VERSION 1.1

APRIL, 2022

PEMROGRAMAN BERORIENTASI OBJEK

ABSTRACT CLASS, INTERFACE, POLYMORPHISM – MODUL 4

DISUSUN OLEH :

- Muhammad Rahadian Arya Saputra
- Riyan Putra Firjatullah

DIAUDIT OLEH :

- Ali Sofyan Kholimi, M.Kom.

PRESENTED BY: TIM LAB-IT

UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN BEROIRENTASI OBJEK

TUJUAN

1. Mahasiswa memahami abstract class
2. Mahasiswa memahami konsep polymorphism
3. Mahasiswa memahami konsep interface

TARGET MODUL

1. Mahasiswa dapat membuat dan mengimplementasikan abstract class
2. Mahasiswa dapat mengimplementasikan konsep polymorphism

PERSIAPAN SOFTWARE/APLIKASI

1. Compiler java (JDK), JRE.
2. Editor Java (NetBeans, Gel, Eclipse, Jcreator, dll).

KEYWORDS

- Polymorphism
- Interface
- Abstract Class
- Method Overriding
- Abstract Method
- Has-a

PERSIAPAN MATERI

- **Abstract Method**

Abstract method merupakan sebuah method yang dideklarasikan dengan menambahkan keyword abstract pada deklarasinya, dan tanpa ada implementasi dari method tersebut. Dalam arti, hanya pendeklarasian saja, tanpa tanda sepasang kurung kurawal. Tetapi diakhiri dengan tanda titik koma(;)

Contoh :

```
public abstract int nameMethod(int var1, int var2);
```

- **Abstract Class**

Abstract class Adalah class yang sifatnya abstrak yang biasanya digunakan untuk mendefinisikan konsep class yang sifatnya umum. Sebagai contoh jika ada class hewan, kucing dan kelinci. karena hewan sifatnya sangat umum, maka bisa dideklarasikan sebagai abstract class. Dalam implementasinya, abstract class tidak bisa dibuat objectnya, namun bisa dimanfaatkan dalam konsep polimorfisme. Di dalam abstract class bisa dideklarasikan variabel class, concrete method dan abstract method.

Concrete method adalah method yang memiliki kode program / body program, artinya sudah diimplementasi. Sedangkan abstract method adalah method yang hanya berupa deklarasi method saja, yang terdiri dari nama method, jenis luaran, modifier dan input parameter tanpa kode program. Selanjutnya, untuk class yang merupakan turunan dari abstract class, jika itu adalah concrete class maka harus mengimplementasi abstract class (bentuknya seperti overriding), sedangkan jika absract class juga maka bisa tidak mengimplementasi abstract class.

```java
//abstract class
public abstract class Person {

    private String name;
    private String gender;

    public Person(String name, String gender){
        this.name = name;
        this.gender = gender;
    }

    //abstract method
    public abstract void work();

    @Override
    public String toString(){
        return "Name" + this.name+"::Gender="+this.gender;
    }

    public void changeName(String newName){
        this.name = newName;
    }
}
```

- **Method Overriding**

  Method Overriding adalah sebuah situasi dimana method class turunan menimpa method milik parent class. Ini bisa terjadi jika terdapat nama method yang sama baik di child class dan juga parent class. Method Overriding dapat dibuat dengan menambahkan anotasi **@Override** di atas nama method atau sebelum pembuatan method.

```java
package override;

class Senjata {
    String cekSenjata() {
        return "Ini berasal dari class Pistol";
    }
}

class Pistol extends Senjata {
    @Override
    String cekSenjata() {
        return "Ini berasal dari class Senjata";
    }
}

class BelajarOverriding {
    public static void main(String[] args) {

        Pistol pistol = new Pistol();
        System.out.println(pistol.cekSenjata());

    }
}
```

- **Interface**

Interface bukanlah class, meskipun deklarasinya menyerupai. Penamaan interface biasanya menggunakan kata sifat seperti Fireable, Rideable, Movable, dan lain-lain. Hal ini berhubungan dengan konsep interface yang digunakan untuk mendeklarasikan sifat-sifat tertentu. Sifat-sifat ini yang nantinya akan diimplementasi oleh class sesuai dengan kebutuhan. Sebagai contoh jika ada interface Fireable dan di dalamnya ada method shot() – menembak, Maka interface ini bisa diimplementasi oleh class Pistol, Tank, Bazooka, PesawatTemput yang memang memiliki kemampuan untuk menembak. Interface hanya memiliki abstract method saja, tidak ada yang concrete, sehingga pada saat menulis deklarasi method bisa tidak menggunakan keyword “abstract”, meskipun boleh juga menggunakan keyword “abstract”. Interface bisa juga memiliki variable, namun variable itu harus diberi nilai serta otomatis menjadi konstanta dengan modifier public, final dan static. Keyword yang digunakan untuk mengimplementasikan interface adalah “**implements**”.

```java
public interface ContohInterface {
    void tampil();
}
```

Untuk penjelasan lebih lanjut mengenai interface dapat dilihat disini

- **Polymorphism**

Polimorfisme merupakan kemampuan objek-objek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama. Polimarfisme juga dapat dikatakan kemampuan sebuah objek untuk memutuskan method mana yang akan diterapkan padanya, tergantung letak objek tersebut pada jenjang pewarisan.

```java
public class Employee extends Person{

    private int empId;

    public Employee(String name, String gender, int id){
        super(name, gender);
        this.empId = id;
    }

    @Override
    public void work(){
        if (empId == 0)
            System.out.println("Not working");
        else
            System.out.println("Working as employee");
    }
}
```

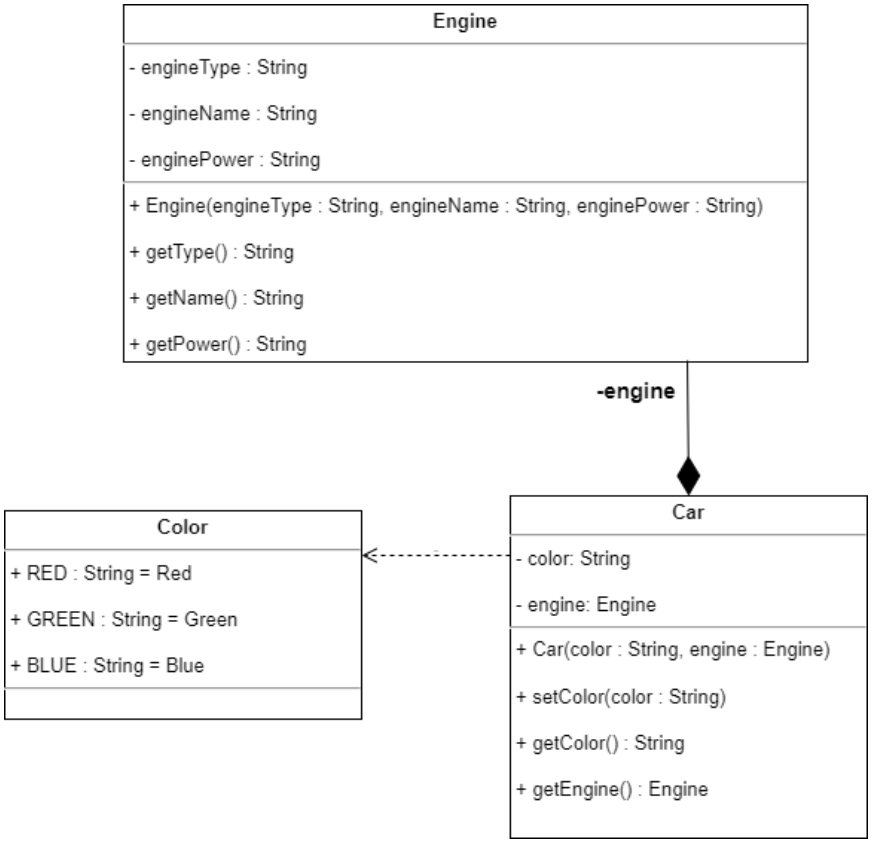
```
public class Main {
    public static void main(String[] args) {

        Person student = new Employee( name: "Davi", gender: "Female", id: 0);
        Person employee = new Employee( name: "Yuki", gender: "Male", id: 25);
        student.work();
        employee.work();

        employee.changeName( newName: "Yuki Kaji");
        System.out.println(employee.toString());

    }
}
```

Untuk penjelasan lebih lanjut tentang konsep polymorphism dapat dilihat disini

- **Konsep has-a**

Selain hubungan Is-A yang merepresentasikan pewarisan, satu lagi hubungan yang umum digunakan dalam pemrograman berorientasi objek. Hubungan ini disebut hubungan Has-A di Java. Ketika objek dari satu kelas dibuat sebagai anggota data di dalam kelas lain, itu disebut hubungan Has-A. Dengan kata lain, hubungan di mana objek dari satu kelas memiliki referensi ke objek dari kelas lain atau turunan lain dari kelas yang sama disebut hubungan Has-A di Java.

Berikut Ilustrasi sederhananya :



Contoh coding :

Buatlah class Car dan Engine

```
public class Engine {
    private String engineType;
    private String engineName;
    private String enginePower;

    public Engine(String engineType, String engineName, String enginePower){
        this.engineType = engineType;
        this.engineName = engineName;
        this.enginePower = enginePower;
    }

    public String getType() {
        return engineType;
    }

    public String getName() {
        return engineName;
    }

    public String getPower() {
        return enginePower;
    }

}
```

```java
public class Car {
    private String color;

    private Engine engine;

    public Car(String color, Engine engine) {
        this.color = color;
        this.engine = engine;

    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public Engine getEngine() {
        return engine;
    }
}
```

Lalu buatlah class Main

```java
public class Main {
    public static void main(String[] args) {
        Engine engine1 = new Engine( engineType: "V8",  engineName: "Chevy",  enginePower: "320 BHP");
        Car c = new Car( color: "RED", engine1);
        System.out.println("Engine Power : " +c.getEngine().getPower());
    }
}
```

Untuk lebih memahami tentang konsep has-a dapat dilihat disini

KEGIATAN PERCOBAAN

PERCOBAAN 1

Abstract

Buatlah class-class berikut dalam satu package

Class Abstract Hewan

```
package Abstract;

public abstract class Hewan {
    abstract void setNama();
    abstract void setMakanan();
}
```

Class Kucing

```
package Abstract;

public class Kucing extends Hewan{
    @Override
    void setNama() {
        System.out.println("Nama Hewan adalah kucing");
    }

    @Override
    void setMakanan() {
        System.out.println("Makanan kucing adalah Ikan");
    }

    void setWarna(){
        System.out.println("Warna putih");
    }
}
```

Class Sapi

```
package Abstract;

public class Sapi extends Hewan{
    @Override
    void setNama() {
        System.out.println("Nama hewan adalah sapi");
    }

    @Override
    void setMakanan() {
        System.out.println("Makanan sapi adalah rumput");
    }

    void setUkuran(){
        System.out.println("ukuran hewan besar");
    }
}
```

Class Main

```
package Abstract;

public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing();
        kucing.setNama();
        kucing.setMakanan();
        kucing.setWarna();

        Sapi sapi = new Sapi();
        sapi.setNama();
        sapi.setMakanan();
        sapi.setUkuran();
    }
}
```

PERCOBAAN 2

Interface

Buatlah class-class berikut dalam satu package

Interface Fireable

```
package Interface;

public interface Fireable {
    public void shot();
}
```

Interface Moveable

```
package Interface;

public interface Moveable {
    public void move();
}
```

Class Tank

```
package Interface;

public class Tank implements Fireable, Moveable{
    @Override
    public void shot() {
        System.out.println("Tank menembak");
    }

    @Override
    public void move() {
        System.out.println("Tank bergerak");
    }
}
```

Class Pistol

```
package Interface;

public class Pistol implements Fireable{
    @Override
    public void shot() {
        System.out.println("Pistol ditembakkan");
    }
}
```

Class Main

```
package Interface;

public class Main {
    public static void main(String[] args) {
        Moveable m = new Tank();
        Fireable f = new Pistol();

        m.move();
        f.shot();
        f = (Fireable) m;
        f.shot();
    }
}
```

PERCOBAAN 2

Polymorphism

Buatlah class berikut pada satu package

Class Militer

```
package oop_w_java_3_polymorphism_militer;

public class Militer {
    protected String unitName = "Militer";
    protected String weapon = "Senjata";
    protected int unitNumber;

    Militer(int unitNumber) {
        this.unitNumber = unitNumber;
    }

    public void report() {
        System.out.println("Unit Number " + this.unitNumber + " is a " + unitName);
    }

    public void attack() {
        System.out.println("Unit Number " + this.unitNumber + " menyerang dengan " + weapon);
    }
}
```

Class Archer

```
package oop_w_java_3_polymorphism_militer;

public class Archer extends Militer {

    Archer (int unitNumber) {
        super(unitNumber);
        super.unitName = "Archer";
        super.weapon = "Panah";
    }
}
```

Class Cavalry

```
package oop_w_java_3_polymorphism_militer;

public class Cavalry extends Militer {

    Cavalry(int unitNumber) {
        super(unitNumber);
        super.unitName = "Cavalry";
        super.weapon = "Pedang";
    }

    public void attack() {
        super.attack();
        System.out.println(" sambil memacu kudanya");
    }
}
```

Class Swordsman

```
package oop_w_java_3_polymorphism_militer;

public class Swordsman extends Militer {

    Swordsman (int unitNumber) {
        super(unitNumber);
        super.unitName = "Swordsman";
        super.weapon = "Pedang";
    }

}
```

Class Main

```java
package oop_w_java_3_polymorphism_militer;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        int i=0, numOfMiliter=0;
        Random random = new Random();
        Militer[] militer = new Militer[10];

        // Dengan menggunakan inheritence, kita tahu bahwa
        // Swordsman, Archer, dan Cavalri adalah Militer
        // Jadi tidak perlu dibuatkan variabel sendiri untuk setiap class, walaupun boleh
        for (i=0;i<10; i++) {
            switch(random.nextInt( bound: 3)) {
                case 0:
                    militer[i] = new Swordsman(++numOfMiliter);
                    break;
                case 1:
                    militer[i] = new Archer(++numOfMiliter);
                    break;
                case 2:
                    militer[i] = new Cavalry(++numOfMiliter);
                    break;
            }
        }
        for (i=0;i<10; i++) {
            militer[i].report();
        }
        for (i=0;i<10; i++) {
            militer[i].attack();
        }
    }
}
```

TUGAS

KEGIATAN 1

Buatlah 1 **Abstract Class BangunRuang** yang di dalam class tersebut terdapat 2 **abstract method** yaitu **getLuasPermukaan** dan **getVolume**. Kemudian turunkan dua method ini kedalam **class Kerucut** dan **Bola** (Harap diperhatikan bahwa masing-masing bangun ruang menerapkan rumus matermatika tersendiri). Kemudian buatlah 1 driver class untuk outputannya.

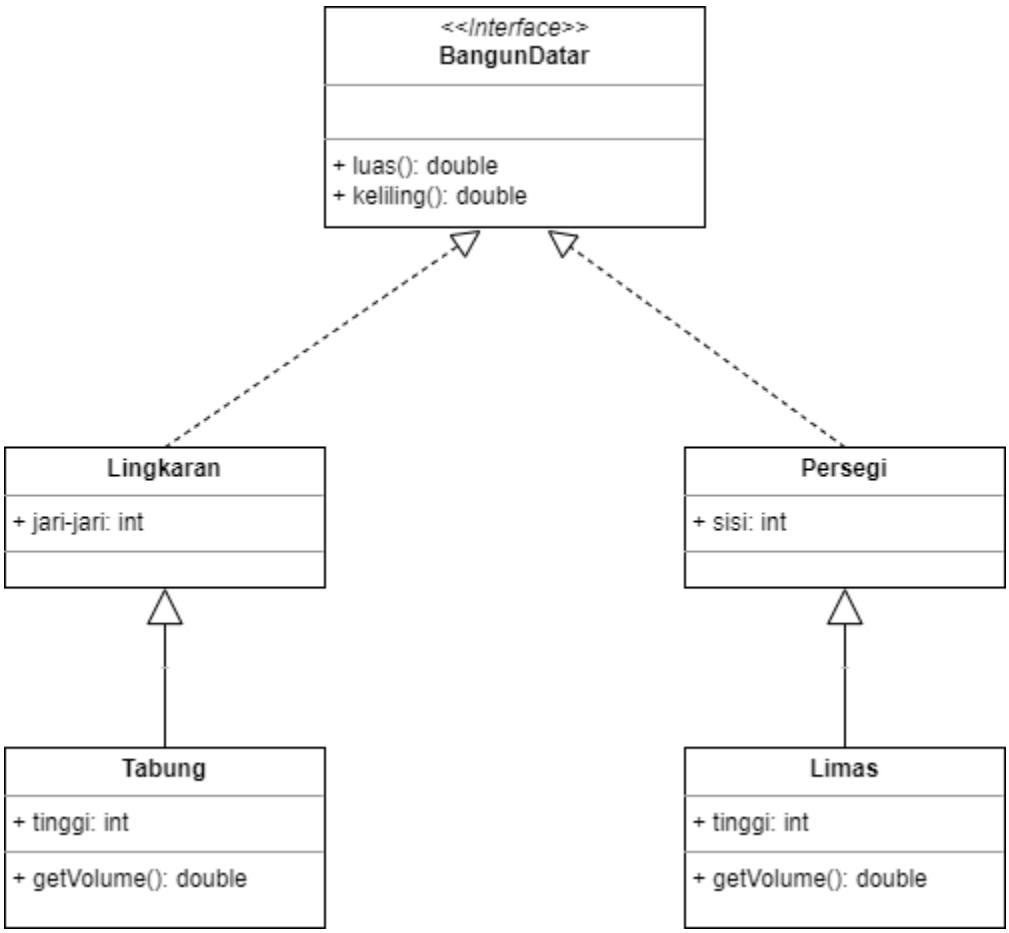
KEGIATAN 2

Buatlah minimal 2 buah class yang mana diantara kedua class itu menerapkan konsep **has-a** . Lalu buatlah driver class untuk outputnya. (Untuk penamaan classnya bebas).

KEGIATAN 3

Pilih salah satu dari soal di bawah ini. Nilai menyesuaikan bobot soal.

- **Soal A (Easy)**

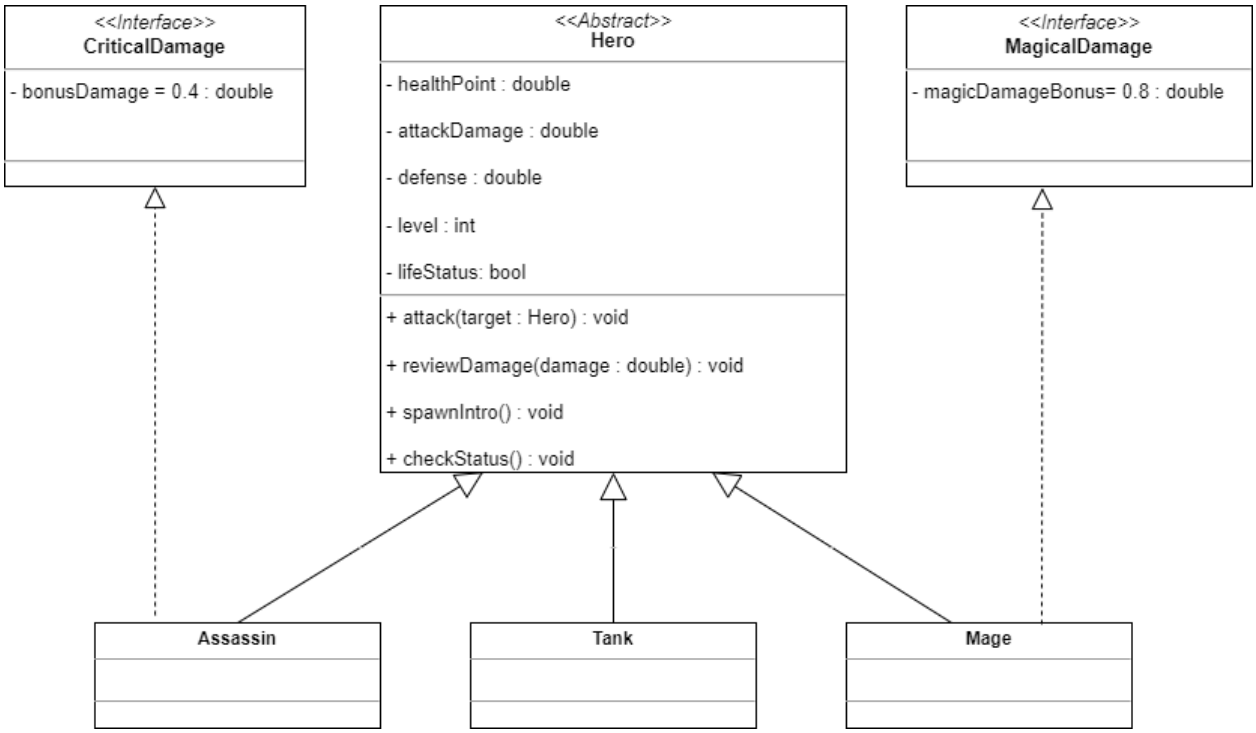


Buatlah sebuah kerangka program penghitungan keliling, luas dan volume sesuai dengan gambar yang telah diberikan di atas dengan syarat :

1. Class Lingkaran dan Persegi mengimplementasikan method yang ada pada interface BangunDatar
2. Class Tabung merupakan subclass dari class Lingkaran, sedangkan class Limas merupakan subclass dari class Persegi

Tambahkan sebuah class Driver untuk mengimplementasikan kerangka program diatas

- **Soal B (Hard)**



Buatlah sebuah fighting game sederhana sesual dengan class diagram diatas dengan syarat dan ketentuan :

- Character memiliki healthPoint, attackDamage, defense, level, dan lifeStatus
  - healthPoint : status yang menunjukkan jumlah satuan Kesehatan karakter game
  - attackDamage : nilai serangan yang akan diberikan untuk mengurangi healthpoint lawan
  - defense : ketahanan tubuh hero. Digunakan ketika mendapat serangan dari lawan
  - level : tingkat kemahiran/ kehebatan hero
  - lifeStatus : status hero (hidup/ mati)

- Method **attack(), reviewDamage(),** dan **checkStatus()** merupakan method concrete, sedangkan **spawnIntro()** merupakan **abstract method**.
- Method **attack(target : Hero)** : menyerang target
- Method **reviewDamage(damage : double)** : mengurangi healthPoint sebesar damage yang telah dikurangi defense atau dengan kata lain bisa disebut **realDamage**
    - Rumus untuk realDamage : **realDamage = attackDamage – defense**
    - healthPoint tidak boleh kurang dari 0 dan jika healthpoint menyentuh nilai 0 maka lifeStatus akan berubah menjadi false atau Hero tersebut telah mati dan pertandingan selesai
- Method **spawnIntro()** : dialog awal Hero ketika telah memasuki giliran Hero tersebut.
    - Contoh : Hero Tank : System.out.println(“Kill me if u can !!”);
- Method **checkStatus()** : menghasilkan data-data atribut Hero seperti sisa healthPoint
- Spesifikasi dasar Hero adalah lifeStatus : true dan penyesuaian tiap jenis Hero sebagai berikut :
    - Assassin :
        - healthPoint : 3000
        - defense : 300
        - attackDamage : 800
    - Tank :
        - healthPoint : 7000
        - defense : 500
        - attackDamage : 500
    - Mage :
        - healthPoint : 2500
        - defense : 200
        - attackDamage : 700
- Tiap jenis Hero memiliki constructor dengan 1 parameter (level : int) yang digunakan untuk menginialisasi nilai-nilai atribut.
- level pada parameter constructor digunakan untuk menambahkan nilai atribut Hero jika level > 0, dengan nilai tiap 1 levelnya :
    - Asassin :
        - (+90 healthPoint)
        - (+15 defense)

    - (+30 attackDamage)
- Tank :
    - (+200 healthPoint)
    - (+15 defense)
    - (+20 attackDamage)
- Mage :
    - (+85 healthPoint)
    - (+10 defense)
    - (+35 attackDamage)

- Untuk Hero Asassin, mendapatkan Critical Damage atau tambahan attackPoint selain dari level. Pada Constructor menambahkan attackDamage sebesar attackDamage * bonusDamage(0.4) pada interface CriticalDamage, dapat juga dituliskan seperti :
    - attackDamage += attackDamage * bonusDamage
- Untuk Hero Mage, mendapatkan Magical Damage atau tambahan attackPoint selain dari level. Pada Constructor menambahkan attackDamage sebesar attackDamage * magicDamageBonus(0.8) pada interface MagicalDamage, dapat juga dituliskan seperti :
    - attackDamage += attackDamage * magicalDamageBonus
- Boleh menambahkan getter dan setter jika perlu.

Tambahkan sebuah class Driver untuk mengimplementasikan objek game yang telah dibuat. Game tersebut nantinya akan berlangsung dengan mode 1v1 dengan giliran attack bergantian dan akan berhenti ketika salah satu Hero mati. Hero yang masih hidup adalah pemenang. Contoh player 1 (Assassin lvl. 8) vs player 2(Mage lvl. 5) outputnya sebagai berikut :

Output :

===Player 1===

Level : 8

Attack damage : 1360.0 Life status : true

Health poin : 3720.0 Defense : 420.0

===Player 2===

Level : 5

Attack damage : 1435.0 Life status : true

Health poin : 2925.0 Defense : 250.0

----------FIGHT BEGIN----------

-----ROUND 1------

---Player 1 TURN---

'Unseen and unheard'

Player 2 HP remaining : 1815.0

---Player 2 TURN---

'Behold the power of my magic'

Player 1 HP remaining : 2705.0

-----ROUND 2------

---Player 1 TURN---

'Unseen and unheard'

Player 2 HP remaining : 705.0

---Player 2 TURN---

'Behold the power of my magic'

Player 1 HP remaining : 1690.0

-----ROUND 3------

---Player 1 TURN---

'Unseen and unheard'

Player 2 HP remaining : 0.0

---Player 2 TURN---

'Mage is dead'

Player 1 HP remaining : 1690.0

Player 1 WIN the game

Catatan : Jika ada source code identik akan ada pengurangan nilai

RUBRIK PENILAIAN

Aspek Penilaian	Poin
Kegiatan 1	10
Kegiatan 2	10
Kegiatan 3 Soal A	20
Kegiatan 3 Soal B	40
Pemahaman	40
Total	**80 |100**