Review

Alfan

Jelaskan tentang konsep Cohesion & Coupling !

Overloaded Methods

Method-method dalam sebuah kelas boleh mempunyai nama sama, tetapi signature-nya berbeda.

Berbeda : tipe parameter, urutan parameter, banyaknya parameter

Buatlah contohnya !

Overloaded Methods

```
class A {
    //apakah 2 method di bawah overload ?
    public void ma(int a) {}
    public int ma(int a) {}
}
```

```
class A {
    //apakah 2 method di bawah overload ?
    public void ma(int aa) {}
    public void ma(int bb) {}
}
```

```
class A {
    //apakah 2 method di bawah overload ?
    public void ma(int a) {}
    public void ma(float a) {}
}
```

Overloaded Methods

```
class A {
    //apakah 2 method di bawah overload ?
    public void ma(int a) {}
    public void ma(int a, int b) {}
}
```

```
class A {
    //apakah 2 method di bawah overload ?
    public void ma(int a, float b) {}
    public void ma(float a, int b) {}
}
```

Catching Exceptions

```
public class ExcDemo  {
    public static void throwIt ()  {
        System.out.print("throwit ");
        throw new RuntimeException();
    }

    public static void main(String [] args) {
        try {
            System.out.print("hello ");
            throwIt();
            System.out.print("goodbye");
        } catch (Exception re) {
            System.out.print("caught ");
        } finally {
            System.out.print("finally ");
        }
        System.out.println("after ");
    }
}
```

Catching Exceptions

```
public class ExcDemo  {
    public static void throwIt ()  {
        System.out.print("throwit");
        throw new IOException();
    }

    public static void throwIt2 ()  {
        System.out.print("throwit2");
        throwIt();
    }

    public static void main(String [] args) {
        try {
            throwIt2();
        } catch (Exception re) {
            System.out.print("caught");
        }
    }
}
```

Catching Exceptions

```
public class ExcDemo  {

    public static void throwit() {
        int a = Integer.parseInt("123ab");
    }

    public static void main(String [] args) {
        try {
            throwit();
        } catch (NumberFormatException e) {
            System.out.print("caught 1");
            int a = 1 / 0;
        } catch (ArithmeticException e) {
            System.out.print("caught 2");
        }
        System.out.print("selesai");
    }
}
```

```
public interface xyz {
     void abc() throws IOException;
}

public interface pqr {
     void abc() throws FileNotFoundException;
}

public class Impl implements xyz, pqr {
     // insert code

}
```

Manakah dari statement di bawah yang dapat menggantikan posisi **//insert code** ?

1. public void abc() throws IOexception
2. public void abc() throws FileNotFoundException
3. public void abc() throws FileNotFoundException, IOexception
4. public void abc() throws IOexception, FileNotFoundException

Konsep Interface & Polimorfisme

Tentukan nilai kebenaran dari pernyataan-pernyataan di bawah:

- Inteface menspesifikasikan operasi/method sebuah kelas tanpa perlu mengetahui implementasinya.
- Interface boleh mengandung method yang **Tidak** abstrak (ada implementasi)
- Interface boleh mempunyai konstanta
- Interface boleh mempunyai instance variable
- Salah satu manfaat dari konsep interface adalah untuk meningkatkan coupling.
- Interface dapat di-*extend* (dengan kata kunci extends)

Konsep Interface & Polimorfisme

Perhatikan kode berikut:

```
public interface Rotatable {
	void rotate();
}

public class Circle implements Rotatable {
	public void rotate() {
		System.out.println("rotate circle");
	}

	public void draw() {
		 System.out.println("draw circle");
	}
}

public class Tire implements Rotatable {
	public void rotate() {
		System.out.println("rotate tire");
	}
}
```

Konsep Interface & Polimorfisme

Apa yang terjadi jika potongan kode berikut dikompilasi, dan kemudian dijalankan ?

(Soal 1)

```
Rotatable r = new Rotatable();
```

(Soal 2)

```
Rotatable[] r = new Rotatable[2];
r[0] = new Circle();
r[1] = new Tire();
```

(Soal 3)

```
Circle c = new Circle();
c.draw();
c.rotate();
Rotatable r = c;
r.draw();
r.rotate();
```

(Soal 4)

```
Rotatable r = new Circle();
Circle c = r;
c.draw();
```

Konsep Interface & Polimorfisme

Apa yang terjadi jika potongan kode berikut dikompilasi, dan kemudian dijalankan ?

(Soal 5)

```
Rotatable r = new Circle();
r.rotate();

Tire t = new Tire();
r = t;
r.rotate();
```

Polimorfisme !

Konsep Interface & Polimorfisme

Apa yang terjadi jika potongan kode berikut dikompilasi, dan kemudian dijalankan ?

(Soal 6)

```
Rotatable r = new Circle();
Tire t = (Tire)r;
t.rotate();
```

(Soal 7)

```
Object o = new Circle();
Rotatable r = o;
r.rotate();
```

Interface Comparable<T>

Buatlah kelas **Garis** yang merepresentasikan sebuah garis di bidang kartesius. Garis terdiri dari 2 buah Point yang merepresentasikan 2 buah ujung Garis.

Jika diurutkan, Garis yang paling pendek ada di sebelah kiri.

```
public class Garis implements Comparable<Garis> {
   private Point p1;
   private Point p2;

   public Garis(int x1, int y1, int x2, int y2) {
       ...
   }

   public double getPanjang() {...}

   public int compareTo(Garis other) {...}
}
```

```
public interface Translatable {
        void translate(int dx, int dy);
}

public class Point implements Translatable {
        private int x;
        private int y;
        ...
        public void translate(int dx, int dy) {...}
}

public class Point3D extends Point {
        private int z;
        ...
}
```

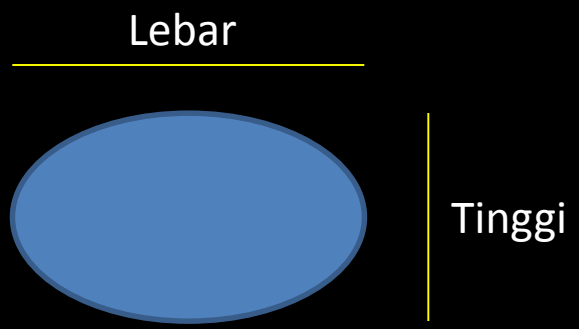
```
Point3D p3 = new Point3D(3,4,5);
Point p = p3; //OK
Translatable t = p3 //OK

Translatable t2 = new Point3D(2,4,3); //OK
```

Konsep Inheritance & Polimorfisme

Tentukan nilai kebenaran dari pernyataan-pernyataan di bawah:

- Di Java, sebuah kelas boleh merupakan turunan dari lebih dari satu kelas (multiple inheritance).
- Atribut yang mempunyai access modifier private tidak diturunkan ke subclass-nya.
- Di java, method bersifat polimorfik

Kelas Elips merepresentasikan sebuah bidang **Elips** yang mempunyai **lebar** dan **tinggi** tertentu.




```
public class Elips {
   private double lebar;
   private double tinggi;

   public Elips(double lebar, double tinggi) {
      this.lebar = lebar;
      this.tinggi = tinggi;
   }

   //setters & getters ...
}
```

Implementasikan kelas **Lingkaran**, dengan memanfaatkan kelas Elips !

```
public class Lingkaran extends Elips {

    public Lingkaran(double diameter) {
        ...
    }

    public double getDiameter() {
        ...
    }

    public double getRadius() {
        ...
    }

    public double getLuas() {
        ...
    }
}
```

```java
public class A {
    private int varA;
    protected int varB = 10;

    public A(int varA) {
       this.varA = varA;
    }

    public int getVarA() {
       return varA;
    }

    public void ma() {...}
    public void mb() {...}
}
```

```java
public class C extends A {
    private int varC;

    public C(int varC) {
       super(20);
       this.varC = varC;
    }

    public int getVarC() {
       return varC;
    }

    public void mc() {...}
}
```

Method apa saja yang ada di kelas C ?

Instance variable apa saja yang bisa diakses dari kelas C ?

Polimorfisme

- Polimorfisme semu : Overloading
- Polimorfisme sejati : Overriding

- Method bersifat polimorfik
  - Untuk melihat implementasi method yang dijalankan, coba Anda lihat OBJECT-nya (dynamic binding)
- Instance variables tidak bersifat polimorfik
  - Untuk melihat variable mana yang diakses, coba Anda lihat TIPE VARIABLE-nya (static binding)

Mana saja method di kelas **GuruKimia** yang merupakan hasil **Overriding** ?

```
public class Guru {
   public void teach() {System.out.print("mengajar");}
   public void read() {System.out.print("membaca");}
   public double evaluate(String NPM) {...}
}

public class GuruKimia extends Guru {
   public void teach() {System.out.print("mengajar kimia");}
   public void read(String buku) {System.out.print("membaca" + buku);}
   public double evaluate(String NPM) {...}
}
```

```
public class Guru {
    public void teach() {
        System.out.println("mengajar");
    }
}

public class GuruKimia extends Guru {
    public void teach() {
        System.out.println("mengajar kimia");
    }
}

public class GuruFisika extends Guru {
    public void teach() {
        System.out.println("mengajar fisika");
    }
}
```

polimorfisme

```
Guru g = new Guru();
g.teach();
GuruKimia gk = new GuruKimia();
GuruFisika gf = new GuruFisika();
g = gk;
g.teach();
g = gf;
g.teach();
```

```
public class Person {}

public class Employee extends Person {}
public class Programmer extends Employee {}
```

Manakah dari opsi-opsi berikut **yang tidak menyebabkan compile error** ?

```
1 Person p = new Person();

2 Person p = new Employee();

3 Person p = new Programmer();

4 Employee e = new Programmer();

5 Programmer p = new Programmer();
  Person per = p;
```

```
6 Person p = new Person();
  Employee e = p;

7 Programmer p = new Employee();

8 Object o = new Programmer();
  Employee e = o;
```

```
public class Person {
        public String nama = “person”;
        public void cetak() {
                System.out.println(“ini orang”);
        }
}

public class Employee extends Person {
        public String nama = “employee”;
        public void cetak() {
                System.out.println(“ini karyawan”);
        }
}
```

Output?

```
Person p = new Employee();
p.cetak();

Employee e = (Employee)p;
e.cetak();
```

```
public class Person {
        public String nama = “person”;
        public void cetak() {
                System.out.println(“ini orang”);
        }
}

public class Employee extends Person {
        public String nama = “employee”;
        public void cetak() {
                System.out.println(“ini karyawan”);
        }
}
```

Output?

```
Person p = new Employee();
System.out.println(p.nama);

Employee e = (Employee)p;
System.out.println(e.nama);
```

```
public class Person {
        public String nama = “person”;
        public void cetak() {
                System.out.println(“ini orang”);
        }
}

public class Employee extends Person {
        public String nama = “employee”;
        public void cetak() {
                System.out.println(“ini karyawan”);
        }
}
```

Output?

```
Person p = new Employee();
System.out.println(((Employee)p).nama);
```

Override method toString() dan equals() dari kelas Object !

```
public class SegitigaSiku {
   private double alas;
   private double tinggi;

   ...

   public String toString() {
      ...
   }

   public boolean equals(Object other) {
      ...
   }
}
```

Implementasikan toString() dan equals() untuk kelas SegitigaSiku.

Untuk toString(), ketika dicetak:

[SegitigaSiku: <alas>, <tinggi>]

Untuk equals(), 2 buah SegitigaSiku adalah sama jika alas dan tingginya juga sama.

Keyword Super

```
public class Point {
        private int x;
        private int y;

        public Point() {
                this.x = this.y = 0;
        }
        public Point(int x, int y) {
                this.x = x;
                this.y = y;
        }
        ...
}

public class Point3D extends Point {
        private int z;

        public Point3D() {
                this.z = 0;
        }
        public Point3D(int x, int y, int z) {
                super(x, y);
                this.z = z;
        }
}
```

Keyword Super

```
public class BankAccount {
        private double balance;

        ...

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

        ...
}
```

```
public class CheckingAccount extends BankAccount {
        private int transactionCount;

        ...

        //Override
        public void deposit(double amount) {
                super.deposit(amount);
                transactionCount++;
        }

        ...
}
```

Casting

```
public class MahasiswaUI {
    public void study() {...}
}

public class MahasiswaFasilkom extends MahasiswaUI {
    public void code() {...}
}
```

Implisit

```
MahasiswaFasilkom mhsF = new MahasiswaFasilkom();
MahasiswaUI mhsUI = mhsF;
```

Eksplisit

```
MahasiswaUI mhsUI = new MahasiswaFasilkom();
MahasiswaFasilkom mhsF = (MahasiswaFasilkom)mhsUI;
mhsF.code();
```

Casting

```
public class MahasiswaUI {
    public void study() {...}
}

public class MahasiswaFasilkom extends MahasiswaUI {
    public void code() {...}
}
```

Yang lebih aman :

```
MahasiswaUI mhsUI = new MahasiswaFasilkom();
if (mhsUI instanceof MahasiswaFasilkom) {
    MahasiswaFasilkom mhsF = (MahasiswaFasilkom)mhsUI;
    mhsF.code();
}
```

Pahami konsep instanceof !

Konsep Abstract Class

Tentukan nilai kebenaran dari pernyataan-pernyataan di bawah:

- Abstract Class tidak boleh diinstansiasi
- Abstract Class boleh mempunyai constructor
- Abstract Class boleh mempunyai instance variable
- Abstract Class minimal mempunyai satu method abstract
- Abstract Class memang dibuat untuk diturunkan

Implementasikan sebuah kelas Shape yang merepresentasikan sebuah bentuk di bidang kartesius. Shape mempunyai informasi posisi x dan y, yang merupakan posisi ujung kiri atas.

Kemudian, sebuah object Shape pasti bisa dihitung luas dan kelilingnya. Namun, pada level Shape, cara menghitung luas dan keliling masih abstract !

Kemudian, implementasikan kelas Lingkaran dan PersegiPanjang yang memanfaatkan kelas Shape tersebut !

```
public abstract class Shape {
    private int x;
    private int y;

    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {return x;}
    public int getY() {return y;}

    public abstract double getKeliling();
    public abstract double getLuas();
}
```

Katanya tidak boleh diinstansiasi, kok boleh ada constructor ?

```
public class Lingkaran extends Shape {
    private double radius;

    public Lingkaran(int x, int y, double radius) {
        super(x, y);
        this.radius = radius;
    }

    public double getKeliling() {
        return 2.0 * Math.PI * radius;
    }

    public double getLuas() {
        return Math.PI * radius * radius;
    }
}
```

```
public class PersegiPanjang extends Shape {
    private double panjang;
    private double lebar;

    public PersegiPanjang(int x, int y,
                          double panjang, double lebar) {
        super(x, y);
        this.panjang = panjang;
        this.lebar = lebar;
    }

    public double getKeliling() {
        return 2.0 * (panjang + lebar);
    }

    public double getLuas() {
        return panjang * lebar;
    }
}
```

OOD – Hubungan Antar Kelas

Tentukan hubungan antar 2 kelas berikut : Inheritance atau Aggregation.

Lingkaran – Tabung

PersegiPanjang – Persegi

Quiz – Question

Tikus – HewanPengerat

Generic Programming

Bagaimana agar algoritma/struktur data dapat diterapkan pada berbagai jenis data ?

Cara membuat dan menggunakan Generic Classes

Cara membuat dan menggunakan Generic Methods

Generic Classes

```
public class MemCell<T>
{
    private T data;

    public MemCell(T data)
    {
        this.data = data;
    }

    public T getData() { return data; }
}
```

Generic Classes

Manakah diantara cara penggunakan kelas MemCell (instansiasi) berikut yang lulus kompilasi ?

```
MemCell m = new MemCell("abc");
MemCell m = new MemCell(8.0);
MemCell<Integer> m = new MemCell<Integer>(4);
MemCell<Double> m = new MemCell<Double>(4);
MemCell<Double> m = new MemCell<Double>(4.0);
MemCell<Boolean> m = new MemCell<Boolean>(true);
```

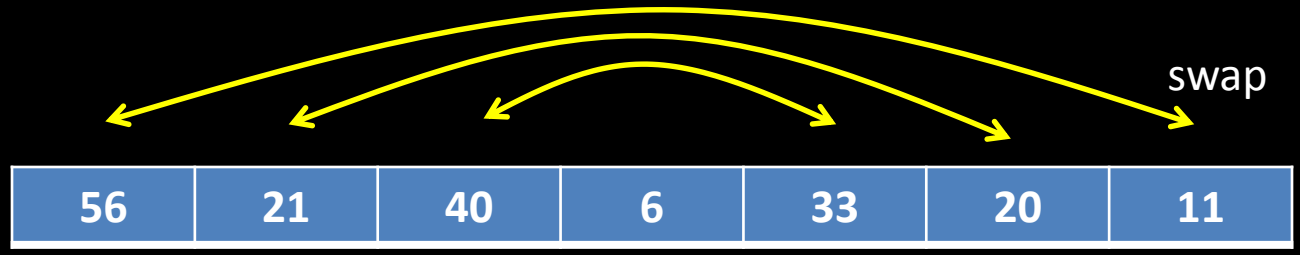
Generic Classes

```
public class Pair<T, S>
{
    private T first;
    private S second;

    public Pair(T firstElement, S secondElement)
    {
        first = firstElement;
        second = secondElement;
    }
    public T getFirst() { return first; }
    public S getSecond() { return second; }
}
```

Generic Methods

Implementasikan static method generic **reverse**, yang mampu membalik urutan elemen dari array of anything.

Jangan membuat array baru ! Gunakan teknik berikut:



11	20	33	6	40	21	56

Generic Methods

```
String[] strs = {"abc", "ab", "a"};
Double[] dbls = {1.0, 2.0, 3.0, 4.0};

reverse(strs);
// strs -> a, ab, abc

reverse(dbls);
// dbls -> 4.0, 3.0, 2.0, 1.0
```