Praktikum 2
Pengenalan OOP di Java

A. Tujuan

1. Mampu memahami dan mengimplementasikan Class dan Objek
2. Mampu memahami dan mengimplementasikan Encapsulation dan Construktor
3. Mampu memahami dan mengimplementasikan Inheritance dan Polymorfisme

B. Dasar Teori

Pemrograman Berorientasi Objek (Object Oriented Programming/OOP) merupakan pemrograman yang berorientasikan kepada objek, dimana semua data dan fungsi dibungkus dalam class-class atau object-object. Setiap object dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Beberapa object berinteraksi dengan saling memberikan informasi satu terhadap yang lainnya.

Masing-masing object harus berisikan informasi mengenai dirinya sendiri dan dapat dihubungkan dengan Object yang lain. Pemrograman berorientasi objek berbeda dengan pemrograman prosedural yang hanya menggunakan satu halaman kebawah untuk mengerjakan banyak perintah atau statement. Penggunaan pemrograman berorientasi objek sangat benyak sekali, contoh : java, php, perl, c#, cobol, dan lainnya.

PBO atau OOP adalah konsep pemrograman yang difokuskan pada penciptaan **kelas** yang merupakan abstraksi/blueprint/prototype/ tempat dari suatu **objek.** Kelas ini harus mengandung sifat (data)/ variabel dan tingkah laku (method)/fungsi umum yang dimiliki oleh objek-objek yang nantinya akan dibuat (diinstansiasi).

CLASS

Suatu template / blueprint / rancangan dari objek yang akan dibuat untuk menggambarkan ciri-ciri objek secara umum. Class dan objek adalah dua aspek utama dari OOP.

CLASS	OBJEK
Buah	Nanas
	Apel
	Jeruk
CLASS	OBJEK
Mobil	Toyota
	Honda
	Daihatsu

OBYEK

benda, baik yang berwujud nyata, maupun yang tidak nyata (unit terkecil dari pemrograman yang masih memiliki data dan fungsi).

Dalam konsep Pemrograman Berorientasi Objek dikenal beberapa istilah umum, yaitu:

C. Percobaan

1. Objek dan Class

- **Menciptakan sebuah objek**

```
public class Percobaan2 {
   int nilai = 8;

    public static void main(String[] args) {
        Percobaan2 obj = new Percobaan2();
        System.out.println(obj.nilai);
    }

}
```

- **Mengubah Atribut**

```
public class Percobaan2 {
   int nilai = 8;

    public static void main(String[] args) {
        Percobaan2 obj = new Percobaan2();
        System.out.println("Nilai pertama :"+obj.nilai);
        obj.nilai = 99;
        System.out.println("Nilai kedua :"+obj.nilai);
    }

}
```

- **Multiple atribut**

```
public class Percobaan2 {
   String nrp = "720999";
   String nama = "Norma Ningsih";
   int umur = 30;

    public static void main(String[] args) {
        Percobaan2 obj = new Percobaan2();
        System.out.println("NRP :"+obj.nrp);
        System.out.println("nama :"+obj.nama);
        System.out.println("nama :"+obj.umur);
    }
}
```

- **Akses atribut dan method dari kelas berbeda**

```
public class Mobil {
    String warna;
    int tahunProduksi;

    public void brand(){
        System.out.println("Mobil ini dari Toyota");
    }
    public void price(int harga){
        System.out.println("Harga mobil :"+harga);
    }

    }

class Utama {
    public static void main(String[] args){
       Mobil objMo = new Mobil();
       objMo.brand();
       objMo.price(100000);
    }
}
```

2. Encapsulation dan Construktor

Enkapsulasi adalah mekanisme untuk membungkus data (variable) dan method bersama -sama menjadi satu. Enkapsulasi memastikan bahwa data "sensitif" disembunyikan dari pengguna. Ketentuan untuk membuat enkapsulasi :

1. Mendeklarasi variable atau atribut class menjadi private
2. Menyediakan method get dan set untuk mengakses dan memperbarui nilai variable private

```
import java.io.*;
public class Siswa {
   private String nama;
   private String jurusan;
   private String nim;

 public void setNama(String nama){
  this.nama =nama;
 }

  public void setJurusan(String jurusan){
   this.jurusan=jurusan;
 }

   public void setNim(String nim){
    this.nim=nim;
 }

   public String getNama(){
     return nama;
     }

     public String getJurusan(){
     return jurusan;
     }

       public String getNim(){
     return nim;
     }
}
```

```
class siswaLagi  {
  public static void main (String[] args) throws IOException{
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));

    Siswa sw = new Siswa();

    System.out.println("nama siswa adalah :");
    String nama = br.readLine();
    System.out.println("Jurusan siswa adalah :");
    String jurusan = br.readLine();
    System.out.println("nim siswa adalah :");
    String nim = br.readLine();

    sw.setNama(nama);
    sw.setJurusan(jurusan);
    sw.setNim(nim);

    System.out.println("nama siswa adalah :"+sw.getNama()+
     "\nJurusan Siswa adalah :"+sw.getJurusan()+
     "\nnim siswa adalah :"+sw.getNim());
  }
}
```

Construktor

Constructor adalah sebuah metode yang dapat digunakan untuk memberikan nilai awal saat objek diciptakan. Metode ini dipanggil secara otomatis oleh Java ketika new dipakai untuk menciptakan instan kelas.

Sifat konstruktor :

A. —Namanya sama dengan nama kelas
B. —Bisa ditambahkan modifier akses : public
C. —Tidak memiliki nilai balik return (termasuk tidak boleh ada kata-kunci void)
D. —Pada satu kelas dapat memiliki lebih dari satu constructor / overloading

```
class Sepatu {
    private String warna;
    private int nomorSepatu;

    /*Konstruktor, sifat :
    a. Namanya sama dengan nama kelas
    b. Tidak memiliki nilai balik (termasuk tidak boleh ada kata kunci void)
     */
    public Sepatu(String warna) {
        this.warna = warna;
    }
    public Sepatu(String warna, int nomorSepatu) {
        this.warna = warna;
        this.nomorSepatu = nomorSepatu;
    }

    //Metode
    public void info() {
        System.out.println("Warna : " + this.warna);
        System.out.println("Nomor Sepatu : " + this.nomorSepatu);
    }
}
```

Punya 2 parameter/variabel instan

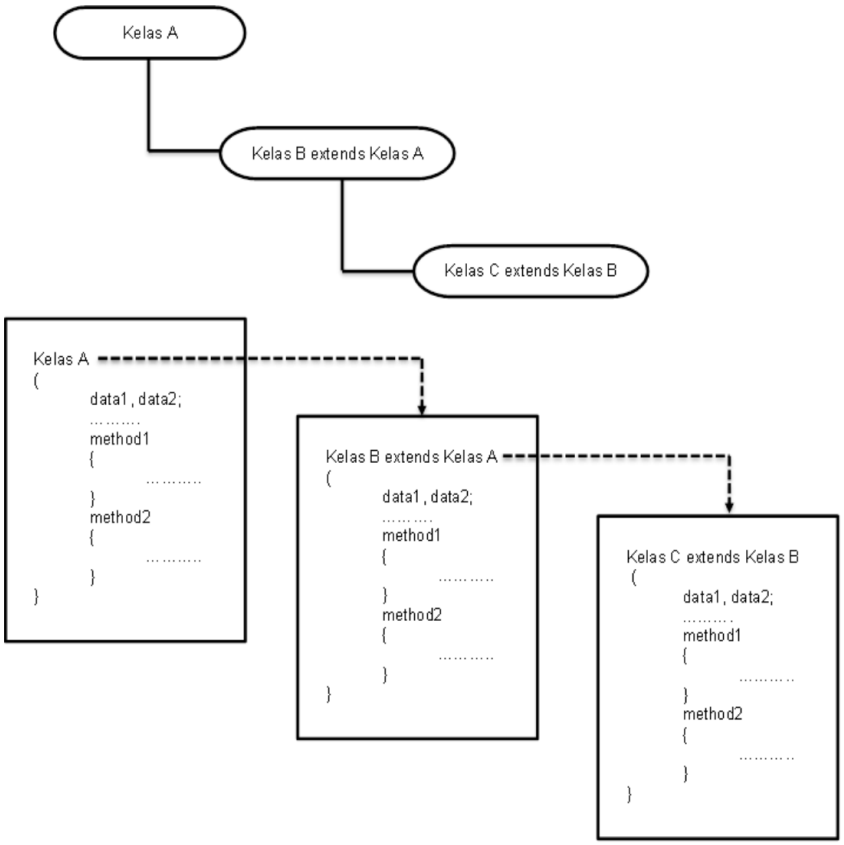
Nama konstruktor = Nama class

Variabel instan dapat diisikan sebagai argumen

```
public class Konstruktor {
    public static void main (String[] args) {
        Sepatu sepatuku1 = new Sepatu("Merah");
        sepatuku1.info();
        Sepatu sepatuku2 = new Sepatu("Merah", 36);
        sepatuku2.info();
    }
}
```

3. Inheritance

- Pewarisan merupakan konsep dalam PBO yang memungkinkan untuk membuat suatu kelas dengan didasarkan pada kelas yang sudah ada, sehingga mewarisi semua metode dan variabelnya.
- Tidak perlu menuliskan kode dari nol.
- Semua metode dan variabel instan yang terdapat pada kelas dasar diturunkan ke kelas turunan.
- Namun, kelas turunan dapat menambahkan metode baru atau variabel instan baru tersendiri.

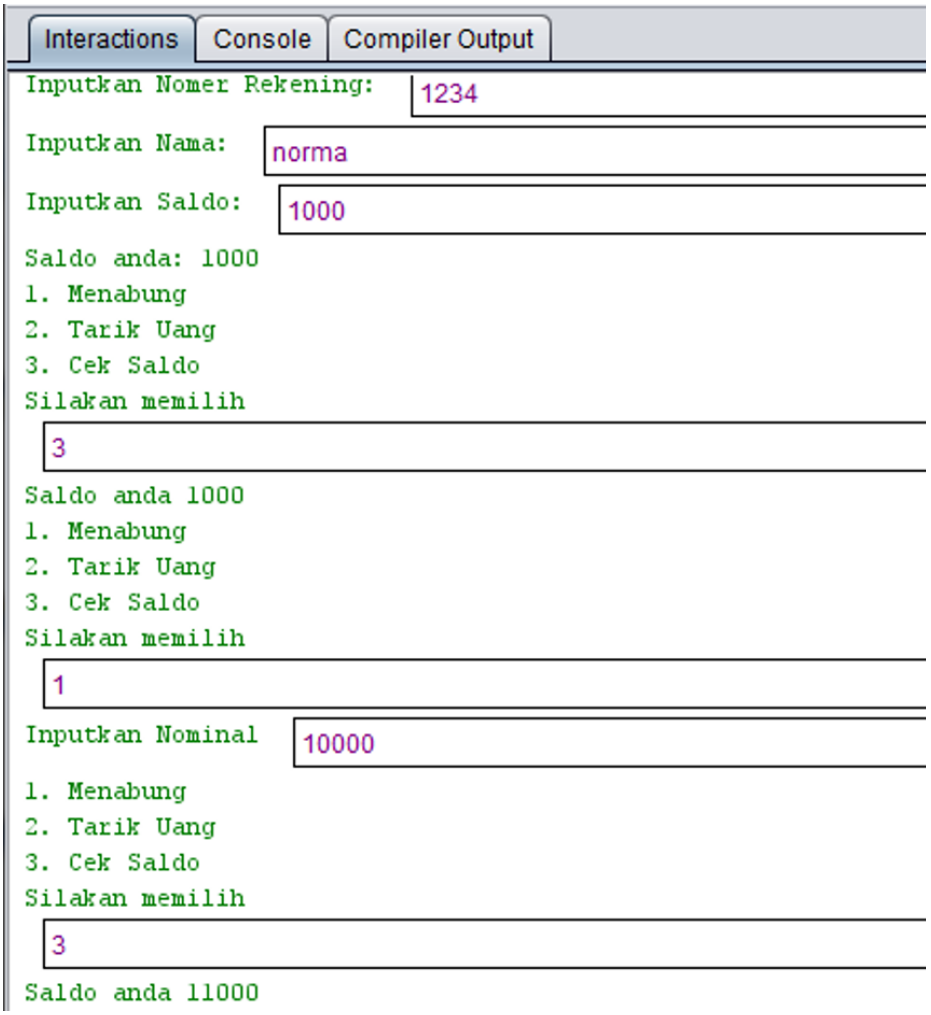



```
class Orang {
   private String nama;
   private int usia;

   //Konstruktor
   public Orang(String nama, int usia) {
      this.nama = nama;
      this.usia = usia;
   }
   //Metode
   public void info() {
      System.out.println("Nama : " + this.nama);
      System.out.println("Usia : " + this.usia);
   }
}
```

```
class Pegawai extends Orang {
   protected String noPegawai;
   //Konstruktor
   public Pegawai(String noPegawai, String nama, int usia) {
      super(nama, usia);
      this.noPegawai = noPegawai;
   }
   //Metode
   public void info2() {
      System.out.println("No. Pegawai : " + this.noPegawai);
      super.info();
   }
}

public class MainPegawai {
   public static void main (String[] args) {
      Pegawai p1 = new Pegawai("101", "Edi", 25);
      p1.info2();
   }
}
```

D. Tugas

1. Gunakan konsep enkapsulasi dengan menggunakan method get dan get untuk kasus dibawah ini :



2. Gunakan konsep inheritance untuk kasus berikut ini :
   - Buat class MatematikaCanggih yang merupakan inherit dari class Matematika
     a. Tambahkan method modulus(int a, int b) yang menghitung modulus dari a dan b
     b. Operator modulus adalah %
   - Buat class MatematikaCanggihBeraksi yang memanggil method pertambahan, perkalian dan modulus