }
    static State getState() {
        return singleton;
    }
    State add5(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 10);
        fireCoinAccepted(event, listeners);
        return Has10State.getState();
    }
    State add10(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 15);
        fireCoinAccepted(event, listeners);
        return Has15State.getState();
    }
    State returnCoins(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 5, 0);
        fireCoinsReturned(event, listeners);
        return Has0State.getState();
    }
}
```

```
package learn.design.stampdispenser.medium;
import java.util.Set;

class Has10State extends State {
    private static Has10State singleton = new Has10State();
    private Has10State() {
    }
    static State getState() {
        return singleton;
    }
    State add5(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 15);
        fireCoinAccepted(event, listeners);
        return Has15State.getState();
    }
    State add10(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 0);
        fireStampDispensed(event, listeners);
        return Has0State.getState();
    }
    State returnCoins(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 10, 0);
        fireCoinsReturned(event, listeners);
        return Has0State.getState();
    }
}
```

```
package learn.design.stampdispenser.medium;

import java.util.Set;
class Has15State extends State {
    private static Has15State singleton = new Has15State();
    private Has15State() {
    }
    static State getState() {
        return singleton;
    }

    State add5(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 0, 0);
        fireStampDispensed(event, listeners);
        return Has0State.getState();
    }
    State add10(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 5, 0);
        fireStampDispensed(event, listeners);
        return Has0State.getState();
    }
    State returnCoins(StampDispenser stampDispenser, Set listeners) {
        StampDispenserEvent event = new StampDispenserEvent(stampDispenser, 15, 0);
        fireCoinsReturned(event, listeners);
        return Has0State.getState();
    }
}
```

```
package learn.design.stampdispenser.medium;
import java.util.Set;
import java.util.HashSet;

/**
 * A stamp dispenser that accepts nickels and dimes and dispenses twenty cent
 * stamps.
 *
 * @author Bill Venners
 */
public class StampDispenser {
    private State currentState = Has0State.getState();
    private Set listeners = new HashSet();

    public StampDispenser() {
    }
    public synchronized void addStampDispenserListener(
        StampDispenserListener l) {
        listeners.add(l);
    }
    public synchronized void removeStampDispenserListener(
        StampDispenserListener l) {
        listeners.remove(l);
    }
    public void add5() {
        currentState = currentState.add5(this, listeners);
    }
    public void add10() {
        currentState = currentState.add10(this, listeners);
    }
    public void returnCoins() {
        currentState = currentState.returnCoins(this, listeners);
    }
}
```

Bab XV.Account by Bill Venner

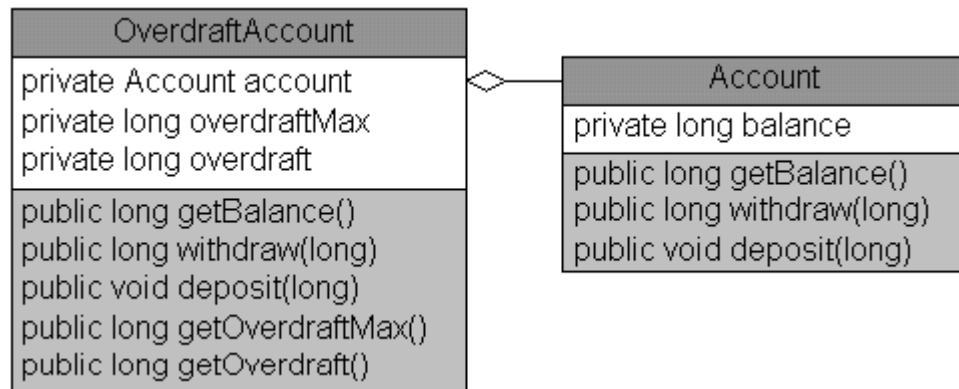
Account
private long balance
public long getBalance() public long withdraw(long) public void deposit(long)

```
package learn.reuse.account;

public class Account {
    private long balance;
    public long withdraw(long amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        if (amount > balance) {
            throw new InsufficientFundsException(amount - balance);
        }
        balance -= amount;
        return amount;
    }
    public void deposit(long amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        long newBal = balance + amount;
        if (newBal < 0) {
            throw new ArithmeticException();
        }
        balance = newBal;
    }
    public long getBalance() {
        return balance;
    }
}
```

```
package learn.reuse.account;

public class InsufficientFundsException extends Exception {
    private long shortfall;
    public InsufficientFundsException(long shortfall) {
        if (shortfall <= 0) {
            throw new IllegalArgumentException();
        }
        this.shortfall = shortfall;
    }
    public InsufficientFundsException(String message, long shortfall) {
        super(message);
        if (shortfall <= 0) {
            throw new IllegalArgumentException();
        }
        this.shortfall = shortfall;
    }
}
```



```

package learn.reuse.account.composition;
import learn.reuse.account.*;
public class OverdraftAccount {
    private Account account = new Account();
    private final long overdraftMax;
    private long overdraft;

    public OverdraftAccount(long overdraftMax) {
        this.overdraftMax = overdraftMax;
    }
    public long getOverdraft() {
        return overdraft;
    }
    public long getOverdraftMax() {
        return overdraftMax;
    }
    public long getBalance() {
        return account.getBalance();
    }
    public long withdraw(long amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        long bal = account.getBalance();

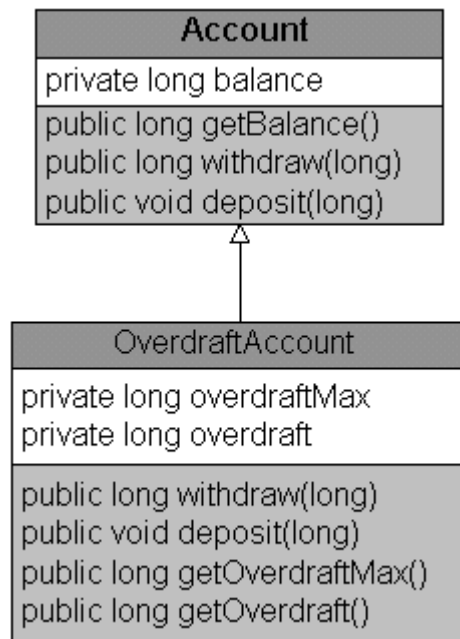
        if (bal >= amount) {
            return account.withdraw(amount);
        }
        long shortfall = amount - bal;
        long extraAvailable = overdraftMax - overdraft;

        if (shortfall > extraAvailable) {
            throw new InsufficientFundsException(shortfall
                - extraAvailable);
        }
        overdraft += shortfall;
        account.withdraw(amount - shortfall);

        return amount;
    }
    public void deposit(long amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }

        if (overdraft > 0) {
            if (amount < overdraft) {
                overdraft -= amount;
            }
            else {
                long diff = amount - overdraft;
                overdraft = 0;
                account.deposit(diff);
            }
        }
        else {
            account.deposit(amount);
        }
    }
}
  
```

}



```

package learn.reuse.account.inheritance;
import learn.reuse.account.*;

public class OverdraftAccount extends Account {
    private final long overdraftMax;
    private long overdraft;

    public OverdraftAccount(long overdraftMax) {
        if (overdraftMax < 0) {
            throw new IllegalArgumentException();
        }
        this.overdraftMax = overdraftMax;
    }
    public long getOverdraft() {
        return overdraft;
    }
    public long getOverdraftMax() {
        return overdraftMax;
    }
    public long withdraw(long amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        long bal = getBalance();

        if (bal >= amount) {
            return super.withdraw(amount);
        }
        long shortfall = amount - bal;
        long extraAvailable = overdraftMax - overdraft;

        if (shortfall > extraAvailable) {
            throw new InsufficientFundsException(shortfall - extraAvailable);
        }
    }
}
  
```

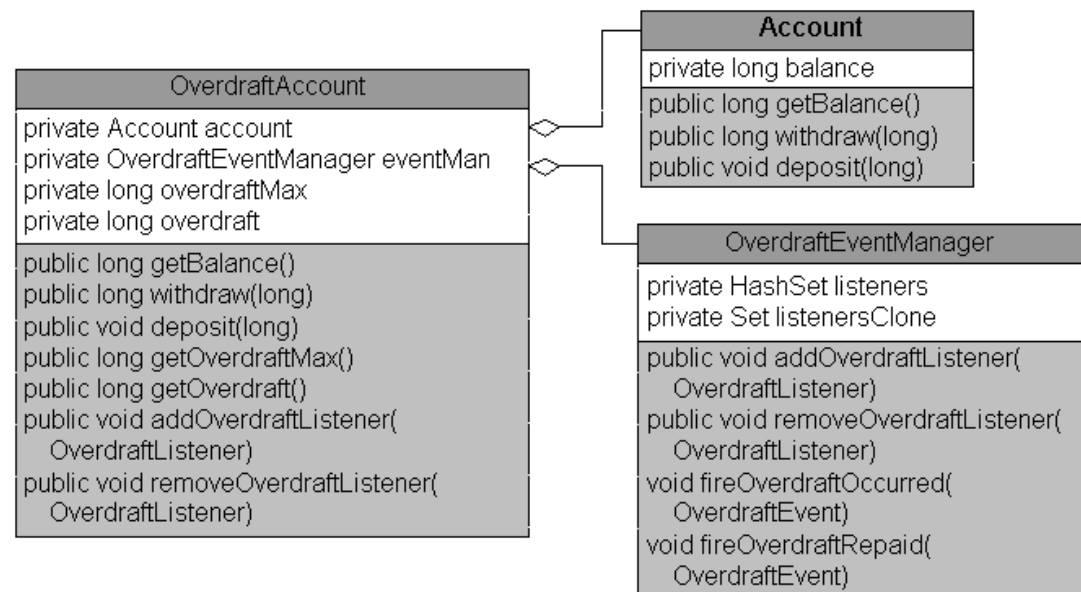


```

        overdraft += shortfall;
        super.withdraw(amount - shortfall);

        return amount;
    }
    public void deposit(long amount) {
        if (overdraft > 0) {
            if (amount < overdraft) {
                overdraft -= amount;
            }
            else {
                long diff = amount - overdraft;
                overdraft = 0;
                super.deposit(diff);
            }
        }
        else {
            super.deposit(amount);
        }
    }
}

```



```

package learn.reuse.account.compositionWithEvent;
import java.util.EventListener;

```

```

public interface OverdraftListener extends EventListener {
    void overdraftOccurred(OverdraftEvent e);
    void overdraftRepaid(OverdraftEvent e);
}

```

```

package learn.reuse.account.compositionWithEvent;

```

```

public class OverdraftEvent extends java.util.EventObject {
    private long overdraft;
    private long amount;

```

```

    public OverdraftEvent(OverdraftAccount source, long overdraft, long amount) {
        super(source);
        if (overdraft <= 0 || amount <= 0) {
            throw new IllegalArgumentException();
        }
        this.overdraft = overdraft;
        this.amount = amount;
    }
    public long getOverdraft() {
        return overdraft;
    }
    public long getAmount() {
        return amount;
    }
}

```

```
}

package learn.reuse.account.compositionWithEvent;
import java.util.Set;
import java.util.Iterator;
import java.util.HashSet;

class OverdraftEventManager {
    private HashSet listeners = new HashSet();
    private Set listenersClone = new HashSet();
    public OverdraftEventManager() {
    }
    public synchronized void addOverdraftListener(OverdraftListener l) {
        listeners.add(l);
        listenersClone = (Set) listeners.clone();
    }
    public synchronized void removeOverdraftListener(OverdraftListener l) {
        listeners.remove(l);
        listenersClone = (Set) listeners.clone();
    }
    public void fireOverdraftOccurred(OverdraftEvent event) {
        Iterator it = listenersClone.iterator();
        while (it.hasNext()) {
            OverdraftListener l = (OverdraftListener) it.next();
            l.overdraftOccurred(event);
        }
    }
    public void fireOverdraftRepaid(OverdraftEvent event) {
        Iterator it = listenersClone.iterator();
        while (it.hasNext()) {
            OverdraftListener l = (OverdraftListener) it.next();
            l.overdraftRepaid(event);
        }
    }
}

package learn.reuse.account.compositionWithEvent;
import learn.reuse.account.*;

public class OverdraftAccount {
    private Account account = new Account();
    private OverdraftEventManager eventMan = new OverdraftEventManager();
    private final long overdraftMax;
    private long overdraft;

    public OverdraftAccount(long overdraftMax) {
        this.overdraftMax = overdraftMax;
    }
    public long getOverdraft() {
        return overdraft;
    }
    public long getOverdraftMax() {
        return overdraftMax;
    }
    public long getBalance() {
        return account.getBalance();
    }
    public long withdraw(long amount) throws InsufficientFundsException {
        if (amount <= 0) {
            throw new IllegalArgumentException();
        }
        long bal = account.getBalance();

        if (bal >= amount) {
            return account.withdraw(amount);
        }
        long shortfall = amount - bal;
        long extraAvailable = overdraftMax - overdraft;

        if (shortfall > extraAvailable) {
            throw new InsufficientFundsException(shortfall
                - extraAvailable);
        }
        overdraft += shortfall;
    }
}
```

```

        account.withdraw(amount - shortfall);
        OverdraftEvent event = new OverdraftEvent(this, overdraft, shortfall);
        eventMan.fireOverdraftOccurred(event);

        return amount;
    }
    public void deposit(long amount) {

        if (amount <= 0) {
            throw new IllegalArgumentException();
        }

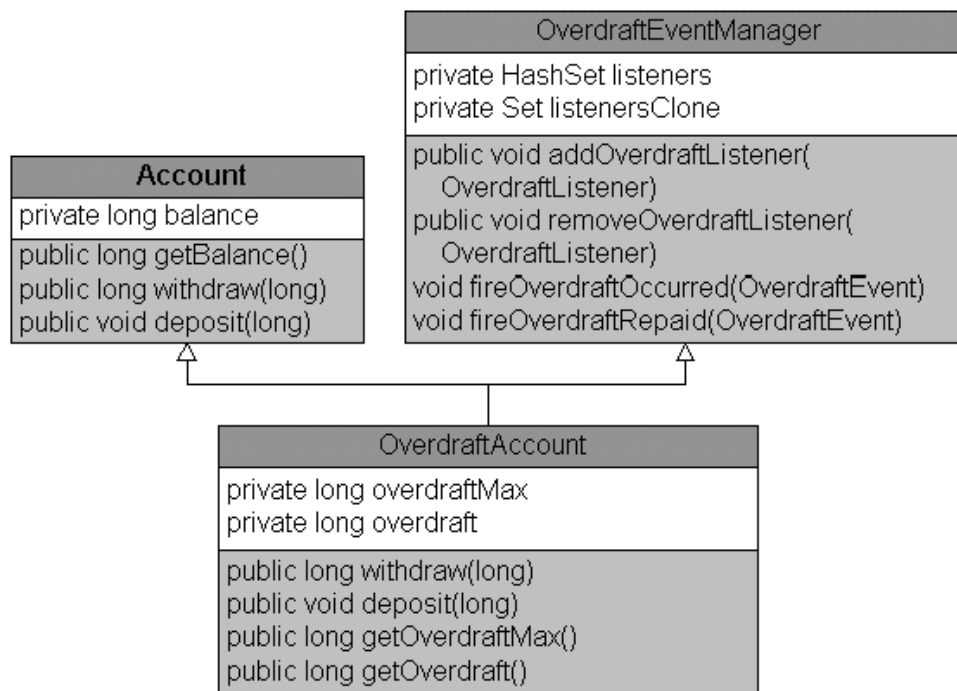
        if (overdraft > 0) {

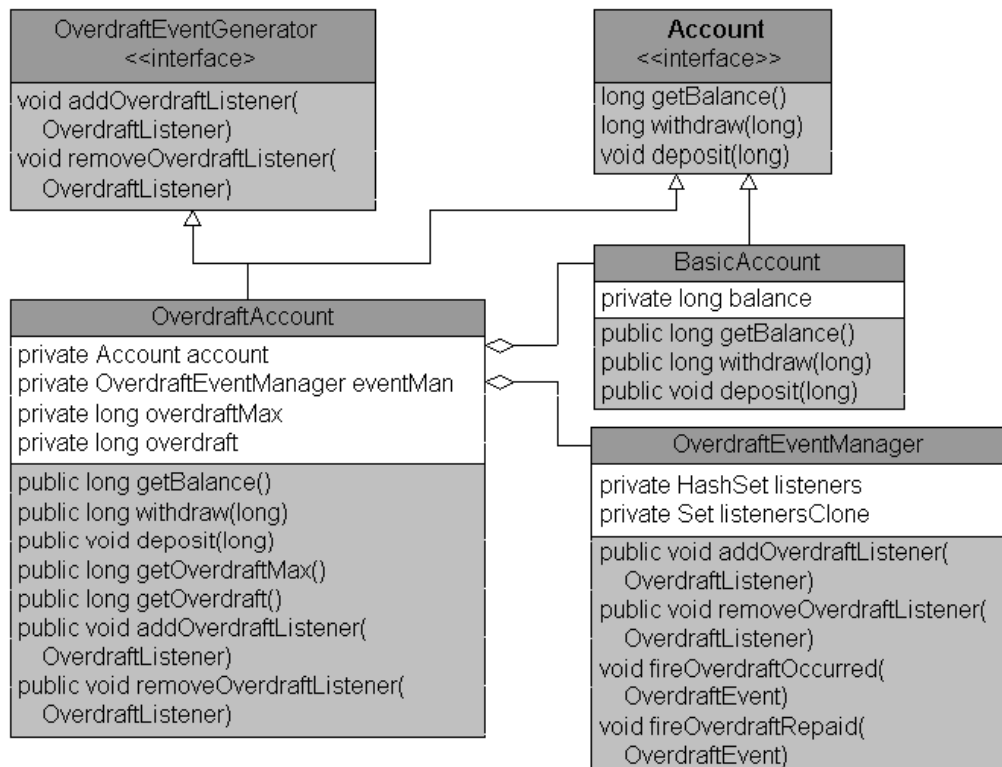
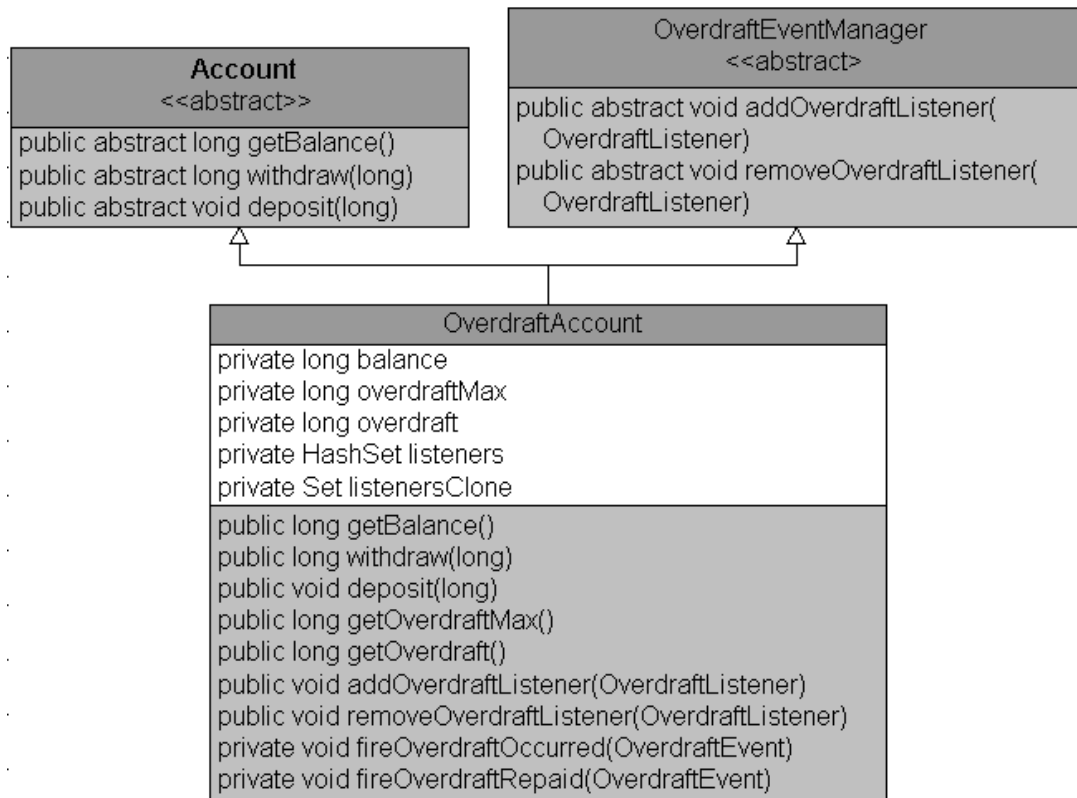
            long amountRepaid = 0;

            if (amount < overdraft) {
                amountRepaid = amount;
                overdraft -= amount;
            }
            else {
                long diff = amount - overdraft;
                amountRepaid = diff;
                overdraft = 0;
                account.deposit(diff);
            }

            OverdraftEvent event = new OverdraftEvent(this, overdraft,
                amountRepaid);
            eventMan.fireOverdraftRepaid(event);
        }
        else {
            account.deposit(amount);
        }
    }
    public void addOverdraftListener(OverdraftListener l) {
        eventMan.addOverdraftListener(l);
    }
    public void removeOverdraftListener(OverdraftListener l) {
        eventMan.removeOverdraftListener(l);
    }
}

```





```

package learn.reuse.account.delegationWithEvent;

public interface Account {
    long withdraw(long amount) throws InsufficientFundsException;
    void deposit(long amount);
    long getBalance();
}
    
```

```
}

public class BasicAccount implements Account{
    private long balance;
    ....
}

package learn.reuse.account.delegationWithEvent;
public interface OverdraftEventGenerator {
    void addOverdraftListener(OverdraftListener l);
    void removeOverdraftListener(OverdraftListener l);
}

class OverdraftEventManager implements OverdraftEventGenerator {
    private HashSet listeners = new HashSet();
    .....
}

public class OverdraftAccount implements Account, OverdraftEventGenerator{
    private Account account = new BasicAccount();
    private OverdraftEventManager eventMan = new OverdraftEventManager();
    ....
}
```

