**MODUL PRAKTIKUM IV**

**ABSTRACT CLASS, INTERFACE DAN POLYMORPHISM**

1. **Tujuan**

1. Siswa mampu memahami class abstract

2. Siswa mampu memahami interface

3. Siswa mampu memahami polymorphism

4. Siswa mampu menciptakan method invocation

5. Siswa mampu memahami polymorphic arguments

1. **Dasar Teori** 
   1. **Kelas Abstrak (Abstract Class)**

Kelas abstrak adalah kelas yang terletak di posisi tertinggi dalam hierarki class. Kelas ini tidak dapat diinstansiasi karena masih bersifat abstrak. Kelas ini hanya berisi variabel-variabel umum dan deklarasi method tanpa detail implementasi. Selanjutnya kelas-kelas yang menjadi turunan dari kelas abstraklah yang akan mengimplementasikan method-method tanpa implementasi tersebut.

* 1. **Interface**

Interface memungkinkan pemrogram untuk berbagi konstanta atau menentukan bentuk metode yang dapat digunakan oleh sejumlah kelas.

Beda interface dengan kelas abstrak :

* 1. Kelas abstrak dapat mengandung method abstract maupun method tidak abstract, sedangkan interface hanya boleh mengandung method abstract
  2. Kelas abstrak dapat mendeklarasikan variabel instans, sedangkan interface hanya dapat mendeklarasikan konstanta
  3. Kelas abstrak digunakan oleh kelas lain melalui metode pewarisan dengan kata kunci **extends**, sedangkan sebuah interface diimplementasikan ke dalam suatu class dengan menggunakan kata kunci **implements**, dan sebuah kelas dapat mengimplementasikan lebih dari satu interface.

1. **Polymorphism**

Polymorphism berarti banyak bentuk. Polymorphism merupakan konsep sederhana dalam bahasa pemrograman berorientasi obyek yang berarti kemampuan sebuah obyek untuk menghasilkan aksi yang berbeda, bila method yang sama dipanggil, dimana aksi method yang dikerjakan tergantung dari tipe obyeknya.

Polymorphism sering dikaitkan dengan penggunaan lebih dari satu metoda dengan nama sama. Penggunaan metoda dengan nama sama dapat diterapkan dengan method overloading dan method overriding. Pada saat obyek yang sudah dibuat tersebut memanggil overridden method pada parent class, compiler Java akan melakukan invocation (pemanggilan) terhadap overriding method pada subclass dimana yang seharusnya dipanggil adalah overridden method.

1. *Virtual Method Invocation* (VMI)

*Virtual Method Invocation* (VMI) terjadi karena objek yang sudah di buat tersebut memanggil overriden method pada parent class dan akan melakukan panggilan pada overriding method yang seharusnya adalah overriden.

Virtual Method Invocation (VMI) bisa terjadi jika adanya polimorfisme dan overriding dalam sebuah program. Pada saat obyek yang sudah di buat tersebut memanggil overridden method pada parent class, compiler Java akan melakukan invocation (pemanggilan) terhadap overriding method pada subclass, dimana yang seharusnya di panggil adalah overridden method. Berikut contoh terjadinya VMI:

|  |
| --- |
| Listing Program |
| Class Parent{  intx=5;  public void Info(){  System.out.println(“Ini class Parent”);  }  }  Class Child extends Parent{  intx=10;  public void Info(){  System.out.println(“Ini class Child”);  }  }  Public class Tes{  Public static void main(String args[]){  Parent tes = new Child();  System.out.println(“Nilai x =“+tes.x);  tes.Info();  }  } | |

Hasil dari running program diatas adalah sebagai berikut:

Nilaix=5

Ini class Child

1. *Polymorphic arguments*

*Polymorphic arguments* adalah tipe suatu parameter yang menerima suatu nilai yang bertipe subclass-nya. Pernyataan instance of sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments } Seringkali pemakaian instance of diikuti dengan casting object dari tipe parameter ke tipe asal.

Polymorphic arguments adalah tipe suatu parameter yang menerima suatu

Nilai yang bertipe subclass-nya. Berikut contoh dari polymorphics arguments:

|  |
| --- |
| Listing Program |
| Class Pegawai{  }  Class Manajer extends Pegawai{  …  }  Public class Tes{  Public static void Proses(Pegawai peg){  …  }  Public static void main(String args[]){  Manajer man = new Manajer(); Proses(man);  }  } | |

Pernyataan instance of sangat berguna untuk mengetahui tipe asal dari suatu Polymorphic arguments. Untuk lebih jelasnya, misalnya dari contoh program sebelumnya, kita sedikit membuat modifikasi pada class Tes dan ditambah sebuah class baru Kurir, seperti yang tampak dibawah ini:

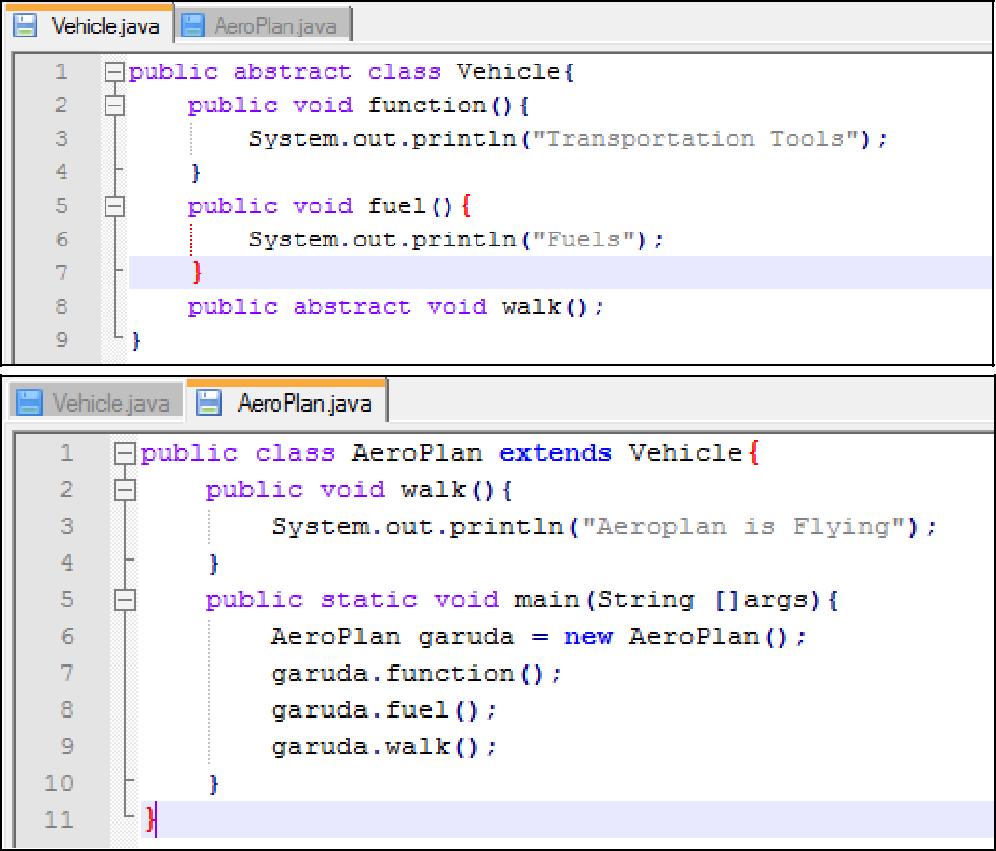
|  |
| --- |
| Listing Program |
| Class Kurir extends Pegawai  }  Public class Tes{  Public static void Proses(Pegawai peg){  if(peginstanceofManajer){  …lakukantugas-tugasmanajer…  }elseif(peginstanceofKurir){  …lakukantugas-tugaskurir…  }else{  …lakukantugas-tugaslainnya…  }  }  Public static void main(String args[]){  Manajer man = new Manajer();  Kurir kur = new Kurir();  Proses(man);  Proses(kur);  }  } | |
|  | |

Seringkali pemakaian instance of diikuti dengan casting object dari tipe parameter ke tipe asal. Pada saat sudah melakukan instance of dari tipe manajer, kita dapat melakukan casting object ke tipe asalnya, yaitu manajer. Caranya adalah seperti berikut:

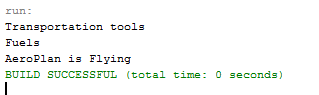
|  |
| --- |
| Listing Program |
| if(peginstanceofManajer)  {  Manajer man =(Manajer)peg;  …lakukantugas-tugasmanajer…  } | |

1. **Latihan**

Latihan 1 : Abstract Class

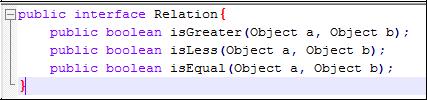


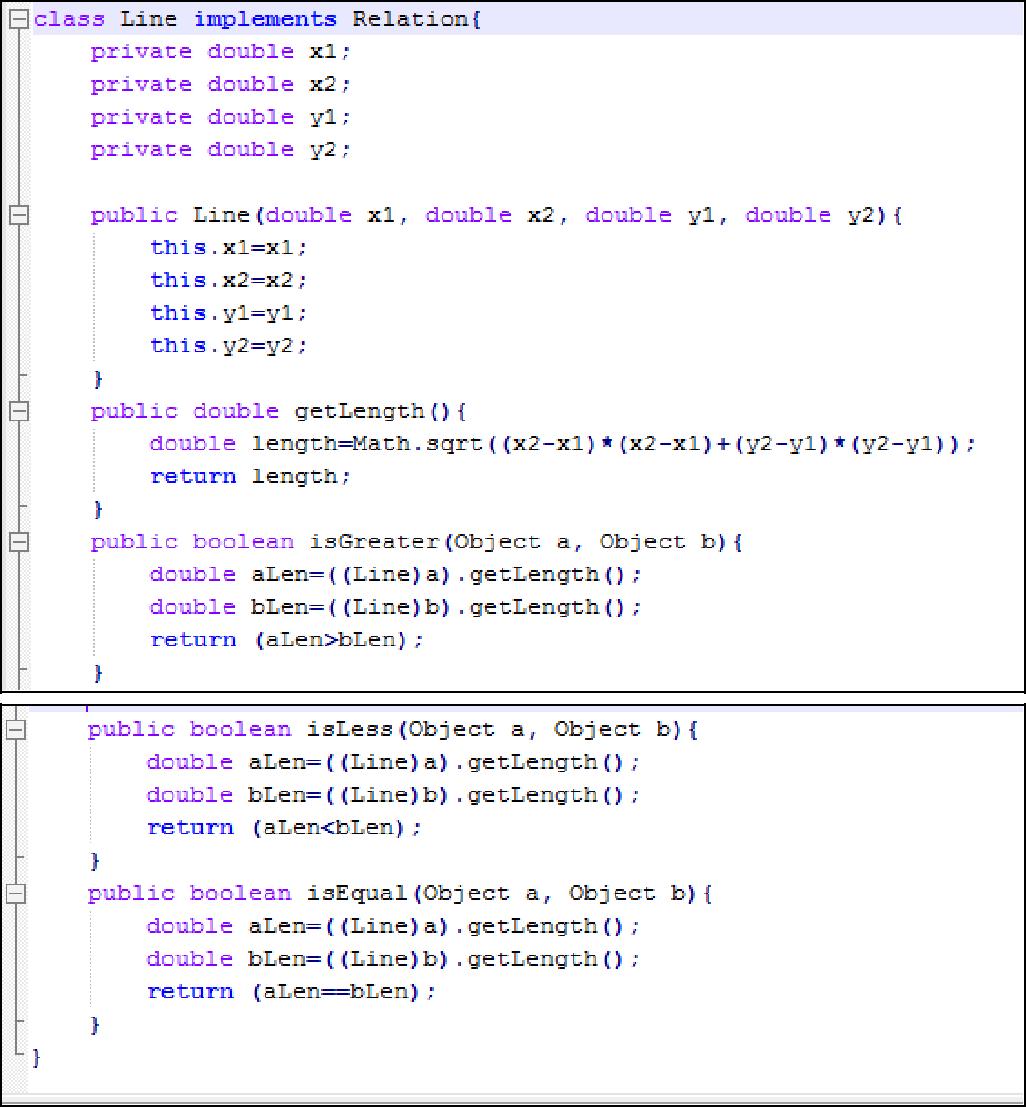
1. Tampilan Output program



1. Ubahlah method fuel pada class Vehicle menjadi abstract. Kemudian lakukan kompilasi ulang pada kedua class. Output setelah kompilasi adalah
2. Lakukan override method fuel pada class AeroPlan.
3. Jelaskan mengapa langkah b dan c berbeda!

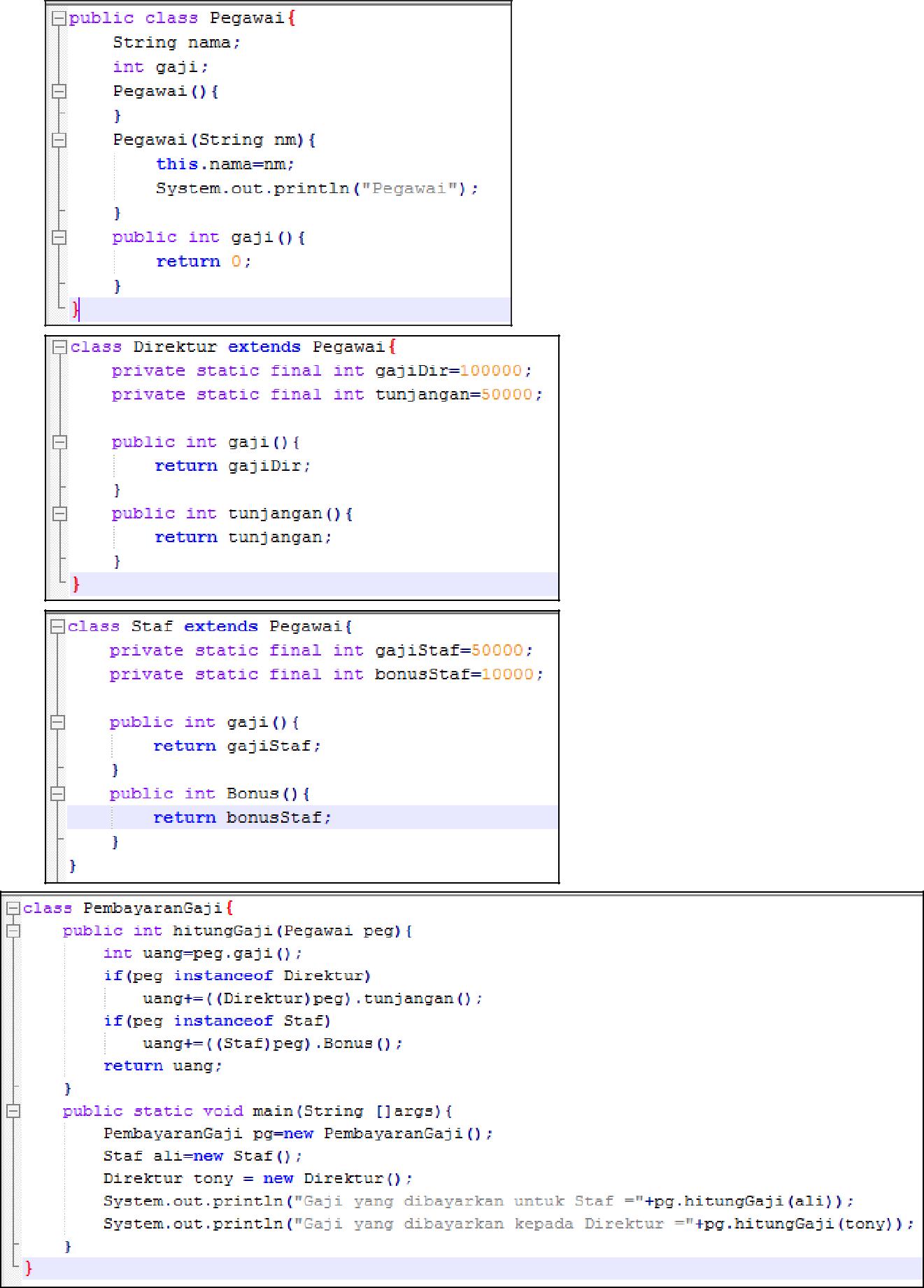
**Latihan 2 : Interface**





1. Tambahkan sebuah class untuk menguji program di atas!
2. Tampilan Output program
3. Kosongkan isi method isEqual. Kemudian jalankan program. Output program adalah
4. Hapus method isEqual. Kemudian jalankan program! Output program adalah

**Latihan 3 Polymorphism**

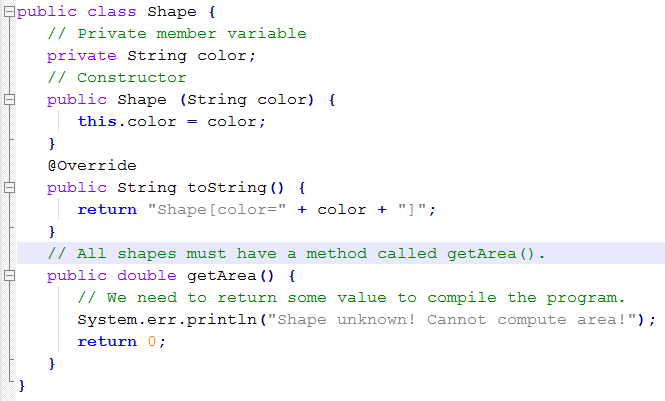




a. Output program

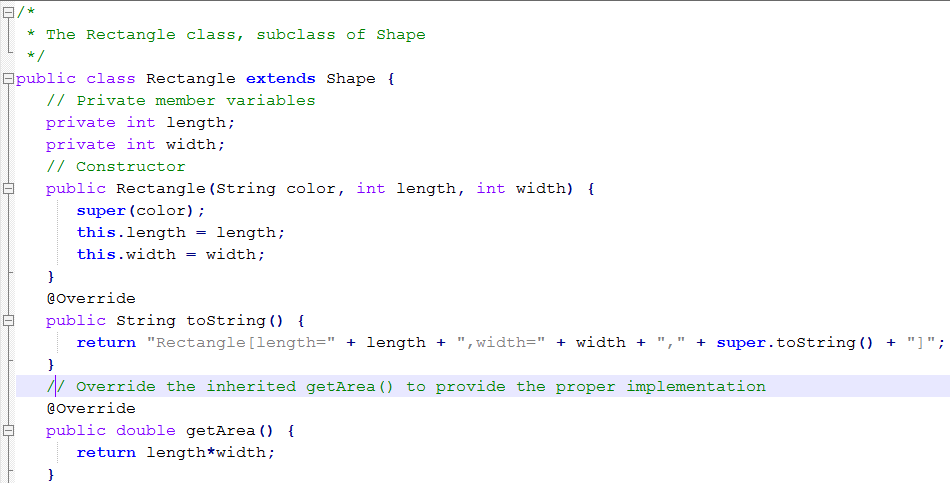
**Latihan 4 Polymorphism**

File Shape.java

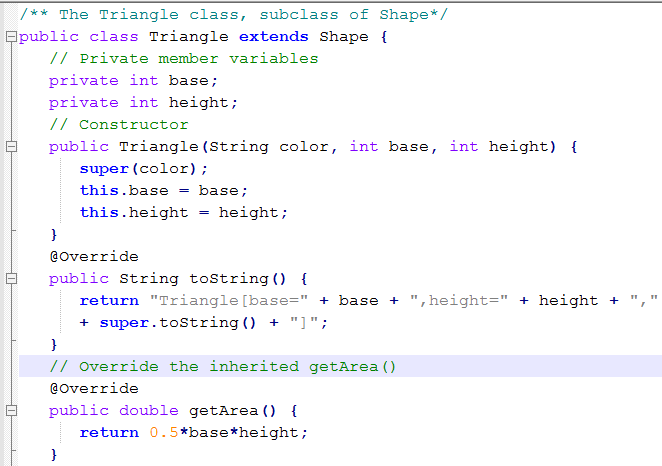




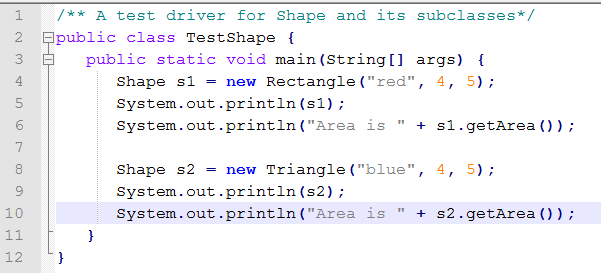
File Rectangle.java



File Triangle.java



File TestShape.java

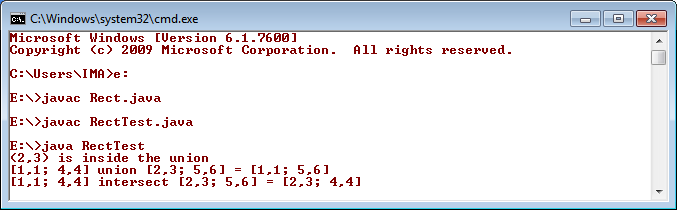


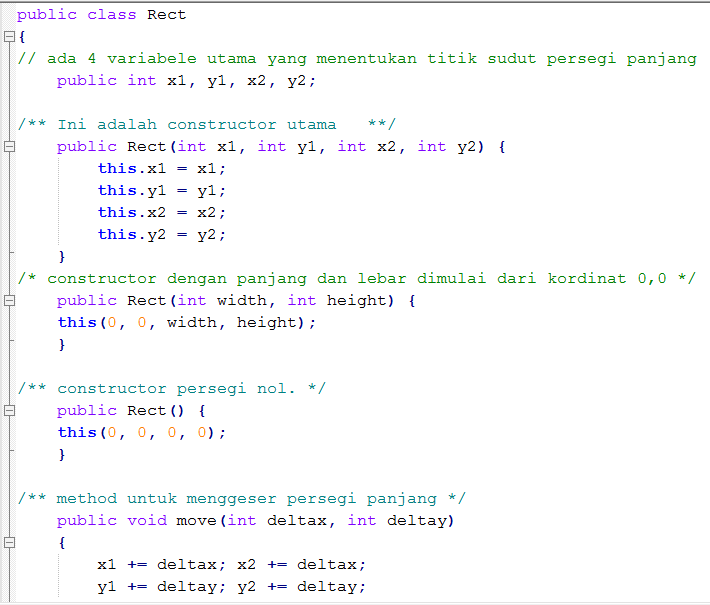
a. Tampilan Output program

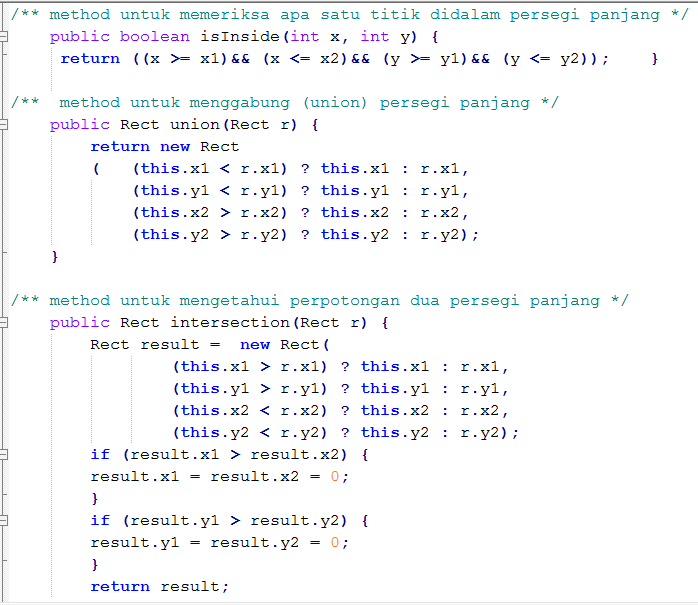
b. Jelaskan kode program baris 4 sampai 10 di atas!

1. **Tugas Praktikum**

Persegi panjang dibentuk oleh empat titik sudut. Membutuhkan constructor dan method seperti program di bawah ini. Buat subclass dengan menggunakan class Rect dan menghasilkan tampilan minimal seperti berikut. Kerjakan berkelompok, setiap kelompok terdiri dari 2 anak.







**SELAMAT MENCOBA DAN BERKREASI**

**--Siswa Telkom Hebat--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**

**Kedisiplinan dalam Belajar**

**Akan Mampu Membentuk Pribadi**

**yang Teratur dan Mandiri di Masa Depan**

**--Siswa Telkom Luar Biasa--**