Bab 1
Review Konsep Dasar dalam Java

1.1 Tujuan

Sebelum melangkah pada fitur-fitur menarik yang ada pada *Java*, mari kita meninjau beberapa hal yang telah Anda pelajari pada pelajaran pemograman pertama Anda. Pelajaran ini menyajikan diskusi tentang perbedaan konsep-konsep berorientasi *object* dalam *Java*.

Setelah melengkapi pelajaran ini, Anda sebaiknya mampu untuk:

1. Mengetahui dan menggunakan konsep dasar beorientasi *object*.
   - *class*
   - *object*
   - atribut
   - *method*
   - konstruktor
2. Mengetahui dengan jelas tentang konsep lanjutan berorientasi *object* dan menggunakannya dengan baik
   - *package*
   - enkapsulasi
   - abstraksi
   - pewarisan
   - polimorfisme
   - *interface*
3. Mengetahui dengan jelas penggunaan kata kunci *this, super, final* dan *static*
4. Membedakan antara *method overloading* dan *method overridding*

1.2 Konsep Berorientasi object

1.2.1 Desain Berorientasi object

Desain berorientasi *object* adalah sebuah teknik yang memfokuskan desain pada *object* dan *class* berdasarkan pada skenario dunia nyata. Hal ini menegaskan keadaan(*state*)*, behaviour* dan interaksi dari *object*. Selain itu juga menyediakan manfaat akan kebebasan pengembangan, meningkatkan kualitas, mempermudah pemeliharaan, mempertinggi kemampuan dalam modifikasi dan meningkatkan penggunaan kembali *software*.

1.2.2 Class

Class mengizinkan Anda dalam mendeklarasikan tipe data baru. Ia dijalankan sebagai *blueprint*, dimana model dari *object* yang Anda buat berdasarkan pada tipe data baru ini.

1.2.3 Object

Sebuah *object* adalah sebuah entiti yang memiliki keadaan, *behaviour* dan identitas yang yang tugasnya dirumuskan dalam suatu lingkup masalah dengan baik. Inilah *instance* sebenarnya dari sebuah *class*. Ini juga dikenal sebagai *instance*. *Instance* dibuat sewaktu Anda meng-*instantiate class* menggunakan kata kunci *new*. Dalam sistem registrasi siswa, contoh dari sebuah object yaitu entiti Student.

1.2.4 Atribut

Atribut menunjuk pada elemen data dari sebuah *object*. Atribut menyimpan informasi tentang *object*. Dikenal juga sebagai *member* data, variabel *instance*, properti atau sebuah *field* data. Kembali lagi ke contoh sistem registrasi siswa, atribut dari sebuah siswa adalah nomor siswa.

1.2.5 Method

Sebuah *method* menjelaskan *behaviour* dari sebuah *object*. *Method* juga dikenal sebagai fungsi atau prosedur. Sebagai contoh, *method* yang mungkin tersedia untuk entiti siswa adalah *method register*.

1.2.6 Konstruktor

Konstruktor adalah sebuah tipe khusus dari *method* yang digunakan untuk membuat dan menginisialisasi sebuah *object* baru. Ingat bahwa konstruktor bukan *member*(yaitu atribut, *method* atau *inner class* dari sebuah *object*).

1.2.7 Package

Package menunjuk pada pengelompokkan *class* dan/atau *subpackages*. Strukturnya dapat disamakan dengan direktorinya.

1.2.8 Enkapsulasi

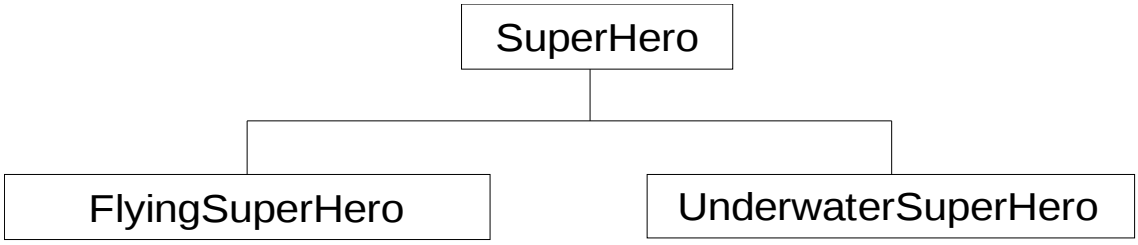
Enkapsulasi menunjuk pada prinsip dari menyembunyikan desain atau mengimplementasikan informasi yang tidak sesuai pada *object* yang ada.

1.2.9 Abstraksi

Sementara enkapsulasi menyembunyikan detail, abstraksi mengabaikan aspek dari subyek yang tidak sesuai dengan tujuan yang ada supaya lebih banyak mengkonsentrasikan yang ada.

1.2.10 Pewarisan

Pewarisan adalah hubungan antara *class* dimana dalam satu *class* ada *superclass* atau *class* induk dari *class* yang lain. Pewarisan menunjuk pada properti dan *behaviour* yang diterima dari nenek moyang dari *class*. Ini dikenal juga sebagai hubungan “is-a”. Perhatikan pada hirarki berikut.



Gambar 1.1: Contoh Pewarisan

SuperHero adalah *superclass* dari *class FlyingSuperHero* dan *UnderwaterSuperHero*. Catatan bahwa *FlyingSuperHero* “is-a” *SuperHero*. Sebagaimana juga *UnderwaterSuperHero* “is-a” *SuperHero*

1.2.11 Polimorfisme

Polimorfisme adalah kemampuan dari sebuah *object* untuk membolehkan mengambil beberapa bentuk yang berbeda. Secara harfiah, “poli” berarti banyak sementara “morph” berarti bentuk. Menunjuk pada contoh sebelumnya pada pewarisan, kita lihat bahwa *object SuperHero* dapat juga menjadi *object FlyingSuperHero* atau *object UnderwaterSuperHero*.

1.2.12 Interface

Sebuah *interface* adalah sebuah *contract* dalam bentuk kumpulan *method* dan deklarasi konstanta. Ketika sebuah class *implements* sebuah *interface*, ini mengimplementasikan semua *method* yang dideklarasikan dalam *interface*.

1.3 Struktur Program Java

Pada bagian ini meringkaskan *syntax* dasar yang digunakan dalam pembuatan aplikasi *Java*.

1.3.1 Mendeklarasikan class Java

```
<classDeclaration> ::=
   <modifier> class <name> {
      <attributeDeclaration>*
      <constructorDeclaration>*
      <methodDeclaration>*
  }
```

dimana <*modifier*> adalah sebuah *access modifier*, yang mana boleh dikombinasikan dengan tipe yang laen dari *modifier*.

Petunjuk Penulisan Program:

** = berarti bahwa boleh ada 0 atau lebih kejadian dari deret tersebut yang menggunakannya juga.*
<description> = menunjukkan bahwa Anda harus mengganti nilai sebenarnya untuk bagian ini daripada menguranginya penulisannya.
Ingat bahwa untuk class teratas, acces modifier yang valid hanyalah public dan package(yakni jika tidak ada acces modifier mengawali kata kunci class).

Contoh berikut ini mendeklarasikan *blueprint SuperHero.*

```
        Class SuperHero {
   String superPowers[];
   void setSuperPowers(String superPowers[])  {
   this.superPowers = superPowers;
   }
   void printSuperPowers() {
   for (int i = 0; i < superPowers.length; i++) {
             System.out.println(superPowers[i]);
        }
 }
        }
```

1.3.2 Mendeklarasikan Atribut

```
<attributeDeclaration> ::=
   <modifier> <type> <name> [= <default_value>];
<type> ::=
   byte | short | int | long | char | float | double | boolean
              | <class>
```

Petunjuk Penulisan Program:
[] = Menunjukkan bahwa bagian ini hanya pilihan.

Inilah contohnya.

```
public class AttributeDemo {
   private String studNum;
   public boolean graduating = false;
   protected float unitsTaken = 0.0f;
   String college;
          }
```

1.3.3 Mendeklarasikan Method

```
<methodDeclaration> ::=
   <modifier> <returnType> <name>(<parameter>*) {
      <statement>*
   }
<parameter> ::=
             <parameter_type> <parameter_name>[,]
```

Sebagai contoh:

```
class MethodDemo {
   int data;
   int getData() {
      return data;
   }
   void setData(int data) {
      this.data = data;
   }
   void setMaxData(int data1, int data2) {
      data = (data1>data2)? data1 : data2;
   }
            }
```

1.3.4 Mendeklarasikan sebuah Konstruktor

```
<constructorDeclaration> ::=
   <modifier> <className> (<parameter>*) {
      <statement>*
            }
```

Jika tidak ada konstruktor yang disediakan secara jelas, konstruktor *default* secara otomatis membuatkannya untuk Anda. Konstruktor *default* tidak membawa argumen dan tidak berisi pernyataan pada tubuh *class*.

Petunjuk Penulisan Program:
Nama konstruktor harus sama dengan nama class.
<modifier> yang valid untuk konstruktor adalah public, protected, dan private.
Konstruktor tidak memiliki nilai return.

Perhatikan contoh berikut.

```
class ConstructorDemo {
   private int data;
   public ConstructorDemo() {
      data = 100;
   }
   ConstructorDemo(int data) {
      this.data = data;
   }
}
```

1.3.5 Meng-instansiasi sebuah class

Untuk meng-*instantiate* sebuah *class*, dengan sederhana kita gunakan kata kunci *new* diikuti dengan pemanggilan sebuah konstruktor. Mari lihat langsung ke contohnya.

```
class ConstructObj {
   int data;
   ConstructObj() {
      /* menginisialisasi data */
   }
   public static void main(String args[]) {
      ConstructObj obj = new ConstructObj();   //di-instantiate
   }
```

```
                    }
```

1.3.6 Mengakses Anggota object

Untuk mengakses anggota dari sebuah *object*, kita gunakan notasi “dot”. Penggunaanya seperti berikut:

```
                    <object>.<member>
```

Contoh selanjutnya berdasar pada sebelumnya dengan pernyataan tambahan untuk mengakses anggota dan *method* tambahan.

```
class ConstructObj {
   int data;
   ConstructObj() {
      /* inisialisasi data */
   }
   void setData(int data) {
      this.data = data;
   }
   public static void main(String args[]) {
      ConstructObj obj = new ConstructObj();   //instantiation
      obj.setData = 10;      //access setData()
      System.out.println(obj.data);//access data
   }
          }
```

1.3.7 Package

Untuk menunjukkan bahwa *file* asal termasuk *package* khusus, kita gunakan *syntax* berikut:

```
<packageDeclaration> ::=
                  package <packageName>;
```

Untuk mengimpor *package* lain, kita gunakan *syntax* berikut:

```
<importDeclaration> ::=
                  import <packageName.elementAccessed>;
```

Dengan ini, *source code* Anda harus memiliki format berikut:

```
                  [<packageDeclaration>]
        <importDeclaration>*
                  <classDeclaration>+
```

Petunjuk Penulisan Program:
+ menunjukkan bahwa boleh ada 1 atau lebih kejadian pada baris ini dalam pengaplikasiannya.

Sebagai contoh.

```
package registration.reports;
import registration.processing.*;
import java.util.List;
import java.lang.*;    //imported by default
class MyClass {
   /* rincian dari MyClass */
```

1.3.8 Acces Modifier

Table berikut meringkas *acces modifier* dalam *Java*.

	private	default/package	***protected***	***public***
class yang sama	Yes	Yes	Yes	Yes
package yang sama		Yes	Yes	Yes
package yang berbeda (subclass)			Yes	Yes
package yang berbeda (non-subclass)				Yes

Tabel 1.2: Acces Modifier

1.3.9 Enkapsulasi

Menyembunyikan elemen dari penggunaan sebuah *class* dapat dilakukan dengan pembuatan anggota yang ingin Anda sembunyikan secara *private*.

Contoh berikut menyembunyikan *field secret*. Catatan bahwa *field* ini tidak langsung diakses oleh *program* lain menggunakan *method getter* dan *setter*.

```
class Encapsulation {
   private int secret;  //field tersembunyi
   public boolean setSecret(int secret) {
      if (secret < 1 || secret > 100) {
         return false;
      }
      this.secret = secret;
      return true;
   }
   public getSecret() {
      return secret;
   }
}
```

1.3.10 Pewarisan

Untuk membuat *class* anak atau *subclass* berdasarkan *class* yang telah ada, kita gunakan kata kunci *extend* dalam mendeklarasikan *class*. Sebuah *class* hanya dapat meng-*extend* satu *class*

induk.
Sebagai contoh, *class Point* di bawah ini adalah *superclass* dari *class ColoredPoint.*

```
import java.awt.*;
class Point {
   int x;
   int y;
}

class ColoredPoint extends Point {
   Color color;
            }
```

1.3.11 Method Overriding

Method subclass override terhadap *method superclass* ketika *subclass* mendeklarasikan *method* yang *signature*nya serupa ke *method* dalam *superclass*. *Signature* dari *method* hanyalah informasi yang ditemukan dalam definisi *method* bagian atas. *Signature* mengikutkan tipe *return*, nama dan daftar *parameter method* tetapi itu tidak termasuk *acces modifier* dan tipe yang lain dari kata kunci seperti *final* dan *static*.

Inilah perbedaan dari *method overloading*. *Method overloading* secara singkat didiskusikan dalam sub bagian pada kata kunci *this*.

```
class Superclass {
   void display(int n) {
      System.out.println("super: " + n);
   }
}

class Subclass extends Superclass {
   void display(int k) {     //method overriding
      System.out.println("sub: " + k);
   }
}

class OverrideDemo {
   public static void main(String args[]) {
      Subclass SubObj = new Subclass();
      Superclass SuperObj = SubObj;
      SubObj.display(3);
      ((Superclass)SubObj).display(4);
   }
            }
```

Ini akan menghasilkan keluaran sebagai berikut.

```
sub: 3
sub: 4
```

Pemanggilan *method* ditentukan oleh tipe data sebenarnya dari *object* yang diminta *method*.

Acces modifier untuk *method* yang dibutuhkan tidak harus sama. Bagaimanapun, *acces modifier* dari *method overridding* mengharuskan salah satunya punya *acces modifier* yang sama seperti itu dari *method overridden* atau *acces modifier* yang kurang dibatasi.
Perhatikan contoh selanjutnya. Periksa yang mana dari *method overridding* berikut akan menyebabkan waktu meng-*compile* akan menyebabkan *error*.

```
class Superclass {
   void overriddenMethod() {
   }
}

class Subclass1 extends Superclass {
   public void overriddenMethod() {
   }
}

class Subclass2 extends Superclass {
   void overriddenMethod() {
   }
}

class Subclass3 extends Superclass {
   protected void overriddenMethod() {
   }
}

class Subclass4 extends Superclass {
   private void overriddenMethod() {
   }
}
```

1.3.12 Class Abstract dan Method

Bentuk umum dari sebuah *method abstract* adalah sebagai berikut:

```
abstract <modifier> <returnType> <name>(<parameter>*);
```

Sebuah *class* yang berisi *method abstract* harus dideklarasikan sebagai sebuah *class abstract.*

```
abstract <modifier> <returnType> <name>(<parameter>*);abstract class
<name> {
   /* constructors, fields and methods */
            }
```

Kata kunci tidak dapat digunakan pada konstruktor atau *method static*. Ini juga penting untuk diingat bahwa class *abstract* tidak dapat di-*instantiate*.

Class yang meng-*extends* sebuah *class abstract* harus mengimplementasikan semua *method abstract.* Jika tidak *subclass* sendiri dapat dideklarasikan sebagai *abstract.*

Petunjuk Penulisan Program:
catatan bahwa mendeklarasikan sebuah method abstract hampir mirip dalam mendeklarasikan class normal kecuali itu suatu method abstract yang tidak memiliki tubuh dan kepala sehingga dengan segera diakhiri dengan semicolon(;).

Sebagai contoh:

```
            abstract class SuperHero {
   String superPowers[];
```

```
   void setSuperPowers(String superPowers[])  {
      this.superPowers = superPowers;
   }
   void printSuperPowers() {
      for (int i = 0; i < superPowers.length; i++) {
         System.out.println(superPowers[i]);
      }
   }
   abstract void displayPower();
}

class UnderwaterSuperHero extends SuperHero {
   void displayPower() {
      System.out.println("Communicate with sea creatures...");
      System.out.println("Fast swimming ability...");
   }
}

class FlyingSuperHero extends SuperHero {
   void displayPower() {
      System.out.println("Fly...");
   }
}
```

1.3.13 Interface

Mendeklarasikan sebuah *interface* pada dasarnya mendeklarasikan sebuah *class* tetapi sebagai penggantinya menggunakan kata kunci *class,* kata kunci *interface* digunakan. Berikut *syntax*nya.

```
<interfaceDeclaration> ::=
   <modifier> interface <name> {
      <attributeDeclaration>*
      [<modifier> <returnType> <name>(<parameter>*);]*
                }
```

Anggotanya adalah *public* ketika *interface* dideklarasikan *public.*

Petunjuk Penulisan Program:
Secara mutlak atribut adalah static dan final dan harus diinisialisasi dengan nilai konstanta.
Seperti mendeklarasikan class teratas, acces modifier yang valid hanyalah public dan package(yakni jika tidak ada acces modifier mengawali kata kunci class).

Class mengimplementasikan sebuah *interface* yang telah ada dengan menggunakan kata kunci *implements. Class* ini dibuat untuk mengimplementasikan semua *method interface*. Sebuah *class* boleh mengimplementasikan lebih dari satu *interface*.

Contoh berikut menunjukkan bagaimana mendeklarasikan dan menggunakan sebuah *interface*.

```
interface MyInterface {
   void iMethod();
}
class MyClass1 implements MyInterface {
   public void iMethod() {
```

```
      System.out.println("Interface method.");
   }

   void myMethod() {
      System.out.println("Another method.");
   }
}

class MyClass2 implements MyInterface {
   public void iMethod() {
      System.out.println("Another implementation.");
   }
}

class InterfaceDemo {
   public static void main(String args[]) {
      MyClass1 mc1 = new MyClass1();
      MyClass2 mc2 = new MyClass2();

      mc1.iMethod();
      mc1.myMethod();
      mc2.iMethod();
   }
}
```

1.3.14 Kata kunci this

Kata kunci *this* dapat digunakan untuk beberapa alasan berikut:

1. Adanya ambigu pada atribut lokal dari variabel lokal
2. Menunjuk pada *object* yang meminta *method non-static*
3. Menunjuk pada konstruktor lain.

Sebagai contoh pada maksud pertama, perhatikan kode berikut dimana variabel *data* disediakan sebagai sebuah atribut dan parameter lokal pada saat yang sama.

```
class ThisDemo1 {
   int data;
   void method(int data) {
      this.data = data;
      /* this.data menunjuk ke atribut
         sementara data menunjuk ke variabel lokal */
   }
         }
```

Contoh berikut menunjukkan bagaimana *object this* secara mutlak menunjuk ketika anggota *non-static* dipanggil.

```
class ThisDemo2 {
   int data;
   void method() {
      System.out.println(data);    //this.data
```

```
        }
        void method2() {
            method();              //this.method();
}
```

Sebelum melihat ke contoh yang lain, mari pertama meninjau pengertian *method overloading*. Konstruktor seperti juga *method* dapat juga menjadi *overload*. *Method* yang berbeda dalam *class* dapat memberi nama yang sama asalkan *list* parameter juga berbeda. *Method overloaded* harus berbeda dalam nomor dan/atau tipe dari parameternya. Contoh selanjutnya memiliki konstruktor *overloaded* dan referensi *this* yang dapat digunakan untuk menunjuk versi lain dari konstruktor.

```
class ThisDemo3 {
   int data;
   ThisDemo3() {
      this(100);
   }
   ThisDemo3(int data) {
      this.data = data;
   }
}
```

Petunjuk Penulisan Program:
Memanggil this() harus ada pernyataan pertama dalam konstruktor.

1.3.15 Kata kunci super

Penggunaan kata kunci *super* berhubungan dengan pewarisan. *Super* digunakan untuk meminta konstruktor *superclass*. *Super* juga dapat digunakan seperti kata kunci *this* untuk menunjuk pada anggota dari *superclass*.

Program berikut mendemonstrasikan bagaimana referensi *super* digunakan untuk memanggil konstruktor *superclass*.

```
class Person {
   String firstName;
   String lastName;
   Person(String fname, String lname) {
      firstName = fname;
      lastName = lname;
   }
}

class Student extends Person {
   String studNum;
   Student(String fname, String lname, String sNum) {
      super(fname, lname);
      studNum = sNum;
   }
         }
```

Petunjuk Penulisan Program:

super() menunjuk pada superclass dengan segera. Ini harus berada pada pernyataan pertama dalam konstruktor superclass.

Kata kunci dapat juga digunakan untuk menunjuk anggota *superclass* seperti yang ditunjukkan pada contoh berikut.

```
class Superclass{
   int a;
   void display_a(){
      System.out.println("a = " + a);
   }
}

class Subclass extends Superclass {
   int a;
   void display_a(){
      System.out.println("a = " + a);
   }
   void set_super_a(int n){
      super.a = n;
   }
   void display_super_a(){
      super.display_a();
   }
}

class SuperDemo {
   public static void main(String args[]){
      Superclass SuperObj = new Superclass();
      Subclass SubObj = new Subclass();
      SuperObj.a = 1;
      SubObj.a = 2;
      SubObj.set_super_a(3);
      SuperObj.display_a();
      SubObj.display_a();
      SubObj.display_super_a();
      System.out.println(SubObj.a);
   }
            }
```

Program tersebut akan menampilkan hasil berikut.

```
            a = 1
a = 2
a = 3
            2
```

1.3.16 Kata Kunci static

Kata kunci *static* dapat digunakan untuk anggota dari sebuah *class.* Kata kunci ini menyediakan *static* atau anggota *class* untuk diakses sama sebelum beberapa *instance* dari *class* dibuat.

Variabel *class* bersifat seperti variabel umum. Ini artinya bahwa variabel dapat diakses oleh semua *instance* dari *class*.

Method class mungkin dapat diambil tanpa membuat sebuah *object* dari *class* tersebut.

Bagaimanapun, mereka hanya dapat mengakses anggota *static* dari *class*. Ditambahkan juga, mereka tidak dapat menunjuk *this* dan *super.*

Kata kunci *static* dapat juga diaplikasikan pada blok. Ini dinamakan dengan blok *static*. Blok ini dieksekusi hanya sekali, ketika *class* diisi. Hal ini biasanya digunakan untuk menginisialisasi variabel *class*.

```
class Demo {
    static int a = 0;
    static void staticMethod(int i) {
        System.out.println(i);
    }
    static {     //blok static
        System.out.println("This is a static block.");
        a += 1;
    }
}

class StaticDemo {
   public static void main(String args[]) {
      System.out.println(Demo.a);
      Demo.staticMethod(5);
      Demo d = new Demo();
      System.out.println(d.a);
      d.staticMethod(0);
      Demo e = new Demo();
      System.out.println(e.a);
      d.a += 3;
      System.out.println(Demo.a+", " +d.a +", " +e.a);
   }
}
```

Keluaran dari *source code* ditunjukkan di bawah ini.

```
This is a static block.
1
5
1
0
1
4, 4, 4
```

1.3.17 Kata Kunci final

Kata kunci *final* dapat diaplikasikan pada variabel, *method* dan *class*. Untuk mengingat fungsi dari kata kunci, ingat bahwa itu secara mudah dibatasi apa yang kita dapat lakukan dengan variabel, *method* dan *class*.

Nilai dari variabel final dapat tidak lama diubah sesudah nilainya telah diatur. Sebagai contoh,

```
final int data = 10;
```

Pernyataan berikut akan menyebabkan terjadi *compilation error*:

```
data++;
```

Method final tidak dapat di-*override* dalam *class* anak.

```
final void myMethod() { //in a parent class
}
```

myMethod tidak dapat lama di-*override* dalam *class* anak.

class final tidak dapat diwariskan tidak seperti *class* yang biasanya.

```
final public class MyClass {
            }
```

Petunjuk Penulisan Program:
Perintah penulisan kata kunci final dan public memungkinkan bertukar tempat.

Pernyataan ini akan menyebabkan kesalahan *compilation* terjadi karena *MyClass* dapat tidak lama di-*extended*.

```
public WrongClass extends MyClass {
}
```

1.3.18 Inner Classes

Sebuah *inner class* secara mudah dideklarasikan dalam *class* lain.

```
class OuterClass {
   int data = 5;
   class InnerClass {
      int data2 = 10;
      void method() {
         System.out.println(data);
         System.out.println(data2);
      }
   }
   public static void main(String args[]) {
      OuterClass oc = new OuterClass();
      InnerClass ic = oc.new InnerClass();
      System.out.println(oc.data);
      System.out.println(ic.data2);
      ic.method();
   }
         }
```

Untuk mampu mengakses anggota dari *inner class*, kita butuh sebuah *instance* dari *inner class*. *Method-method* dari *inner class* dapat secara langsung mengakses anggota dari *outer class*.

1.4 Latihan

1.4.1 Tabel Perkalian

Tulis *program* yang mempunyai masukkan *size* dari *user* dan mencetak tabel perkalian dengan *size* yang ditetapkan.
Size untuk tabel perkalian : 5
Tabel perkalian dari size 5:

	1	2	3	4	5
1	1				
2	2	4			
3	3	6	9		
4	4	8	12	16	
5	5	10	15	20	25

1.4.2 Greatest Common Factor(GCF)

Tulis sebuah program yang mempunyai tiga *integer* dan menghitung nilai GCF dari tiga angka. GCF adalah angka terbesar yang secara rata dibagi ke semua angka yang diberikan.

Input 1: 25	Input 1: 1	Input 1: 9
Input 2: 15	Input 2: 2	Input 2: 27
Input 3: 35	Input 3: 3	Input 3: 12
GCF: 5	GCF: 1	GCF: 3

1.4.3 Shape

Buatlah *class* S*hape*. *Class* memiliki dua *field String*: *name* dan *size. class* mempunyai *method printShapeInfo*, dimana hanya mengeluarkan nilai *name* dan *field size* dari *object Shape.* Juga memiliki *method printShapeName* dan *printShapeSize,* dimana mencetak nama dan *size* dari *object*, berturut-turut.
Menggunakan pewarisan, buat *class Square* dengan *field* yang sama dan *method* seperti itu dari *class Shape*. *Class* ini mempunyai dua tambahan *field integer*: *length* dan *width*. *Method printShapeLength* dan *printShapeWidth* yang mencetak panjang dan lebar *object* yang juga termasuk dalam *class* ini. Anda juga harus meng-*override printShapeInfo* untuk mencetak keluaran *field* tambahan dalam *subclass* juga.

1.4.4 Binatang

Buatlah *interface Animal* yang mempunyai tiga *method*:*eat* dan *move*. Semua *method* ini tidak punya argumen atau nilai *return*. *Method* ini hanya mengeluarkan bagaimana *object Animal* makan dan bergerak. Sebagai contoh, seekor kelinci memakan wortel dan bergerak dengan melompat. Buat class *Fish* dan *Bear* yang menggunakan *interface Animal*. Terserah kepada Anda bagaimana menggunakan *method eat* dan *move.*

Bab 2
Aplikasi Berbasis Teks

2.1 Tujuan

Pembahasan kali ini akan menitikberatkan pada bahasan penggunaan *argument command-line.* Selebihnya, Anda akan mempelajari mengenai penggunaan *streams* untuk mendapatkan nilai *input* dari *user* pada saat *runtime,* sekaligus dalam proses manipulasi file.

Setelah menyelesaikan pembahasan ini, Anda diharapkan dapat :

1. Mendapatkan *input* dari *command-line*
2. Mengetahui cara untuk memanipulasi properties dari sistem
3. Membaca *standart input*
4. Membaca dan menulis *file*

2.2 Argument Command-Line dan System Properties

Seperti yang telah Anda ketahui pada pembahasan sebelumnya, *JAVA* mengijinkan *user* untuk memasukkan data dari *command-line*. Sebagai contoh, untuk meneruskan *argument* 1 dan 2 kepada *program Java* bernama *Calculate*, anda dapat menuliskan baris berikut pada *command prompt*

```
java Calculate 1 2
```

Pada contoh berikut ini, data 1 disimpan pada variabel *args[0]*, begitu pula dengan data 2 yang disimpan pada *args[1]*. Sehingga, tujuan dari deklarasi *String args[]* sebagai sebuah *parameter* pada *method* utama menjadi jelas.

Selain melewatkan *argument* menuju *method* utama, Anda juga dapat memanipulasi *system properties* dari *command-line*.

System properties hampir menyamai *environment variables*, namun tidak memiliki ketergantungan pada spesifikasi *platform* yang digunakan. Sebuah *property* secara sederhana berupa pemetaan antara *property name* dan *value* yang dimilikinya. Hal ini ditunjukkan pada *Java* dalam *class Properties*. *Class System* menyediakan sebuah *method* untuk menentukan *system properties* yang digunakan, *method getProperties* yang menghasilkan sebuah *object Properties. Class* yang sama juga menyediakan *method getProperty* yang memiliki dua buah bentuk.

public static String getProperty(String key)
Bentuk ini menghasilkan nilai *String* dari *System Properties* yang ditunjukkan oleh *key* yang ditentukan. Jika hasil menunjukkan nilai *null*, berarti tidak terdapat *property* dengan *key* yang ditentukan.
public static String getProperty(String key, String def)
Bentuk ini juga menghasilkan nilai *String* dari *System Properties* sesuai *key* yang ditentukan. Akan menghasilkan nilai *def*, sebuah nilai *default*, jika tidak terdapat *property* dengan *key* yang sesuai.

Tabel 1.1: getProperty() method dari class System

Kita tidak dapat cukup berhenti pada detail dari *system properties*, namun dilanjutkan dengan memanipulasi *system properties* yang digunakan. Jika Anda tertarik mempelajari lebih lanjut tentang *system properties*, Anda dapat menelusuri dokumentasi API yang disediakan.

Anda dapat menggunakan *argument* opsional –D pada perintah *Java* dalam *command-line* untuk menambahkan *property* baru.

```
java -D<name>=value
```

Sebagai contoh, untuk mengatur *system property* dengan nama *user. Home* bernilai *phillipines*, gunakan perintah berikut :

```
java -Duser.home=philippines
```

Untuk menampilkan daftar *system properties* yang tersedia pada sistem Anda, gunakan *method getProperties* seperti yang ditunjukkan sebagai berikut :

```
System.getProperties().list(System.out);
```

2.3 Membaca Standard Input

Dibandingkan dengan mendapatkan masukan *user* dari *command-line*, sebagian *user* lebih memilih untuk memasukkan data bilamana diminta oleh *program* pada saat eksekusi. Satu cara dalam melakukan hal ini adalah dengan menggunakan *stream*. Sebuah *stream* adalah abstraksi dari sebuah *file* atau sebuah perangkat yang mengijinkan beberapa *set item* untuk dibaca atau ditulisi. *Streams* terhubung dengan *physical devices* seperti *keyboards*, *consoles* dan *files*. Terdapat dua bentuk umum dari *streams*, *byte streams* dan *character streams*. *Byte streams* digunakan pada data biner, sedangkan *character streams* digunakan pada karakter *Unicode*. *System.in* dan *System.out* adalah dua contoh dari *byte streams* yang digunakan dalam *Java*. Contoh pertama mereferensikan pada *keyboard*, kemudian contoh kedua mereferensikan pada *console*.

Untuk membaca karakter dari *keyboard*, Anda dapat menggunakan *byte stream System.in* yang terdapat pada *object BufferedReader*. Baris berikut menunjukkan bagaimana untuk melakukan hal tersebut :

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Method read dari *object BufferedReader* selanjutnya digunakan untuk membaca nilai *input* dari perangkat *input*.

```
ch=(int)br.read(); //method read menghasilkan nilai integer
```

Cobalah contoh kode berikut :

```
import java.io.*;

class FavoriteCharacter {
	public static void main(String args[]) throws IOException {
		System.out.println("Hi, what's your favorite character?");
		char favChar;
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		favChar = (char) br.read();
		System.out.println(favChar + " is a good choice!");
	}
}
```

Jika Anda lebih memilih untuk membaca keseluruhan baris daripada membaca satu karakter tiap waktu, gunakan *method readLine* :

```
str = br.readLine();
```

Berikut ini sebuah *program* yang hampir menyerupai contoh sebelumnya, namun membaca keseluruhan *string*, bukan satu karakter.

```
import java.io.*;

class GreetUser {
	public static void main(String args[]) throws IOException {
		System.out.println("Hi, what's your name?");
		String name;
		BufferedReader br = new BufferedReader(new
		InputStreamReader(System.in));
		name = br.readLine();
		System.out.println("Nice to meet you, " + name + "! :)");
	}
}
```

Pada saat menggunakan *streams*, jangan lupa untuk meng*import package java.io* seperti yang ditunjukkan dibawah ini :

```
import java.io.*;
```

Satu hal lagi yang perlu untuk diingat, pembacaan dari *streams* dapat menyebabkan terjadinya *exception*. Jangan lupa untuk menangani *exception* tersebut menggunakan perintah *try-catch* atau dengan mengindikasikan *exception* pada *klausa throws* dalam *method*.

2.4 Menangani File

Pada beberapa kasus, masukan data disimpan pada sebuah *file*. Selanjutnya, terdapat beberapa cara jika Anda ingin menyimpan *output* dari *program* pada sebuah file. Pada sistem terkomputerisasi, data dari Siswa yang dapat digunakan sebagai *input* oleh sistem umumnya terseimpan pada sebuah *file* terpisah. Kemudian, salah satu kemungkinan *output* dari sistem adalah informasi tentang mata pelajaran yang diikuti oleh siswa. Sekali lagi, *output* dalam hal ini dapat disimpan dalam sebuah *file*. Seperti yang terlihat pada aplikasi, terdapat suatu kebutuhan untuk membaca dan menulis sebuah *file*. Anda akan mempelajari tentang *file input* dan *output* pada bagian ini.

2.4.1 Membaca sebuah File

Untuk membaca sebuah *file*, Anda dapat menggunakan *class FileInputStream*. Berikut ini adalah salah satu *constructor* dari *class* tersebut :

```
FileInputStream(String filename)
```

Constructor tersebut membuat sebuah koneksi terhadap *file* dimana nama dari *file* tersebut ditunjukkan sebagai sebuah *argument*. *Exception* berupa *FileNotFoundException* akan muncul jika *file* tidak ditemukan atau tidak dapat dibuka dan kemudian dibaca.

Setelah membuat sebuah *input stream*, Anda kemudian dapat menggunakannya untuk membaca sebuah *file* dengan menggunakan *method read*. *Method read* menghasilkan sebuah nilai *integer*, dan akan menunjukkan nilai 1 jika telah mencapai batas akhir *file*.

Berikut ini contohnya :

```
import java.io.*;

class ReadFile {
  public static void main(String args[]) throws IOException {
          System.out.println("What is the name of the file to read from?");
          String filename;
          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
          filename = br.readLine();
          System.out.println("Now reading from " + filename + "...");
          FileInputStream fis = null;
          try {
                  fis = new FileInputStream(filename);
          } catch (FileNotFoundException ex) {
                  System.out.println("File not found.");
          }
          try {
                  char data;
                  int temp;
                  do {
                          temp = fis.read();
                          data = (char) temp;
                  if (temp != -1) {
                          System.out.print(data);
                  }
               } while (temp != -1);
          } catch (IOException ex) {
                  System.out.println("Problem in reading from the file.");
          }
  }
}
```

2.4.2 Menulis sebuah file

Untuk menuliskan sebuah *file*, Anda dapat menggunakan *class FileOutputStream*. Berikut ini salah satu *constructor* yang dapat Anda gunakan.

```
FileOutputStream(String filename)
```

Constructor tersebut menyediakan jalur *output stream* terhadap sebuah file yang akan ditulisi. Sebuah *Exception* berupa *FileNotFoundException* akan muncul jika file yang dimaksud tidak dapat dibuka untuk ditulisi.

Jika *output stream* telah dibuat, Anda dapat menggunakannya untuk menulisi *file* yang dituju menggunakan *method write*. *Method* tersebut menggunakan penandaan sebagai berikut :

```
void write(int b)
```

Parameter *b* mereferensikan data yang akan dituliskan pada *file* sesuai dengan hasil *output stream*.

Program berikut menunjukkan contoh penulisan terhadap file :

```
import java.io.*;

class WriteFile {
       public static void main(String args[]) throws IOException {
              System.out.println("What is the name of the file to be written
              to?");
              String filename;
              BufferedReader br = new BufferedReader(new
                  InputStreamReader(System.in));
              filename = br.readLine();
              System.out.println("Enter data to write to " + filename +
                  "...");
              System.out.println("Type q$ to end.");
              FileOutputStream fos = null;
              try                                                              {

                     fos = new FileOutputStream(filename);
              } catch (FileNotFoundException ex) {
                     System.out.println("File cannot be opened for
                          writing.");
              }
              try {
                     boolean done = false;
                     int data;
              do {
                     data = br.read();
                     if ((char)data == 'q') {
                            data = br.read();
                            if ((char)data == '$') {
                                   done = true;
                            } else {
                                   fos.write('q');
                                   fos.write(data);
                            }
                     } else {
                            fos.write(data);
                     }
                 } while (!done);
              } catch (IOException ex) {
                     System.out.println("Problem in reading from the file.");
              }
       }
}
```

2.5 Latihan

2.5.1 Spasi menjadi Underscore (_)

Buatlah sebuah *program* yang memuat dua *String* sebagai *argument*, sumber dan nama *file* tujuan. Kemudian, baca *file* sumber dan tuliskan isi dari *file* tersebut terhadap *file* tujuan, seluruh spasi yang ada (‘ ‘) diubah menjadi *underscore* (‘ _ ‘).

Bab 3
GUI Event Handling

3.1 Tujuan

Pada modul ini, Anda akan belajar bagaimana mengendalikan *events triggered* ketika *user* berinteraksi dengan aplikasi GUI Anda. Setelah menyelesaikan modul ini, Anda akan dapat mengembangkan aplikasi GUI yang merespon interaksi *user*.

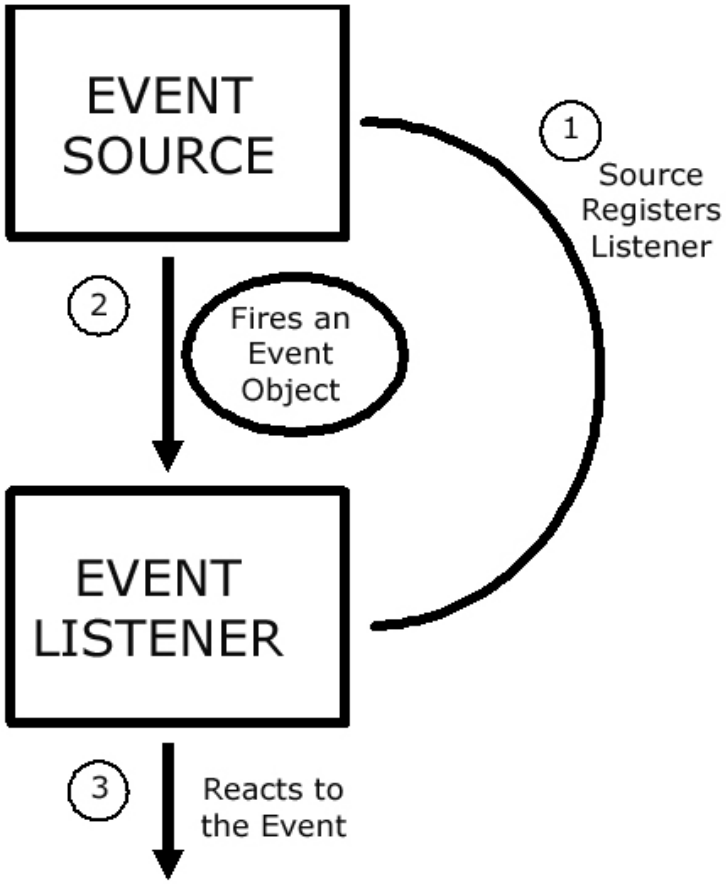
Setelah menyelesaikan modul ini, Anda diharapkan mampu:
1. Menerangkan komponen-komponen *delegation event model*
2. Mengerti bagaimana *delegation event model* bekerja
3. Menciptakan aplikasi GUI yang berinteraksi dengan *user*
4. Mendiskusikan manfaat dari *class-class adapter*
5. Mendiskusikan keuntungan-keuntungan dari menggunakan *inner* dan *anonymous class*

3.2 Delegation Event Model

Delegasi *event* model menguraikan bagaimana program Anda dapat merespon interaksi dari *user*. Untuk memahami model, mari kita pelajari pertama-tama dengan tiga komponen utamanya.

1. *Event Source*
   The event source mengacu pada komponen GUI yang men-*generate event*. Sebagai contoh, jika *user* menekan tombol, *event source* dalam hal ini adalah tombol.
2. *Event Listener/Handler*
   The event listener menerima berita dari *event-event* dan proses-proses interaksi *user*. Ketika tombol ditekan, *listener* akan mengendalikan dengan menampilkan sebuah informasi yang berguna untuk *user*.
3. *Event Object*
   Ketika sebuah *event* terjadi (misal, ketika *user* berinteraksi dengan komponen GUI), sebuah objek *event* diciptakan. Objek berisi semua informasi yang perlu tentang *event* yang telah terjadi. Informasi meliputi tipe dari *event* yang telah terjadi, seperti ketika *mouse* telah di-klik. Ada beberapa *class event* untuk kategori yang berbeda dari *user action*. Sebuah *event object* mempunyai tipe data tentang salah satu dari *class* ini.

Di bawah ini adalah *delegation event model*.



Gambar 8.1: Delegation Event Model

Pada awalnya, sebuah *listener* seharusnya diregistrasikan dengan sebuah *source* sehingga dapat menerima informasi tentang *event-event* yang terjadi pada *source* tersebut. Hanya *listeners* yang sudah teregistrasi yang dapat menerima pemberitahuan *event-event*. Ketika telah teregistrasi, sebuah *listener* hanya tinggal menunggu sampai *event* terjadi.

Ketika sesuatu terjadi dengan *event source*, sebuah *event object* akan menguraikan *event* yang diciptakan. *Event* kemudian ditembak oleh *source* pada *listeners* yang teregistrasi.

Saat *listener* menerima sebuah *event object* (pemberitahuan) dari *source*, dia akan bekerja. Menerjemahkan pemberitahuan dan memproses *event* yang terjadi.

3.2.1 Registrasi Listeners

Event source mendaftarkan sebuah *listener* melalui *method add<Type>Listener*.

```
void add<Type>Listener(<Type>Listener listenerObj)
```

<Type> tergantung pada tipe dari *event source*. Dapat berupa *Key*, *Mouse*, *Focus*, *Component*, *Action* dan lainnya.
Beberapa *listeners* dapat diregistrasi dengan satu *event source* untuk menerima pemberitahuan *event*.

Listener yang telah teregistrasi dapat juga tidak diregistrasikan lagi menggunakan *remove<Type>Listener* methods.

```
void remove<Type>Listener(<Type>Listener listenerObj)
```

3.3 Class-Class Event

Sebuah *event object* mempunyai sebuah *class event* sebagai tipe data acuannya. Akar dari hirarki *class event* adalah *class EventObject*, yang dapat ditemukan pada paket *java.util*. *Immediate subclass* dari *class EventObject* adalah *class AWTEvent*. *Class AWTEvent* didefinisikan pada paket *java.awt.* Itu merupakan akar dari semua *AWT-based events*. Berikut ini beberapa dari *class-class AWT event.*

Class Event	***Deskripsi***
ComponentEvent	*Extends AWTEvent*. Dijalankan ketika sebuah komponen dipindahkan, di-*resize*, dibuat *visible* atau *hidden*.
InputEvent	*Extends ComponentEvent*. Abstrak *root class event* untuk semua komponen-level *input class-class event*.
ActionEvent	*Extends AWTEvent*. Dijalankan ketika sebuah tombol ditekan, melakukan *double-klik* daftar *item*, atau memilih sebuah *menu*.
ItemEvent	*Extends AWTEvent*. Dijalankan ketika sebuah *item* dipilih atau di-*deselect* oleh *user*, seperti sebuah *list* atau *checkbox*.
KeyEvent	*Extends InputEvent*. Dijalankan ketika sebuah *key* ditekan, dilepas atau diketikkan.
MouseEvent	*Extends InputEvent*. Dijalankan ketika sebuah tombol *mouse* ditekan, dilepas, atau di-klik (tekan dan lepas), atau ketika sebuah *kursor mouse* masuk atau keluar dari bagian *visible* dari komponen.
TextEvent	*Extends AWTEvent*. Dijalankan ketika nilai dari *text field* atau *text area* dirubah.
WindowEvent	*Extends ComponentEvent*. Dijalankan sebuah objek *Window* dibuka, ditutup, diaktifkan, nonaktifkan, *iconified*, *deiconified*, atau ketika *focus* ditransfer kedalam atau keluar *window*.

Tabel 1.2: Class-Class Event

Catatan, bahwa semua subclass-subclass *AWTEvent* mengikuti konvensi nama ini:

```
<Type>Event
```

3.4 Event Listeners

Event listeners adalah *class* yang mengimplementasikan *interfaces <Type>Listener*. Tabel di bawah menunjukkan beberapa *listener interfaces* yang biasanya digunakan.

Event Listeners	***Deskripsi***
ActionListener	Bereaksi atas perubahan *mouse* atau *keyboard*.
MouseListener	Bereaksi atas pergerakan *mouse*.
MouseMotionListener	*Interface MouseMotionListener* mendukung *MouseListener*. Menyediakan *method-method* yang akan memantau pergerakan *mouse*,seperti *drag* dan pemindahan *mouse*.
WindowListener	Bereaksi atas perubahan *window*.

Tabel 1.3: Event Listeners

3.4.1 Method ActionListener

Interface ActionListener hanya terdiri dari satu *method*.

ActionListener Method
public void actionPerformed(ActionEvent e)
Mengendalikan *ActionEvent e* yang terjadi.

Tabel 1.3.1: Method ActionListener

3.4.2 Method MouseListener

Di bawah ini adalah *method-method MouseListener* yang seharusnya digunakan dalam penerapan *class*.

MouseListener Methods
public void mouseClicked(MouseEvent e)
Dipanggil pada saat tombol *mouse* di click (seperti tekan dan lepas).
public void mouseEntered(MouseEvent e)
Dipanggil pada saat kursor *mouse* memasuki area komponen.
public void mouseExited(MouseEvent e)
Dipanggil pada saat kursor *mouse* meninggalkan area komponen.
public void mousePressed(MouseEvent e)
Dipanggil pada saat tombol *mouse* ditekan di atas komponen
public void mouseReleased(MouseEvent e)
Dipanggil pada saat tombol *mouse* dilepas di atas komponen

Tabel 1.3.2: Method-Method MouseListener

3.4.3 Method-Method MouseMotionListener

MouseMotionListener mempunyai dua *method* untuk diimplementasikan.

MouseListener Methods
public void mouseDragged(MouseEvent e)
Digunakan untuk memantau pergerakan *mouse* yang melintasi objek pada saat tombol *mouse* ditekan. Tindakan ini persis sama dengan tindakan pada saat memindahkan sebuah *window*.
public void mouseMoved(MouseEvent e)
Digunakan untuk memantau pergerakan *mouse* pada saat *mouse* melintasi area suatu objek. Pada saat ini tidak ada *mouse* yang ditekan, hanya memindahkan *pointer mouse* melalui objek.

Tabel 1.3.3: The MouseMotionListener methods

3.4.4 Method-Method WindowListener

Di bawah ini *method-method* dari *interface WindowListener*.

WindowListener Methods
public void windowOpened(WindowEvent e)
Dipanggil pada saat objek *window* dibuka (pertama kali *window* dibuat tampil).
public void windowClosing(WindowEvent e)
Dipanggil pada saat *user* mencoba untuk menutup objek *Window* dari menu sistem objek.
public void windowClosed(WindowEvent e)
Dipanggil pada saat objek *Window* ditutup setelah memanggil penempatan (misal, *release* dari *resource-resource* yang digunakan oleh *source*) pada objek.
public void windowActivated(WindowEvent e)
Dilibatkan ketika objek *Window* adalah *window* yang aktif (*window* masih dipakai).
public void windowDeactivated(WindowEvent e)
Dilibatkan ketika objek *Window* tidak lagi merupakan *window* yang aktif.
public void windowIconified(WindowEvent e)
Dipanggil ketika objek *Window* di-*minimize*.
public void windowDeiconified(WindowEvent e)
Dipanggil ketika objek *Window* kembali setelah di-minimize ke keadaan normal.

Tabel 1.3.4: Method-Method WindowListener

3.4.5 Petunjuk untuk Menciptakan Aplikasi Handling GUI Events

Berikut ini langkah-langkah yang Anda butuhkan untuk mengingat ketika ingin membuat aplikasi GUI dengan *event handling*.

1. Buatlah sebuah *class* yang menguraikan dan menampilkan tampilan dari aplikasi GUI Anda.
2. Buatlah sebuah *class* yang menerapkan *interface listener* yang sesuai. *Class* ini boleh mengacu pada *class* yang sama seperti pada langkah awal.
3. Dalam menerapkan *class*, gunakan semua *methods* dengan *interface listener* yang sesuai. Uraikan masing-masing *method* bagaimana Anda ingin mengendalikan *event-event*. Anda dapat memberikan implementasi kosong untuk *method* yang tidak ingin Anda gunakan.
4. Daftarkan objek *listener*, *instansiasi* dari *class listener* pada langkah 2, dengan *source component* menggunakan *method add<Type>Listener*.

3.4.6 Contoh Mouse Events

```
import java.awt.*;
import java.awt.event.*;

public class MouseEventsDemo extends Frame implements
                        MouseListener, MouseMotionListener {
   TextField tf;
   public MouseEventsDemo(String title){
      super(title);
      tf = new TextField(60);
      addMouseListener(this);
   }
   public void launchFrame() {
      /* Menambah komponen ke frame */
      add(tf, BorderLayout.SOUTH);
      setSize(300,300);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent me) {
      String msg = "Mouse clicked.";
      tf.setText(msg);
   }
   public void mouseEntered(MouseEvent me) {
      String msg = "Mouse entered component.";
      tf.setText(msg);
   }
   public void mouseExited(MouseEvent me) {
      String msg = "Mouse exited component.";
      tf.setText(msg);
   }
   public void mousePressed(MouseEvent me) {
      String msg = "Mouse pressed.";
    tf.setText(msg);
   }
```

```
    public void mouseReleased(MouseEvent me) {
       String msg = "Mouse released.";
       tf.setText(msg);
    }
    public void mouseDragged(MouseEvent me) {
       String msg = "Mouse dragged at " + me.getX() + "," +
                                                 me.getY();
       tf.setText(msg);
    }
    public void mouseMoved(MouseEvent me) {
       String msg = "Mouse moved at " + me.getX() + "," +
                                                 me.getY();
       tf.setText(msg);
    }
    public static void main(String args[]) {
       MouseEventsDemo med = new MouseEventsDemo("Mouse Events
                                                    Demo");
       med.launchFrame();
     }
}
```

3.4.7 Contoh Menutup Window

```
import java.awt.*;
import java.awt.event.*;


class CloseFrame extends Frame implements WindowListener {
   Label label;

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(this);
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public void windowActivated(WindowEvent e) {
   }
   public void windowClosed(WindowEvent e) {
   }
   public void windowClosing(WindowEvent e) {
      setVisible(false);
      System.exit(0);
   }
   public void windowDeactivated(WindowEvent e) {
   }
   public void windowDeiconified(WindowEvent e) {
   }
```

```
   public void windowIconified(WindowEvent e) {
   }
   public void windowOpened(WindowEvent e) {
   }

   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}
```

3.5 Adapter Classes

Menerapkan semua *method* dari *interface* yang semuanya akan membutuhkan banyak pekerjaan. Di satu sisi, Anda terkadang lebih sering tertarik menerapkan hanya beberapa *method* dari *interface* saja. Untungnya, *Java* menyediakan untuk kita *class- class adapter* yang menerapkan semua *method* dari masing-masing *listener interface* dengan lebih dari satu *method*. Implementasi dari *method-method* semuanya adalah kosong.

3.5.1 Close Window Example

```
import java.awt.*;
import java.awt.event.*;

class CloseFrame extends Frame{
   Label label;
   CFListener w = new CFListener(this);

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(w);
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}

class CFListener extends WindowAdapter{
    CloseFrame ref;
    CFListener( CloseFrame ref ){
       this.ref = ref;
       }
    public void windowClosing(WindowEvent e) {
         ref.dispose();
         System.exit(1);
   }
}
```

3.6 Inner Class dan Anonymous Inner Class

Bagian ini memberi Anda tinjauan ulang atas konsep yang sudah Anda pelajari di kursus pemrograman pertama. *Inner class* dan *anonymous inner class* sangatlah bermanfaat untuk GUI *event handling*.

3.6.1 Inner Class

Inner class, seperti namanya, adalah sebuah *class* yang dideklarasikan di dalam *class* lain. Kegunaan *inner classes* akan dapat membantu Anda menyederhanakan program, terutama dalam *event handling* seperti yang ditunjukkan pada contoh.

3.6.2 Contoh Menutup Window

```
import java.awt.*;
import java.awt.event.*;

class CloseFrame extends Frame{
       Label label;

       CloseFrame(String title) {
            super(title);
            label = new Label("Close the frame.");
            this.addWindowListener(new CFListener());
       }

       void launchFrame() {
            setSize(300,300);
            setVisible(true);
       }

       class CFListener extends WindowAdapter {
            public void windowClosing(WindowEvent e) {
                  dispose();
                  System.exit(1);
            }
       }

       public static void main(String args[]) {
            CloseFrame cf = new CloseFrame("Close Window
                                                  Example");
            cf.launchFrame();
       }
}
```

3.6.3 Anonymous Inner Class

Anonymous inner class adalah *inner class* yang tanpa nama. Kegunaan dari *anonymous inner class* akan menyederhanakan kode-kode Anda lebih lanjut. Di bawah ini merupakan modifikasi dari contoh bagian sebelumnya.

3.6.4 Contoh Menutup Window

```
import java.awt.*;
import java.awt.event.*;

class CloseFrame extends Frame{
   Label label;

   CloseFrame(String title) {
      super(title);
      label = new Label("Close the frame.");
      this.addWindowListener(new WindowAdapter() {
             public void windowClosing(WindowEvent e){
                  dispose();
                  System.exit(1);
             }
        });
   }

   void launchFrame() {
      setSize(300,300);
      setVisible(true);
   }

   public static void main(String args[]) {
      CloseFrame cf = new CloseFrame("Close Window Example");
      cf.launchFrame();
    }
}
```

3.7 Latihan

3.7.1 Tic-Tac-Toe

Extend program papan *Tic-Tac-Toe* yang telah Anda kembangkan sebelumnya dan tambahkan *event handlers* ke kode tersebut untuk membuat program berfungsi penuh. Permainan *Tic-Tac-Toe* dimainkan dengan dua pemain. Pemain mengambil giliran mengubah. Setiap giliran, pemain dapat memilih kotak pada papan. Ketika kotak dipilih, kotak ditandai oleh simbol pemain (O dan X biasanya digunakan sebagai simbol). Pemain yang sukses menaklukkan 3 kotak membentuk garis horisontal, vertikal, atau diagonal, memenangkan permainan. Permainan akan berakhir ketika pemain menang atau ketika semua kotak telah terisi.

Gambar 8.2 : Program Tic-Tac-Toe

Bab 4
Applet

4.1 Tujuan

Applets adalah satu dari fitur yang paling menarik dalam *java*. Applet merupakan program yang Anda jalankan melalui *web browser*. Anda akan belajar tentang membentuk applet pada pembelajaran ini.

Setelah melengkapi pembelajaran ini, Anda harus dapat :

1. mendefinisikan apa yang dimaksud dengan applet
2. Membentuk applet anda sendiri
3. Mengetahui siklus yang terdapat pada applet
   - *init*
   - *start*
   - *stop*
   - *destroy*
4. menggunakan *methods* applet yang lain
   - *paint*
   - *showStatus*
   - *Methods* untuk memainkan sebuah *audio clip*
5. Memahami *html tag* pada applet

4.2 Membentuk Applets

Sebuah applet adalah tipe yang spesial dari program *java* yang dieksekusi melalui internet. Secara khusus berjalan pada suatu *web browser* seperti *Netscape Navigator*, *Mozilla*, atau *Microsoft Internet Explorer*. Bagaimanapun, jika dibandingkan dengan aplikasi *Java* yang normal, tidak diijinkan mengakses applet pada komputer yang mana dijalankan untuk alasan keamanan. Applet ini cukup terbatas jika dibandingkan dengan aplikasi *java*.

Pada *module* ini, Anda akan mempelajari tentang membuat applet menggunakan AWT.

4.2.1 Hello World Applet

Class Applet adalah sebuah *subclass* dari *class Panel* yang didefinisikan dalam AWT. Jalan terbaik untuk memahami bagaimana untuk membentuk Applet adalah dengan contoh. Jadi, berikut ini adalah contoh applet sederhana yang menampilkan "*Hello world!*".

```
import java.awt.*;
```

```
import java.applet.*;
/* masukkan bagian ini dalam kode html
   <applet code="AppletDemo" width=300 height=100>
   </applet>
*/

public class AppletDemo extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 80, 25);
   }
}
```

Setelah proses kompilasi, usahakan jalankan contoh ini menggunakan baris perintah *java*. Apa yang terjadi? Ingat bahwa applet adalah aplikasi *java* yang spesial. Mereka tidak dieksekusi menggunakan perintah *java*. Bahkan applet berjalan pada *web browser* atau menggunakan *applet viewer*. Untuk membuka applet melalui sebuah *web browser*, secara sederhana buka dokumen HTML dimana applet terintegrasi kedalamnya menggunakan *applet HTML tag* (Perintah mengeluarkan kode pada contoh *Hello World*).

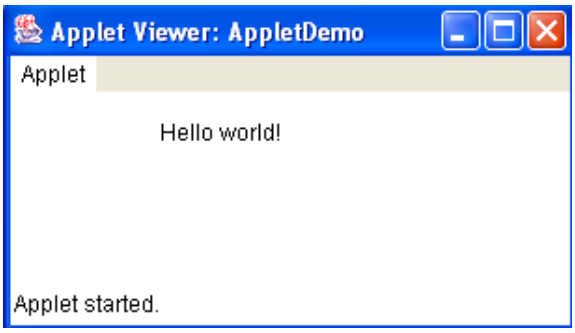
Cara lain untuk menjalankan sebuah applet adalah melalui perintah *appletviewer*. Untuk memudahkan ikuti *syntax* berikut ini:

```
appletviewer <java filename>
```

Sebagai contoh, untuk menjalankan contoh applet yang dijalankan , menggunakan:

```
appletviewer AppletDemo.java
```

Tag HTML pada contoh yang diberikan mengindikasikan bahwa sebuah applet dibuat dengan lebar 300 pixel dan tinggi 100 pixel. kemudian, *method drawString* menggambar string "*Hello world!*" pada posisi pixel (80,25) menghitung kebawah dari bagian kanan.



Gambar 1.1.1: contoh Applet

Ketika membuat sebuah applet, dibutuhkan suatu *extend class Applet*. Sebagaimana yang disebutkan sebelumnya, *class*nya dapat ditemukan dalam *java.applet package*. Oleh karena itu, mengimport *the java.applet package* merupakan suatu keharusan. Juga, telah disebutkan sebelumnya bahwa *class Applet* adalah *subclass* dari *class Panel*. Hal ini mengimplikasikan bahwa beberapa *methods* dari *class applet* ditemukan dalam *class Panel*.Untuk mengakses *methods* atau *fields* dalam *class Panel* atau *class-class* induk, diperlukan suatu aksi untuk *import package java.awt* .

4.3 Method-Method Applet

Bagian ini membahas *methods applet* yang akan Anda temukan manfaatnya.

4.3.1 Siklus Applet (The Applet Life Cycle)

Bahkan untuk memulai eksekusi pada *main method* seperti dalam aplikasi khas *Java*, *browser* atau *applet viewer* berhubungan dengan applet melalui *method-method* berikut :

1. *init()*
   init adalah *method* yang dipanggil pertama kali. Yang sebenarnya berisi permintaan pertama ketika applet di *load*.
2. *start()*
   Setelah meminta *method init*, mulai dengan *method* yang dipangil selanjutnya. *method* ini meminta dokumen HTML yang ditampilkan applet setiap waktu. Eksekusi ringkasan dengan *method* ini dilakukan ketika applet ditampilkan kembali.
3. *stop()*
   Ketika *web browser* meninggalkan dokumen HTML applet, *method* ini dipanggil untuk menginformasikan applet bahwa dia harus menghentikan proses eksekusinya.
4. *destroy()*
   method ini dipanggil ketika applet perlu dihapus dari kelangkapan *memory*. *methode stop* selalu dipanggil sebelum *method* ini diminta untuk dijalankan.

Ketika membuat applet, sedikitnya beberapa dari *method* ini telah menolaknya. contoh applet berikut menolak *method* berikut.

```
import java.applet.*;
import java.awt.*;
/*
   <applet code="LifeCycleDemo" width=300 height=100>
   </applet>
*/

class LifeCycleDemo extends Applet {
    String msg ="";
    public void init() {
        msg += "initializing... ";
        repaint();
    }
    public void start() {
        msg += "starting... ";
        repaint();
    }
    public void stop() {
        msg += "stopping... ";
        repaint();
    }
    public void destroy() {
        msg += "preparing for unloading...";
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString(msg, 15, 15);
    }
}
```

Contoh dokumen html berikut tergabung dalam *applet LifeCycleDemo* .

```
<HTML>
<TITLE>Life Cycle Demo</TITLE>
   <applet code="LifeCycleDemo" width=300 height=100>
   </applet>
</HTML>
```

4.3.2 Method paint

Method lain yang tidak kalah penting adalah *method paint*, yang mana *class Applet* menurunkannya dari *class* induknya yaitu *class Component,* Yang meminta *output applet* setiap waktu yang diperlukan untuk dapat digambar kembali. Sebagai contoh dari setiap *instance* adalah ketika sebuah *applet* tersembunyi oleh *window* lain dapat dibuat terlihat lagi. *Method* ini selalu menolak ketika anda ingin membuat bagaimana applet yang Anda buat harus terlihat seperti yag anda inginkan. Pada contoh *Hello World*, applet memiliki *string* "*Hello world!*"pada *background* setelah menolak *method paint*.

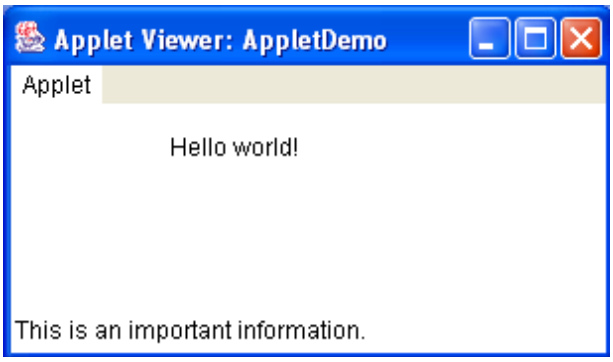
4.3.3 ShowStatus Method

Applet memiliki *window status*, dimana memberi informasi kepada Anda tentang apa yang sebenarnya dilakukan applet. Jika anda ingin memberi *output* ke *window status*, secara sederhana memanggil *method showStatus.*
Contoh berikut ini sama seperti contoh *Hello World* tapi dengan pernyataan tambahan yang memodifikasi isi dari *window status*.

```
import java.awt.*;
import java.applet.*;
/*
   <applet code="AppletDemo" width=300 height=100>
   </applet>
*/

public class AppletDemo extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 80, 25);
      showStatus("This is an important information.");
   }
}
```

Berikut ini adalah contoh hasil *output*nya:



Gambar 1.2.3: contoh showStatus()

4.3.4 Memainkan Audio Clips

Applets juga menyediakan layanan melalui adanya suatu *method* yang memungkinkan Anda untuk memainkan *file audio*. Memainkan *audio clips* dalam sebuah applet melibatkan dua langkah dasar :

1. Dapatkan *audio clip* menggunakan *method getAudioClip*.
2. Untuk memainkan *audio clip*, menggunakan *method play* atau *loop* pada *object audio clip*. *play* memungkinkan Anda untuk memainkan *audio* satu kali mengingat *loop* berulang pada *audio clip* dan berhenti hanya ketika *method stop* dipanggil.

Contoh berikutnya memainkan *file audio* secara terus-menerus hingga *method stop* applet dipanggil.

```
import java.awt.*;
import java.applet.*;
/*
   <applet code="AudioApplet" width=300 height=100>
   </applet>
*/

public class AudioApplet extends Applet {
   AudioClip ac;
   public void init() {
     try {
        /*audio clip tersimpan dalam direktori yang sama
seperti kode javanya*/
        /* spaceMusic telah terdownload dari java.sun.com */
        ac = getAudioClip(getCodeBase(), "spaceMusic.au");
        ac.loop();
      } catch (Exception e) {
        System.out.println(e);
      }
   }
   public void stop() {
      ac.stop();
   }
   public void paint(Graphics g) {
      g.drawString("Playing space music!", 80, 25);
   }
}
```

4.4 Applet HTML Tags

Dalam contoh terdahulu, Anda sudah melihat bagaimana applet HTML tags digunakan dalam dokumen HTML atau *source code java*. Sekarang, Anda akan dikenalkan pada versi applet HTML tags yang lebih lengkap.

```
<APPLET
   [CODEBASE = codebaseURL]
   CODE = appletFile
   [ATL = alternateText]
   [NAME = appletInstanceName]
   WIDTH = widthInPixels HEIGHT = heightInPixels
   [ALIGN = alignment]
   [VSPACE = vspaceInPixels] [HSPACE = hspaceInPixels]
>
[<PARAM NAME = parameterName1 VALUE = parameterValue1>]
[<PARAM NAME = parameterName2 VALUE = parameterValue2>]
...
[<PARAM NAME = parameterNamen VALUE = parameterValuen>]
[HTML that will be dsiplayed in the absence of Java]
</APPLET>
```

Kata kunci Applet HTML Tag
CODEBASE
Directory dimana *class applet* diletakkan. Untuk dokumen HTML, *directory* URL sesuai dengan *setting* awalnya/defaultnya.
CODE
Nama *file* yang berisi kode applet applet. Dengan atau tanpa nama ekstensi *.java* atau *.class .*
ALT
Text ditampilkan jika *browser* mengerti *applet tags* tapi applet tidak dapat dieksekusi secara langsung. Mungkin terjadi jika *Javanya disabled*.
NAME
Nama dari applet. Digunakan untuk memungkinkan applets yang lain untuk berkomunikasi dengan applet ini dengan menunjukkan suatu applet berdasarkan namanya.
WIDTH, HEIGHT
Width dan *height* dari *window* applet. Dinyatakan dalam *pixel*.
ALIGN

Kata kunci Applet HTML Tag
Alignment atau pengaturan posisi dari applet. satu diantara "left", "right", "top", "bottom", "middle", "baseline", "texttop", "absmiddle", atau "absbottom". Peletakan posisi secara Default tergantung pada lingkungan. "top" – posisi atas dari applet diratakan dengan item tertinggi dalam baris yang ada. "bottom", baseline – posisi bawah dari applet diratakan dengan bawah dari content lain dalam baris yang sama. "middle" – tengah dari applet diratakan dengan bawah dari content yang lain dalam baris yang sama. "texttop" – posisi atas dari applet diratakan dengan posisi atas dari applet diratakan dengan posisi tertinggi dari posisi atas pada baris yang sama. "absmiddle" – tengah dari applet diratakan dengan vertical middle dari content lain pada baris yang sama. "absbottom" – posisi bawah dari applet diratakan dengan posisi bawah dari content lain dalam baris yang sama.
VSPACE, HSPACE
Spasi diatas dan dibawah (VSPACE) dan pada sisi (HSPACE) dari applet..
PARAM NAME, VALUE
Untuk mengelompokkan *parameter* yang dapat menampilkan applet; applet dapat meminta *method getParameter(String paramName).*

Table 1.3: Applet HTML Tags

Contoh dibawah ini mendemokan bagaimana untuk mengakses parameter tertentu pada *HTML tag*.

```
import java.awt.*;
import java.applet.*;
/*
   <applet code="ParamDemo" width=300 height=100>
   <param name="myParam" value="Hello world!">
   </applet>
*/

public class ParamDemo extends Applet {
   public void paint(Graphics g) {
      g.drawString(getParameter("myParam"), 80, 25);
   }
}
```

Output dari program ini hanya sama seperti *applet Hello World*.

4.5 Latihan

4.5.1 Tic-Tac-Toe Applet satu-player

Buat *game* satu-*player Tic-Tac-Toe*. *User* memainkannya melawan komputer. Untuk setiap giliran, pemain harus menggeser kotak dari papan. Sekali sebuah kotak terpilih, kotak tersebut ditandai oleh symbol pemain (O dan X yang selalu digunakan sebagai symbol). pemain yang berhasil mengatasi 3 kotak membentuk baris horizontal, vertical atau diagonal memenangkan *game* ini. *Game* ini berakhir ketika pemain menang atau ketika semua kotak sudah berhasil dibentuk. Desain dan gerakan komputer seakan-akan *user* akan memengkan pertandingan melawan komputer.

Bab 5
Abstract Windowing Toolkit dan Swing

5.1 Tujuan

Tanpa mempelajari tentang *grapichal user interface* (GUI) API, Anda masih tetap bisa membuat suatu program. Tetapi, program Anda akan kelihatan tidak menarik dan tidak nyaman digunakan bagi para *user*. Memiliki GUI yang baik dapat memberi efek pada penggunaan aplikasi. *Java* menyediakan banyak *tool* seperti *Abstract Windowing Toolkit* dan *Swing* untuk mengembangkan aplikasi GUI yang interaktif.

Setelah menyelesaikan bab ini, Anda diharapkan untuk :

1. Memahami persamaan dan perbedaan antara *AWT* dan *Swing*
2. Perbedaan antara komponen dan kontainer.
3. Mendesain aplikasi GUI menggunakan AWT.
4. Mendesain aplikasi GUI menggunakan Swing.
5. Menjelaskan tentang *flow layout, border layout,* dan *grid layout* dalam komponen GUI
6. Membuat tampilan yang komplek dalam mendesain aplikasi GUI.

5.2 Abstract Windowing Toolkit (AWT) vs. Swing

The Java Foundation Class (JFC), merupakan bagian penting dari *Java SDK*, yang termasuk dalam koleksi dari API dimana dapat mempermudah pengembangan aplikasi *JAVA GUI*. JFC termasuk diantara 5 bagian utama dari API yaitu *AWT* dan *Swing*. Tiga bagian yang lainnya dari API adalah *Java2D*, *Accessibility,* dan *Drag* dan *Drop*. Semua itu membantu pengembang dalam mendesain dan mengimplementasikan aplikasi visual yang lebih baik.

AWT dan *Swing* menyediakan komponen GUI yang dapat digunakan dalam membuat aplikasi *Java* dan *applet*. Anda akan mempelajari *applet* pada bab berikutnya. Tidak seperti beberapa komponen AWT yang menggunakan *native code,* keseluruhan *Swing* ditulis menggunakan bahasa pemrograman *Java*. *Swing* menyediakan implementasi *platform-independent* dimana aplikasi yang dikembangkan dengan *platform* yang berbeda dapat memiliki tampilan yang sama. Begitu juga dengan AWT menjamin tampilan *look and feel* pada aplikasi yang dijalankan pada dua mesin yang berbeda menjadi terlihat sama. *Swing API* dibangun dari beberapa API yang mengimplementasikan beberapa jenis bagian dari AWT. Kesimpulannya, komponen AWT dapat digunakan dengan komponen *Swing*.

5.3 Komponen GUI pada AWT

5.3.1 Window Classes Fundamental

Dalam mengembangkan aplikasi GUI, komponen GUI seperti tombol atau *textfield* diletakkan di dalam kontainer. Berikut ini adalah daftar dari beberapa kelas penting pada kontainer yang telah disediakan oleh AWT.

AWT Class	*Description*
Komponen	*Abstract Class* untuk objek yang dapat ditampilkan pada *console* dan berinteraksi dengan *user*. Bagian utama dari semua kelas AWT.
Kontainer	*Abstract Subclass* dari *Component Class*. Sebuah komponen yang dapat menampung komponen yang lainnya.
Panel	Turunan dari *Container Class*. Sebuah *frame* atau *window* tanpa titlebar, menubar tidak termasuk *border*. Superclass dari applet class.
Window	Turunan dari *Container class*. *Top level window*, dimana berarti tidak bisa dimasukkan dalam objek yang lainnya.Tidak memiliki *border* dan menubar.

Frame	Turunan dari *window class*. *Window* dengan judul, menubar, *border* dan pengatur ukuran di pojok. Memiliki empat konstruktor , dua diantaranya memiliki penulisan seperti dibawah ini :
	`Frame()`
	`Frame(String title)`

Tabel 1.2.1: Kelas kontainer AWT

Untuk mengatur ukuran *window*, menggunakan metode *setSize*.

```
void setSize(int width, int height)
```

mengubah ukuran komponen ini dengan *width* dan *height* sebagai parameter.

```
void setSize(Dimension d)
```

mengubah ukuran dengan *d.width* dan *d.height* berdasar pada spesifikasi *Dimension d.*

Default dari *window* adalah *not visible* atau tak tampak hingga Anda mengatur *visibility* menjadi *true.* Inilah *syntax* untuk metode *setVisible.*

```
void setVisible(boolean b)
```

Dalam mendesain aplikasi GUI, Object *Frame* selalu digunakan. Dibawah ini adalah contoh bagaimana membuat sebuah aplikasi.

```
import java.awt.*;

public class SampleFrame extends Frame {
   public static void main(String args[]) {
      SampleFrame sf = new SampleFrame();
      sf.setSize(100, 100);   //Coba hilangkan baris ini
      sf.setVisible(true);    //Coba hilangkan baris ini
   }
}
```

perhatikan bahwa tombol tutup pada *frame* tidak akan bekerja karena tidak ada mekanisme *event handling* yang ditambahkan di dalam aplikasi. Anda akan belajar tentang *event handling* pada modul selanjutnya.

5.3.2 Grafik

Beberapa metode grafik ditemukan dalam *Graphic* class. Dibawah ini adalah daftar dari beberapa metode.

drawLine()	drawPolyline()	setColor()
fillRect()	drawPolygon()	getFont()
drawRect()	fillPolygon()	setFont()
clearRect()	getColor()	drawString()

Tabel 1.2.2a: Beberapa metode dari kelas Graphics

Hubungan dari kelas ini adalah *Color* class, dimana memiliki tiga konstruktor.

Constructor Format	***Description***
Color(int r, int g, int b)	Nilai integer 0 - 255.
Color(float r, float g, float b)	Nilai float 0.0 - 1.0.
Color(int rgbValue)	Panjang nilai : 0 ke 2^{24}-1 (hitam ke putih). Red: bits 16-23 Green: bits 8-15 Blue: bits 0-7

Dibawah ini adalah contoh program yang menggunakan beberapa metode di dalam *Graphic class*.

```
import java.awt.*;

public class GraphicPanel extends Panel {
   GraphicPanel() {
      setBackground(Color.black);  //constant in Color class
   }
   public void paint(Graphics g) {
      g.setColor(new Color(0,255,0));    //green
      g.setFont(new Font("Helvetica",Font.PLAIN,16));
      g.drawString("Hello GUI World!", 30, 100);
      g.setColor(new Color(1.0f,0,0));   //red
      g.fillRect(30, 100, 150, 10);
   }
   public static void main(String args[]) {
      Frame f = new Frame("Testing Graphics Panel");
```

```
        GraphicPanel gp = new GraphicPanel();
        f.add(gp);
        f.setSize(600, 300);
        f.setVisible(true);
    }
}
```

Agar *panel* dapat terlihat atau *visible,* dia harus diletakkan didalam *window* yang dapat terlihat seperti sebuah *frame*.

5.3.3 Beberapa komponen AWT

Berikut ini adalah daftar dari kontrol AWT. Kontrol adalah komponen seperti tombol atau *textfield* yang mengijinkan *user* untuk berinteraksi dengan aplikasi GUI. Berikut ini semua subkelas dari *Components class*.

Label	Button	Choice
TextField	Checkbox	List
TextArea	CheckboxGroup	Scrollbar

Tabel 1.2.3: Komponen AWT

Berikut adalah aplikasi membuat sebuah *frame* dengan kontrol yang telah dimasukkan di dalamnya.

```
import java.awt.*;

class FrameWControls extends Frame {
   public static void main(String args[]) {
      FrameWControls fwc = new  FrameWControls();
      fwc.setLayout(new FlowLayout());  //more on this later
      fwc.setSize(600, 600);
      fwc.add(new Button("Test Me!"));
      fwc.add(new Label("Labe"));
      fwc.add(new TextField());
      CheckboxGroup cbg = new CheckboxGroup();
      fwc.add(new Checkbox("chk1", cbg, true));
      fwc.add(new Checkbox("chk2", cbg, false));
      fwc.add(new Checkbox("chk3", cbg, false));
      List list = new List(3, false);
      list.add("MTV");
      list.add("V");
      fwc.add(list);
      Choice chooser = new Choice();
      chooser.add("Avril");
      chooser.add("Monica");
      chooser.add("Britney");
      fwc.add(chooser);
      fwc.add(new Scrollbar());
      fwc.setVisible(true);
   }
```

```
}
```

5.4 Layout Manager

Posisi dan ukuran suatu komponen ditentukan oleh *layout manager*. *Layout manager* mengatur tampilan dari komponen di dalam kontainer. Berikut ini beberapa *layout manager* yang terdapat di dalam *Java*.

1.FlowLayout
2.BorderLayout
3.GridLayout
4.GridBagLayout
5.CardLayout

Layout manager dapat diatur menggunakan metode *setLayout* dari *Container class.* Metode ini dapat ditulis sebagai berikut.

```
void setLayout(LayoutManager mgr)
```

Jika Anda memilih untuk tidak menggunakan *layout manager*, Anda dapat mengisi *null* sebagai argumen untuk metode ini. Tetapi selanjutnya, Anda akan mengatur posisi elemen secara manual dengan menggunakan metode *setBounds* dari *Components class*.

```
public void setBounds(int x, int y, int width, int height)
```

Metode ini mengatur posisi berdasarkan pada argumen x dan y, dan ukuran berdasarkan argumen *width* dan *height.* Hal ini akan cukup menyulitkan dan membosankan untuk aplikasi jika Anda memiliki beberapa objek komponen didalam objek kontainer. Anda akan memanggil metode ini untuk setiap komponen.

5.4.1 FlowLayout Manager

FlowLayout Manager adalah default manager untuk *Panel class* dan subkelasnya, termasuk *applet class*. Cara meletakkan komponen dari *FlowLayout Manager* dimulai dari kiri ke kanan dan dari atas ke bawah, dimulai dari pojok kiri atas. Seperti pada saat Anda mengetik menggunakan editor kata pada umumnya. Berikut adalah bagaimana *FlowLayout Manager* bekerja, dimana memiliki tiga konstruktor seperti daftar di bawah ini.

FlowLayout Constructors
`FlowLayout()`
Membuat objek baru *FlowLayout* dengan posisi di tengah dan lima unit horizontal dan vertikal gap dimasukkan pada komponen sebagai *default*.
`FlowLayout(int align)`
Membuat objek baru *FlowLayout* dengan posisi spesifik dan lima unit horizontal dan vertikal gap dimasukkan pada komponen sebagai *default*.
`FlowLayout(int align, int hgap, int vgap)`
Membuat objek baru *FlowLayout* dengan argumen pertama sebagai posisi pada komponen

FlowLayout Constructors
dan *hgap* untuk horizontal dan *vgap* untuk vertikal pada komponen

Tabel 1.3.1: Konstruktor FlowLayout

Gap dapat dikatakan sebagai jarak antara komponen dan biasanya diukur dengan satuan *pixel*. Posisi argumen mengikuti penulisan sebagai berikut :

1.FlowLayout.LEFT

2.FlowLayout.CENTER

3.FlowLayout.RIGHT

Bagaimanakah *output* dari program berikut?

```
import java.awt.*;

class FlowLayoutDemo extends Frame {
   public static void main(String args[]) {
      FlowLayoutDemo fld = new FlowLayoutDemo();
      fld.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
      fld.add(new Button("ONE"));
      fld.add(new Button("TWO"));
      fld.add(new Button("THREE"));
      fld.setSize(100, 100);
      fld.setVisible(true);
   }
}
```

Gambar berikut adalah contoh dari hasil yang berjalan pada *platform Window*.

Tabel 13.1: Contoh hasil dalam window

5.4.2.BorderLayout Manager

BorderLayout membagi kontainer menjadi lima bagian diantaranya utara, selatan, timur, barat, dan tengah. Setiap komponen dimasukkan ke dalam *region* yang spesifik. *Region* utara dan selatan membentuk jalur horizontal sedangkan *region* timur dan barat membentuk jalur vertikal. Dan *region* tengah berada pada perpotongan jalur horizontal dan vertikal. Tampilan ini adalah bersifat *default* untuk objek *Window*, termasuk objek dari subkelas *Window* yaitu tipe

Frame dan *Dialog*.

Konstruktor BorderLayout
`BorderLayout()`
Membuat objek *BorderLayout* baru tanpa spasi yang diaplikasikan diantara komponen yang berbeda.
`BorderLayout(int hgap, int vgap)`
Membuat objek *BorderLayout* baru dengan spasi *unit hgap* horizontal dan *unit vgap* vertikal yang diaplikasikan diantara komponen yang berbeda.

Tabel 1.3.2: Konstruktor BorderLayout

Seperti pada *FlowLayout Manager*, parameter *hgap* dan *vgap* disini juga menjelaskan jarak antara komponen dengan kontainer.

Untuk menambahkan komponen kedalam region yang spesifik, gunakan metode menambahkan dan melewatkan dua argumen yaitu : komponen yang ingin dimasukkan ke dalam *region* dan *region* mana yang ingin dipakai untuk meletakkan komponen. Perlu diperhatikan bahwa hanya satu komponen yang dapat dimasukkan dalam satu *region*. Menambahkan lebih dari satu komponen pada kontainer yang bersangkutan, maka komponen yang terakhir ditambahkan yang akan ditampilkan. Berikut ini adalah daftar dari kelima *region*.

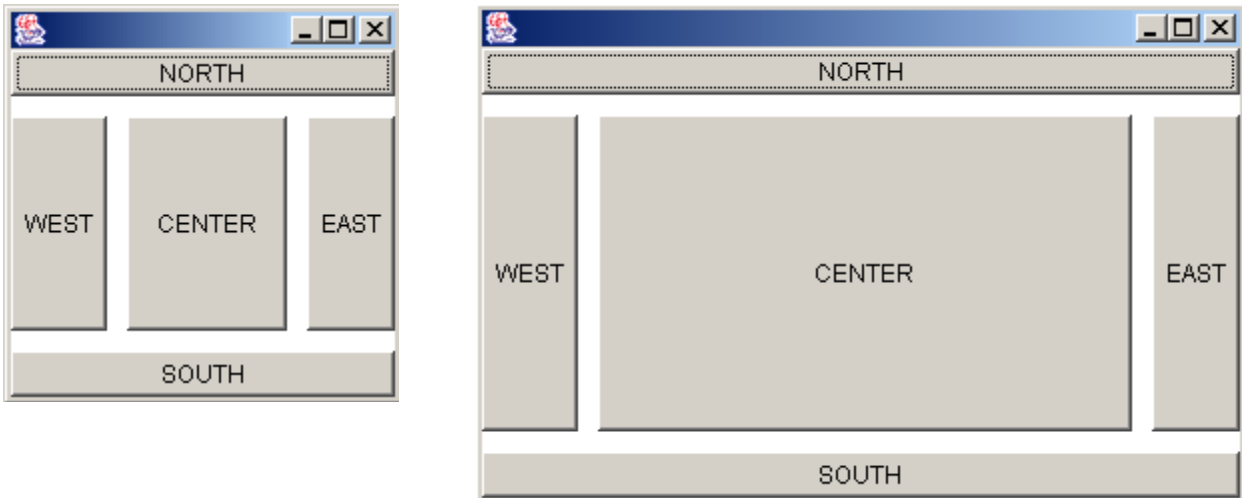
1. *BorderLayout.NORTH*
2. *BorderLayout.SOUTH*
3. *BorderLayout.EAST*
4. *BorderLayout.WEST*
5. *BorderLayout.CENTER*

Berikut ini adalah contoh program yang menunjukkan bagaimana *BorderLayout* bekerja.

```
import java.awt.*;

class BorderLayoutDemo extends Frame {
   public static void main(String args[]) {
      BorderLayoutDemo bld = new BorderLayoutDemo();
      bld.setLayout(new BorderLayout(10, 10)); //may remove
      bld.add(new Button("NORTH"), BorderLayout.NORTH);
      bld.add(new Button("SOUTH"), BorderLayout.SOUTH);
      bld.add(new Button("EAST"), BorderLayout.EAST);
      bld.add(new Button("WEST"), BorderLayout.WEST);
      bld.add(new Button("CENTER"), BorderLayout.CENTER);
      bld.setSize(200, 200);
      bld.setVisible(true);
   }
}
```

Berikut ini adalah hasil dari contoh program tersebut. Gambar kedua menunjukkan efek dari mengubah bentuk dari *frame*.



Gambar 1.3.2: hasil contoh program

5.4.3 GridLayout Manager

Dengan *GridLayout manager*, komponen juga diposisikan dari kiri ke kanan dan dari atas ke bawah seperti pada *FlowLayout manager. GridLayout manager* membagi kontainer menjadi baris dan kolom. Semua *region* memiliki ukuran yang sama. Hal tersebut tidak mempedulikan ukuran sebenarnya dari komponen.

Berikut ini adalah daftar dari konstruktor untuk *GridLayout class*.

Konstruktor GridLayout
`GridLayout()`
Membuat objek *GridLayout* baru dengan satu baris dan satu kolom sebagai *default*
`GridLayout(int rows, int cols)`
Membuat objek *GridLayout* baru dengan jumlah baris dan kolom sesuai dengan keinginan
`GridLayout(int rows, int cols, int hgap, int vgap)`
Membuat objek *GridLayout* baru dengan jumlah baris dan kolom yang ditentukan. Unit spasi *hgap* horizontal dan *vgap* vertikal diaplikasikan ke dalam komponen.

Tabel 1.3.3: Konstruktor GridLayout

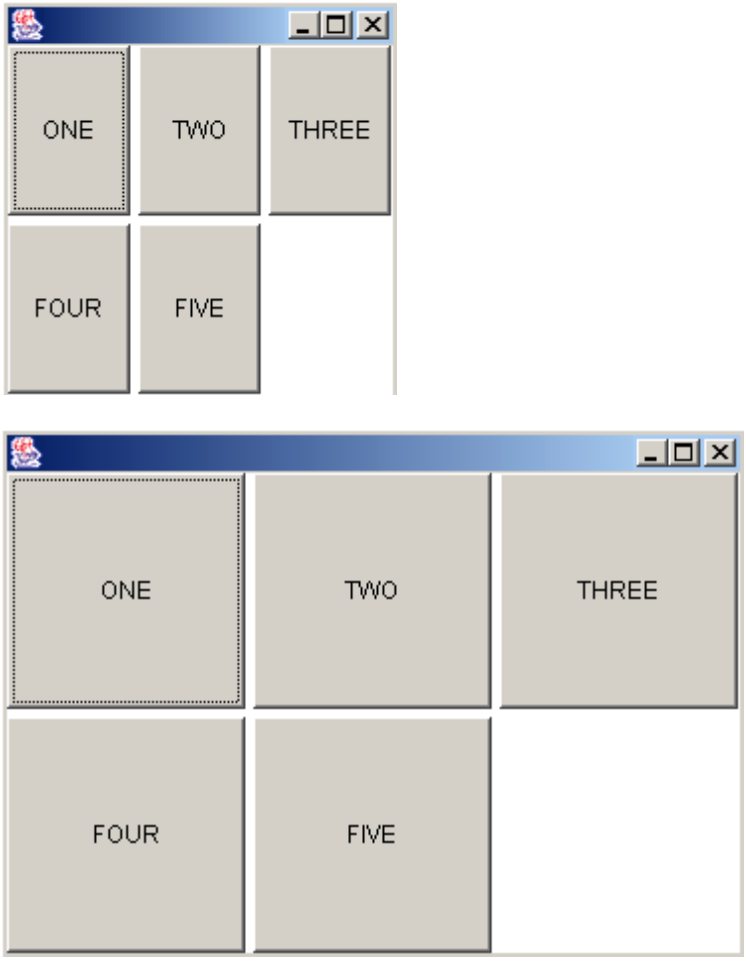
Cobalah program ini.

```
import java.awt.*;

class GridLayoutDemo extends Frame {
   public static void main(String args[]) {
      GridLayoutDemo gld = new GridLayoutDemo();
      gld.setLayout(new GridLayout(2, 3, 4, 4));
      gld.add(new Button("ONE"));
```

```
            gld.add(new Button("TWO"));
            gld.add(new Button("THREE"));
            gld.add(new Button("FOUR"));
            gld.add(new Button("FIVE"));
            gld.setSize(200, 200);
            gld.setVisible(true);
        }
    }
```

Berikut ini adalah hasil dari program.



Gambar 1.3.3: hasil contoh program

5.4.4 Panel dan Tampilan kompleks

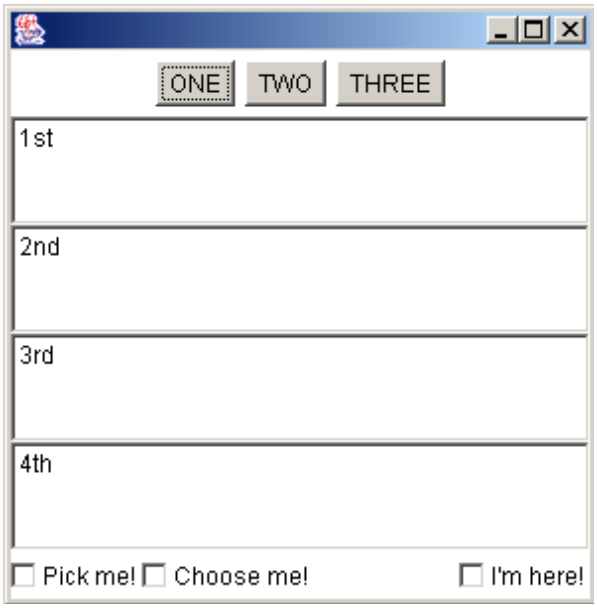
Untuk membuat tampilan yang lebih komplek, Anda dapat menggabungkan *layout manager* yang berbeda dengan menggunakan *panel*. Ingatlah bahwa *panel* adalah kontainer dan komponen pada saat yang sama. Anda dapat memasukkan komponen ke dalam *panel* dan kemudian menambahkan *panel* ke dalam *region* yang Anda inginkan di dalam kontainer.

Perhatikan teknik yang digunakan pada contoh berikut.

```
            import java.awt.*;

            class ComplexLayout extends Frame {
               public static void main(String args[]) {
                  ComplexLayout cl = new ComplexLayout();
                  Panel panelNorth = new Panel();
                  Panel panelCenter = new Panel();
                  Panel panelSouth = new Panel();
                  /* North Panel */
                  //Panels use FlowLayout by default
                  panelNorth.add(new Button("ONE"));
                  panelNorth.add(new Button("TWO"));
                  panelNorth.add(new Button("THREE"));
                  /* Center Panel */
                  panelCenter.setLayout(new GridLayout(4,4));
                  panelCenter.add(new TextField("1st"));
                  panelCenter.add(new TextField("2nd"));
                  panelCenter.add(new TextField("3rd"));
                  panelCenter.add(new TextField("4th"));
                  /* South Panel */
                  panelSouth.setLayout(new BorderLayout());
                  panelSouth.add(new Checkbox("Choose me!"),
                                                  BorderLayout.CENTER);
                  panelSouth.add(new Checkbox("I'm here!"),
                                                    BorderLayout.EAST);
                  panelSouth.add(new Checkbox("Pick me!"),
                                                    BorderLayout.WEST);
                  /* Adding the Panels to the Frame container */
                  //Frames use BorderLayout by default
                  cl.add(panelNorth, BorderLayout.NORTH);
                  cl.add(panelCenter, BorderLayout.CENTER);
                  cl.add(panelSouth, BorderLayout.SOUTH);
                  cl.setSize(300,300);
                  cl.setVisible(true);
               }
}
```

Berikut ini adalah hasil dari program.



Gambar 1.3.4: Hasil dari contoh program

5.5 Komponen Swing

Seperti pada *package AWT*, *package* dari *Swing* menyediakan banyak kelas untuk membuat aplikasi GUI. *Package* tersebut dapat ditemukan di *javax.swing*. Perbedaan utama antara keduanya adalah komponen *Swing* ditulis menyeluruh menggunakan *Java* mengingat yang belakangan tidak. Kesimpulannya, program GUI ditulis menggunakan banyak kelas dari *package Swing* yang mempunyai tampilan *look* and *feel* yang sama meski dijalankan pada beda *paltform*. Lebih dari itu, *Swing* menyediakan komponen yang lebih menarik seperti *color chooser* dan *option pane.*

Nama dari komponen GUI milik *Swing* hampir sama persis dengan komponen GUI milik AWT. Perbedaan jelas terdapat pada penamaan komponen. Pada dasarnya, nama komponen *Swing* sama dengan nama komponen AWT tetapi dengan tambahan huruf J pada prefixnya. Sebagai contoh, satu komponen dalam AWT adalah *button class*. Sedangkan pada *Swing*, nama komponen tersebut menjadi *Jbutton class*. Berikut adalah daftar dari komponen *Swing*.

Komponen Swing	***Penjelasan***
JComponent	Kelas induk untuk semua komponen *Swing*, tidak termasuk *top-level* kontainer
JButton	Tombol “push”. Korespondesi pada *button class* dalam *package AWT*
JCheckBox	Item yang dapat dipilih atau tidak oleh pengguna. Korespondensi pada *checkbox class* dalam *package AWT*
JFileChooser	Mengijinkan pengguna untuk memilih sebuah *file*. Korespondensi pada *filechooser class* dalam *package AWT*

Komponen Swing	*Penjelasan*
JTextField	Mengijinkan untuk mengedit *text* satu baris. Korespondensi pada *textfield class* dalam *package AWT*.
JFrame	Turunan dan korepondensi pada *frame class* dalam *package AWT* tetapi keduanya sedikit tidak cocok dalam kaitannya dengan menambahkan komponen pada kontainer. Perlu mendapatkan *content pane* yang terbaru sebelum menambah sebuah komponen.
JPanel	Turunan *Jcomponent*. Kontainer *class* sederhana tetapi bukan *top-level*. Korespondensi pada *panel class* dalam *package AWT*.
JApplet	Turunan dan korepondensi ke *Applet class* dalam *package AWT*. Juga sedikit tidak cocok dengan *applet class* dalam kaitannya dengan menambahkan komponen pada kontainer
JOptionPane	Turunan *Jcomponent*. Disediakan untuk mempermudah menampilkan pop-up kotak dialog.
JDialog	Turunan dan korespondensi pada *dialog class* dalam *package AWT*. Biasanya digunakan untuk menginformasikan sesuatu kepada pengguna atau *prompt* pengguna untuk *input*.
JColorChooser	Turunan *Jcomponent*. Mengijinkan pengguna untuk memilih warna

Tabel 1.4: Beberapa komponen Swing

Untuk daftar yang lengkap dari komponen *Swing*, Anda dapat melihatnya di dokumentasi API.

5.5.1 Setting Up Top-Level Containers

Seperti disebutkan diatas, *top-level containers* seperti *Jframe* dan *Japplet* dalam *Swing* sangat tidak cocok dengan AWT. Ini adalah syarat menambahkan komponen ke dalam kontainer. Jika Anda ingin menambahkan langsung sebuah komponen kedalam kontainer sebagai kontainer AWT, pertama-tama Anda telah mendapatkan *content pane* dari kontainer. Untuk melakukan hal tersebut, Anda akan menggunakan metode *getContentPane* dari kontainer.

5.5.2 Contoh Jframe

```
import javax.swing.*;
import java.awt.*;

class SwingDemo {
   JFrame frame;
   JPanel panel;
   JTextField textField;
   JButton button;
   Container contentPane;
   void launchFrame() {
      /* initialization */
      frame = new JFrame("My First Swing Application");
      panel = new JPanel();
```

```
            textField = new JTextField("Default text");
            button = new JButton("Click me!");
            contentPane = frame.getContentPane();
            /* add components to panel– uses FlowLayout by default */
            panel.add(textField);
            panel.add(button);
            /* add components to contentPane– uses BorderLayout */
            contentPane.add(panel, BorderLayout.CENTER);
            frame.pack();
            //causes size of frame to be based on the components
            frame.setVisible(true);
        }
        public static void main(String args[]) {
            SwingDemo sd = new SwingDemo();
            sd.launchFrame();
        }
    }
```

Perlu diperhatikan pada *package java.awt* masih saja diimpor karena *layout manager* yang digunakan terdapat pada *package* tersebut. Juga, memberi judul pada *frame* dan mengepack komponen di dalam *frame* dapat juga dilakukan untuk *frame AWT.*

Petunjuk penulisan program:
Perhatikan penulisan kode yang digunakan pada contoh ini tampak berlawanan dengan contoh untuk AWT. Komponen dideklarasikan sebagai fields, metode launchFrame ditentukan, dinisialisasikan dan penambahan semua komponen dilaksanakan di dalam metode launchFrame. Kita tidak lagi meng-extend Frame class. Keuntungan penggunaan gaya ini akan lebih berguna ketika sampai pada event handling.

Berikut adalah keluaran dari program diatas.

Gambar 1.4.2: Hasil contoh program

5.5.3 Contoh JOptionPane

```
import javax.swing.*;

class JOptionPaneDemo {
    JOptionPane optionPane;
    void launchFrame() {
        optionPane = new JOptionPane();
        String name = optionPane.showInputDialog("Hi, what's your
                                                  name?");
        optionPane.showMessageDialog(null,
                "Nice to meet you, " + name + ".", "Greeting...",
                optionPane.PLAIN_MESSAGE);
        System.exit(0);
    }
    public static void main(String args[]) {
        new JOptionPaneDemo().launchFrame();
```

```
                }
}
```

Lihat, begitu mudahnya memasukkan *input* dari *user*.
Berikut ini adalah hasil dari contoh program diatas

Gambar 1.4.3: Hasil contoh program

5.6 Latihan

5.6.1 Tic-Tac-Toe

Buatlah tampilan GUI untuk program *tic-tac-toe*. Papannya terdiri dari enam kotak. Ingatlah bahwa Anda akan menambahkan kode ini pada tahap akhir untuk mengatasi interaksi antar pengguna. Jadi, desainlah papan Anda dengan benar. Pastikanlah Anda memilih komponen yang pantas untuk papan tersebut. keluarkan semua sisi artistik Anda. Anda dapat menggunakan AWT atau *Swing* untuk latihan ini.

Gambar 1.5.1: papan Tic-Tac-Toe