POLYMORPHISM, ABSTRACT CLASS & INTERFACE

Polymorph yang ada di Java programming sebenarnya mengambil ide dari konsep biologi, dimana sebuah spesies atau organisme dapat memiliki banyak **bentuk** atau **langkah**. Selama anda praktikum, dimulai dari pengenalan apa itu objek, kelas, method, inheritance, dsb, anda sudah menerapkan *polymorphism*. Polymorphism dapat dicapai dalam 3 hal:

1. Melalui overloading method (di kelas itu sendiri)
2. Melalui overriding method lewat inheritance, dan
3. Melalui overriding method lewat interface.

Untuk lebih jelasnya dari masing-masing pendekatan di atas, ketikan beberapa kode program di bawah ini, amati *behaviour* nya selama dijalankan (note: disarankan melalui proses *debugging*).

- Melalui overloading method (di kelas itu sendiri)

Instruksi: buat sebuah project baru dengan nama **Polymorphism**, kemudian re-name nama package menjadi **del.ac.id.main**, lalu di dalam package ini buat sebuah kelas dengan nama **Car.java**

```
6   package del.ac.id.main;
7
8   /**
9    *
10   * @author teams
11   */
12  public class Car {
13      private static final float BY_ENGINE_BRAKE = 10; // dengan turun kopling
14      private static final float BY_FRICTION_BRAKE = 50; // dengan rem biasa
15      private float currentSpeed;
16
17      public Car(float currentSpeed){
18          this.currentSpeed = currentSpeed;
19      }
20
21      // jika seseorang memperlambat mobil
22      // by default (lepas gas mobil)
23      public float decreaseSpeed(){
24          return this.currentSpeed -= 5;
25      }
26
27      public float decreaseSpeed(String brakeMechanism){
29          if(brakeMechanism.toLowerCase().equals("engine")){
30              return this.currentSpeed -= BY_ENGINE_BRAKE;
31          }else if(brakeMechanism.toLowerCase().equals("friction")){
32              return this.currentSpeed -= BY_FRICTION_BRAKE;
33          }else{
34              return decreaseSpeed();
35          }
36      }
37  }
```

Penjelasan: pada kode program di atas, sebuah mobil memiliki 3 mekanisme memperlambat kendaraan. Pertama, jika seseorang ingin memperlambat mobil dengan tidak terburu-buru, maka dapat melepas gas mobil. Mobil akan

diperlambat sebanyak 5 km/jam. Akan tetapi, bila seseorang sudah memprediksi bahwa ada benda besar di depan mobilnya, dan membutuhkan penurunan kecepatan, maka orang tersebut dapat memilih memakai mekanisme pengereman manual (engine brake atau friction brake).
Polymorphism melalui pendekatan pertama (overloading method) terlihat jelas pada contoh di atas, yaitu method **decreaseSpeed**, ada yang pakai parameter (pengereman manual) dan ada yang tidak pakai (pengurangan kecepatan dengan lepas gas). Ingate konsep *overloading*! Overloading akan tercapai apabila beberapa method memiliki nama method yang sama, **namun** parameternya berbeda (apakah jumlah parameter, maupun tipe data yang dipakai parameter).

Instruksi: buat sebuah file driver, simpan dengan nama **Polymorphism.java**

```
6   package del.ac.id.main;
7
8   /**
9    *
10   * @author teams
11   */
12  public class Polymorphism {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
            Car objCar = new Car((float)150);
20          System.out.println(String.format("Kurangi kecepatan dengan lepas gas: %.2f km/h",
21                  objCar.decreaseSpeed()));
22          System.out.println(String.format("Kurangi kecepatan dengan engine brake: %.2f km/h",
23                  objCar.decreaseSpeed("engine")));
24          System.out.println(String.format("Kurangi kecepatan dengan friction brake: %.2f km/h",
25                  objCar.decreaseSpeed("friction")));
26      }
27
28  }
```

Penjelasan: kode driver di atas menunjukan penggunaan polymorphism. Lakukan *debugging* untuk mengamati perilakunya.

- Melalui overriding method lewat inheritance

Instruksi: buatlah file **Truck.java** yang meng-extend kelas **Car.java**

```
12  public class Truck extends Car {
13      private float speed;
14      // constructor di kelas Truck ini
15      // menerima sebuah parameter, yang nantinya
16      // akan "dilempar" nilainya ke kelas indukan
17      // melalui konstruktor
18      public Truck(float currentSpeed){
19          super(currentSpeed);
20      }
21
22      @Override
23      public float decreaseSpeed() {
24          speed = getCurrentSpeed() - (float)13.5;
25          return speed;
26      }
27
28      @Override
29      public float decreaseSpeed(String brakeMechanism) {
30          speed = getCurrentSpeed();
31          if(brakeMechanism.toLowerCase().equals("engine")){
32              return speed -= 20;
33          }else if(brakeMechanism.toLowerCase().equals("friction")){
34              return speed -= 10;
35          }else{
36              return decreaseSpeed();
37          }
38      }
39  }
```

Penjelasan: dari contoh di atas kelas **Truck.java** secara tidak langsung sudah melakukan *polymorphism* melalui inheritance. Terbukti method **decreaseSpeed** dari kelas indukan **Car.java** telah di-*override* di kelas anakan **Truck.java**

Instruksi: buatlah file backup untuk kelas **Car.java** simpan di dalam folder submission anda. Hal ini dikarenakan kelas ini akan dimodifikasi sedikit saja. Tambahkan sebuah method getter **getCurrentSpeed** pada kelas ini (**Car.java**), seperti kode di bawah:

```
21      public float getCurrentSpeed(){
22          return this.currentSpeed;
23      }
```

Instruksi: buatlah file backup untuk driver **Polymorpism.java** anda, simpan di dalam folder submission. Ubahlah beberapa potongan program di dalamnya seperti kode berikut:

```
12  public class Polymorphism {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
            Car objTruck = new Truck(120);
20          System.out.println(String.format("Kurangi kecepatan dengan lepas gas: %.2f km/h",
21                  objTruck.decreaseSpeed()));
22          System.out.println(String.format("Kurangi kecepatan dengan engine brake: %.2f km/h",
23                  objTruck.decreaseSpeed("engine")));
24          System.out.println(String.format("Kurangi kecepatan dengan friction brake: %.2f km/h",
                    objTruck.decreaseSpeed("friction")));
26      }
27
28  }
```

Instruksi: jalankan kode program di atas, amati perilaku dan hasil keluarannya. Apakah hasilnya berbeda dengan sebelumnya (contoh pertama)? Bila **ya**, coba anda cari tahu apa penyebab perbedaan tersebut! Buat laporan anda.

- Melalui overriding method lewat interface

Instruksi: buat sebuah interface lalu simpan dengan nama **IVehicle.java**

```
    public interface IVehicle {
        float decreaseSpeed();
        float decreaseSpeed(String brakeMechanism);
15  }
16
```

Instruksi: buat file backup **Car.java** untuk contoh yang kedua di atas, simpan di dalam folder submission anda. Ubahlah sedikit kode program seperti di bawah ini.

```
    public class Car implements IVehicle{
13      private static final float BY_ENGINE_BRAKE = 10; // dengan turun kopling
14      private static final float BY_FRICTION_BRAKE = 50; // dengan rem biasa
15      private float currentSpeed;
16
17      public Car(float currentSpeed){
18          this.currentSpeed = currentSpeed;
19      }
20
21      public float getCurrentSpeed(){
22          return this.currentSpeed;
23      }
24
25      // jika seseorang memperlambat mobil
26      // by default (lepas gas mobil)
27      @Override
        public float decreaseSpeed(){
29          return this.currentSpeed -= 5;
30      }
31
32      @Override
        public float decreaseSpeed(String brakeMechanism){
34          switch (brakeMechanism.toLowerCase()) {
35              case "engine":
36                  return this.currentSpeed -= BY_ENGINE_BRAKE;
37              case "friction":
38                  return this.currentSpeed -= BY_FRICTION_BRAKE;
39              default:
40                  return decreaseSpeed();
41          }
42      }
```

Penjelasan: kode program di atas menunjukan bagaimana melakukan *polymorphism* melalui interface. Ketika anda membuat sebuah interface, maka kelas yang mengimplementnya harus melakukan *overriding*, hal ini ditandai adanya *flag* **@Override** seperti contoh di atas.

EASIEST CHALLENGE!
Buatlah notasi class diagram nya untuk kasus 1 – 3 di atas. Gunakan visio, starUML, atau draw.io (online) bila perlu.

ABSTRACT CLASS
Abstract kelas "mirip-mirip" dengan interface. Sesuai dengan namanya, abstrak, kelas ini **boleh seluruhnya** atau **secara parsial** memiliki abstrak method di dalam kelas itu. Akan tetapi bila anda membuat seluruh abstrak method di dalam sebuah kelas abstrak, lebih baik dibuat menjadi interface. Ketika anda menggunakan sebuah abstrak kelas, anda harus menggunakan ***keyword* "extends"**, sama seperti inheritance (hanya keyword nya saja). Ingat! Kelas abstrak tidak boleh di-instansiasi, **namun** harus **diturunkan** seperti layaknya inheritance. Kapan anda memutuskan untuk memakai kelas abstrak daripada interface?

1. Jika anda ingin menurunkan **sebagian saja** method untuk di-share (turunkan) ke dalam kelas anakan, dan ini benar-benar *closely related* (sangat berhubungan dengan kelas anakan).
2. Hampir sama dengan penjelasan no.1 di atas, anda memerlukan abstrak kelas jika banyak kesamaan method property (data member) dengan kelas anakan. Anda juga dapat menggunakan beberapa akses modifier yang ada di dalam kelas abstrak, seperti protected dan private yang tidak diperbolehkan sama sekali di interface.

Kapan anda gunakan interface?

1. Ketika beberapa orang ingin menerapkan method-method yang ada di interface dengan "caranya masing-masing". Hal ini mengindikasikan kalau kelas yang mengimplement sebuah interface itu **kurang** *closely related*.
2. Ketika anda ingin memanfaatkan **multiple interfaces** melalui sebuah kelas yang mengimplementasikan.

Kasus sederhana untuk menunjukan abstrak kelas adalah, semua kendaraan dapat dikatakan **MovingObject**, dimana memiliki perilaku: punya gaya (F), punya percepatan (a), dan punya massa (m). Baik itu **Car**, **Truck**, maupun **Motorcycle**, pasti memiliki ketiga properti di atas. Akan tetapi, cara merakit **Car**, **Truck**, dan **Motorcycle** sangat berbeda. Hal ini dapat di implementasikan oleh caranya masing-masing. Yang menjadi fokusnya untuk abstrak kelas adalah semua state atau perilaku (termasuk data member yang **bukan private**) yang sama dapat langsung digunakan oleh kelas turunannya, selebihnya yang **hal yang berbeda** itu dapat diterapkan dimasing-masing kelas turunanya. Inilah letak keunggulan abstrak kelas yang tidak dimiliki oleh interface. Kesamaan antara interface dengan

abstrak kelas hanya dilihat dari "**hal yang berbeda**" itu saja. Dimana masing-masing sub-kelas harus mengimplementasikan dengan gaya masing-masing.

Ketikan kode program berikut untuk menambah pemahaman anda mengenai abstrak kelas.

Instruksi: buatlah sebuah kelas abstrak dengan nama **MovingObject.java** di dalam package yang ada sebelumnya (del.ac.id.main).

```
12    //abstract public class MovingObject // sama saja dengan dibawah
      public abstract class MovingObject {
14        private float beratKendaraan, percepatan;
15
16        public void setBeratKendaraan(float value){ beratKendaraan = value; }
17        public void setPercepata(float value){ percepatan = value; }
18        public float getBeratKendaraan(){ return beratKendaraan; }
19        public float getPercepatan(){ return percepatan; }
20
21        public float getGayaPergerakan(){
22            return getBeratKendaraan() * getPercepatan();
23        }
24
          abstract void caraMerakitObjekBergerak();
26    }
```

Penjelasan: pada kode program di atas menunjukan bagaimana membuat sebuah abstrak kelas. **Hal yang sama** yang dapat di-share ke kelas anakannya adalah "hanya" **getGayaPergerakan**, selebihnya **caraMerakitObjekBergerak** berbeda-beda untuk masing-masing jenisnya.

Instruksi: buatlah sebuah kelas **Motorcycle.java** yang meng-extend kelas abstrak di atas.

```
12    public class Motorcycle extends MovingObject {
13
14        @Override
          void caraMerakitObjekBergerak() {
16            System.out.print("Cara merakit sepeda motor adalah pasang dua buah roda");
17        }
18    }
```

Penjelasan: kode program di atas menunjukan bagaimana menurunkan sifat dari abstrak kelas ke kelas anakan. Disana terdapat method **caraMerakitObjekBergerak** yang telah di-*override*. Hal ini mirip dengan interface. Method lain, **getGayaPergerakan**, dan sebagainya, tidak perlu lagi dibuat, karena sifat ini ada dimasing-masing jenis kendaraan, baik itu car, truck, atau motorcycle.

Instruksi: buatlah backup file untuk **Car.java** yang telah ada sebelumnya karena akan diubah beberapa potongan code.

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package del.ac.id.main;

/**
```

```
 *
 * @author teams
 */
public class Car extends MovingObject implements IVehicle{
    private static final float BY_ENGINE_BRAKE = 10; // dengan turun
kopling
    private static final float BY_FRICTION_BRAKE = 50; // dengan rem
biasa
    private float currentSpeed;

    public Car(float currentSpeed){
        this.currentSpeed = currentSpeed;
    }

    public float getCurrentSpeed(){
        return this.currentSpeed;
    }

    @Override
    void caraMerakitObjekBergerak() {
        System.out.print("Cara  merakit  mobil  adalah  tambahkan  4
roda.\n");
        System.out.print("Selain itu tambahkan mesin 2000cc. \n");
    }

    // jika seseorang memperlambat mobil
    // by default (lepas gas mobil)
    @Override
    public float decreaseSpeed(){
        return this.currentSpeed -= 5;
    }

    @Override
    public float decreaseSpeed(String brakeMechanism){
        switch (brakeMechanism.toLowerCase()) {
            case "engine":
                return this.currentSpeed -= BY_ENGINE_BRAKE;
            case "friction":
                return this.currentSpeed -= BY_FRICTION_BRAKE;
            default:
                return decreaseSpeed();
        }
    }
}
```

Penjelasan: hampir sama dengan kelas **Motorcycle.java** semua kesamaan dapat langsung digunakan terkecuali method **caraMerakitObjekBergerak**.

Instruksi: backup dan ubahlah kode program driver **Polymorphism.java**

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package del.ac.id.main;

/**
 *
 * @author teams
```

```
 */
public class Polymorphism {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Car objCar = new Car(120);
        System.out.println(String.format("Kurangi kecepatan dengan
lepas gas: %.2f km/h",
                objCar.decreaseSpeed()));
        System.out.println(String.format("Kurangi kecepatan dengan
engine brake: %.2f km/h",
                objCar.decreaseSpeed("engine")));
        System.out.println(String.format("Kurangi kecepatan dengan
friction brake: %.2f km/h",
                objCar.decreaseSpeed("friction")));
        // invoking caraMerakitObjekBergerak dari instanse objCar
        objCar.caraMerakitObjekBergerak();
        // invoking method getGayaPergerakan dari instance objCar
        objCar.setBeratKendaraan(500);
        objCar.setPercepata(120);
        System.out.println(String.format("Gaya mobil anda adalah
%.2f newton", objCar.getGayaPergerakan()));

        MovingObject objMotorcycle = new Motorcycle();
        //   invoking   caraMerakitObjekBergerak   dari   instanse
objMotorcycle
        objMotorcycle.caraMerakitObjekBergerak();
        //   invoking   method   getGayaPergerakan   dari   instance
objMotorcycle
        objMotorcycle.setBeratKendaraan(100);
        objMotorcycle.setPercepata(80);
        System.out.println(String.format("\nGaya motor anda adalah
%.2f newton", objMotorcycle.getGayaPergerakan()));
    }

}
```

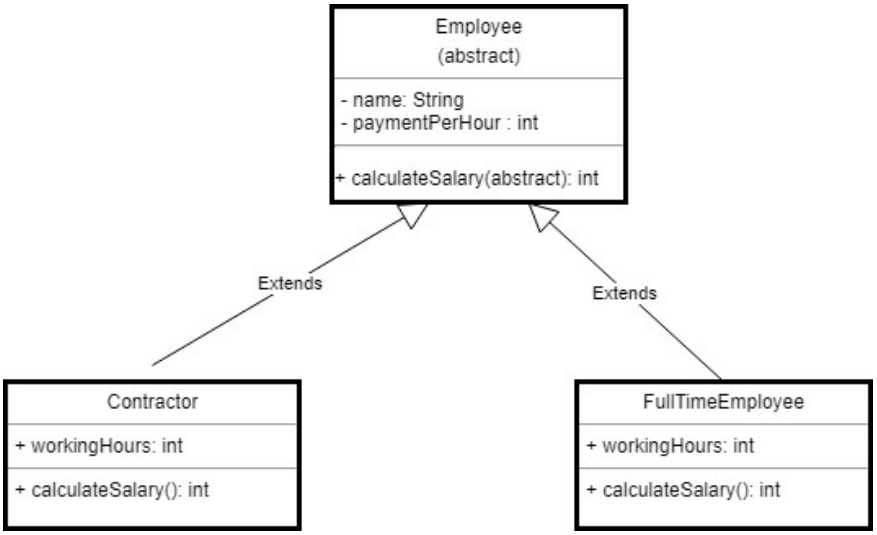
Penjelasan: disini menunjukan cara bagaimana anda melakukan invoking method dari sebuah kelas yang mana diturunkan sifatnya dari kelas abstrak.

EASIEST CHALLENGE!

Buatlah notasi kelas diagramnya untuk kasus terakhir ini. Gunakan, visio, starUML, atau draw.io (online).

TUGAS

1. Buatlah kode java untuk kelas diagram di bawah ini. (Note: cara mengimplementasikan body method dibebaskan, selama berbeda dengan rekan anda).



2. Sama dengan nomor 1, namun kelas diagramnya seperti di bawah: