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.

Sebelum 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.

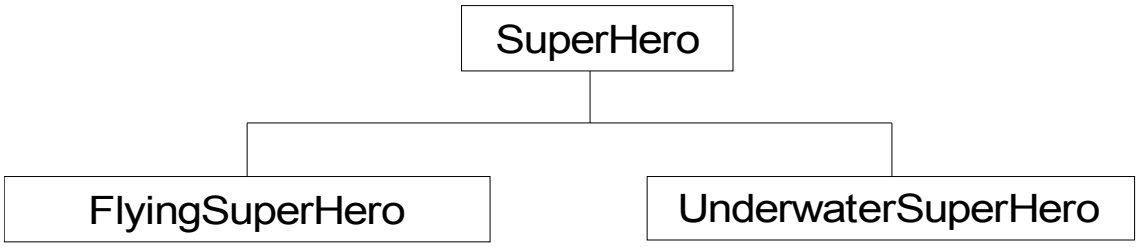
1.2.9 Abstraksi

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 kode 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.*