UJIAN AKHIR SEMESTER BERBASIS PROYEK MATA KULIAH PEMROGRAMAN BERORIENTASI OBJEK (PBO)

DOSEN PENGAMPU : SAYEKTI HARITS SURYAWAN, S.Kom., M.Kom.

Oleh :

NUR DILA YUANTI	(2211102441163)
ANNISA SYIFA AULIYA	(2211102441169)
RANI TRI SETIAWATI	(2211102441191)

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR

DESEMBER 2023

BAB I
PENDAHULUAN

A. Latar Belakang

Pemrograman berbasis objek adalah paradigma pemrograman yang berfokus pada konsep objek sebagai unit dasar dari pemrograman. Dalam konteks ini, kita akan mengeksplorasi pemrograman berbasis objek dengan menggunakan platform pengembangan permainan edukatif bernama Greenfoot. Greenfoot adalah lingkungan pengembangan yang dirancang untuk membantu pemula memahami konsep pemrograman berbasis objek melalui pembuatan permainan sederhana.

Laporan ini bertujuan untuk memberikan pemahaman yang mendalam tentang konsep pemrograman berbasis objek dengan menggunakan platform pengembangan permainan edukatif yang disebut Greenfoot. Konsep-konsep dasar seperti enkapsulasi, pewarisan, dan polimorfisme akan dijelaskan secara terperinci, sambil memberikan panduan langkah-demi-langkah tentang cara mengimplementasikannya menggunakan Greenfoot. Dalam laporan ini, pembaca akan diajak untuk memahami fitur-fitur utama Greenfoot dan bagaimana platform ini dapat menjadi alat yang efektif untuk pembelajaran pemrograman berbasis objek.

BAB II

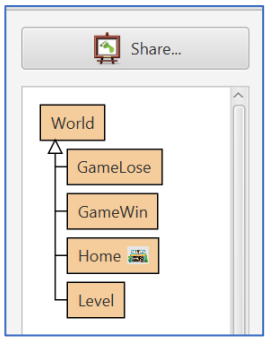
PEMBAHASAN

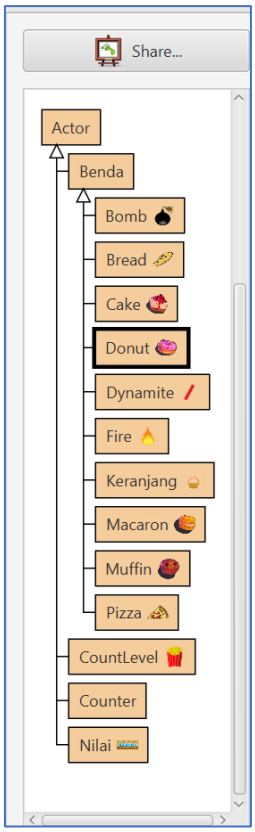
A. Game dengan Konsep Pemrograman Berorientasi Objek (PBO)/OOP

a. Kelas dan Objek

- Kelas (Class)

  Kelas adalah sebuah blueprint atau cetak biru yang mendefinisikan struktur dan perilaku objek. Kelas menciptakan suatu tipe data baru dengan properti (atribut) dan metode (fungsi) tertentu.





- Objek

  Objek adalah instance (salinan konkret) dari suatu kelas. Setiap objek memiliki atribut yang unik dan dapat menjalankan metode yang didefinisikan oleh kelasnya.

  Berikut adalah objek yang terdapat pada game:

  1. **level** (dalam kelas **Home**)
  2. **sound1** (dalam kelas **Home**)
  3. **imgWin** (dalam kelas **GameWin**)
  4. **imgLose** (dalam kelas **GameLose**)
  5. **cont, win, lose** (dalam kelas **Level**)
  6. **images** (dalam kelas **Leve**l)
  7. **keranjang, nilai** (dalam metode **isi**() di kelas **Level**)
  8. Berbagai objek dari kelas turunan **Benda** seperti **Bread**, **Cake**, **Donut**, dll.

  Catatan : Bentuk implementasi dapat dilihat pada gambar yang ada pada kelas di atas.

b. Pewarisan (Inheritance)

Pewarisan atau inheritance memungkinkan kelas yang satu (kelas turunan atau subclass) untuk mewarisi properti dan metode dari kelas yang lain (kelas induk atau superclass). Dengan kata lain, kelas turunan dapat mengakses dan menggunakan atribut dan perilaku yang telah didefinisikan oleh kelas induk.

Berikut ini adalah kelas-kelas yang merupakan contoh pewarisan:

1. Benda :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Benda extends Actor {
```

Kelas '**Benda**' merupakan kelas anak (subclass) dari kelas '**Actor**'.

2. Bomb :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bomb extends Benda {
```

Kelas '**Bomb**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

3. Bread :

```
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bread extends Benda {
```

Kelas '**Bread**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

4. Cake :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Cake extends Benda {
```

Kelas '**Cake**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

5. Donut :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Donut extends Benda {
```

Kelas '**Donut**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

6. Dynamite :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Dynamite extends Benda {
```

Kelas '**Dynamite**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

7. Fire :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Fire extends Benda {
```

Kelas '**Fire**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

8. Keranjang :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Keranjang extends Benda {
```

Kelas '**Keranjang**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

9. Macaron :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Macaron extends Benda {
```

Kelas '**Macaron**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

10. Muffin :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Muffin extends Benda {
```

Kelas '**Muffin**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

11. Pizza :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Pizza extends Benda {
```

Kelas '**Pizza**' merupakan kelas anak (subclass) dari kelas '**Benda**'.

Dengan demikian, kelas-kelas di atas menunjukan penggunaan pewarisan di mana setiap objek didalam game yang jatuh dari atas(seperti '**Bomb**', '**Bear**','**Cake**', dll.) adalah turunan dari kelas '**Benda**'.

c. Polimorfisme

Polimorfisme memungkinkan suatu objek untuk menunjukkan perilaku yang berbeda tergantung pada konteks penggunaannya. Hal ini dapat mencakup kemampuan suatu metode atau fungsi untuk memiliki implementasi yang berbeda di berbagai kelas yang terkait.

Contoh polimorfisme dalam kode tersebut adalah melalui metode '**act()**' yang di implementasikan di beberapa kelas seperti '**Bomb**',

‘**Bread**’,’**Cake**’,’**Donut**’,’**Dynamite**’,’**Fire**’,’**Keranjang**’,’**Macaron**’, dan ‘**Muffin**’. Semua kelas ini meng-override metode ‘**act()**’ dari kelas induk ‘**Actor**’.

Ketika objek dari kelas-kelas ini dipanggil dalam lingkungan game, Greenfoot akan memanggil metode act() yang sesuai dengan kelas objek tersebut. Hal ini menciptakan polimorfisme karena meskipun kita menggunakan metode yang sama (act()), implementasinya berbeda untuk setiap kelas.

Contoh :

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bomb extends Benda {
    public void act() {
        setLocation(getX(), getY()+4);
        meledak();
    }
```

Dengan demikian, melalui metode act() yang di-override pada kelas-kelas turunan, kita mencapai polimorfisme di mana setiap objek dapat berperilaku sesuai dengan implementasi kelasnya sendiri ketika metode act() dipanggil.

d. Enkapsulasi

Enkapsulasi melibatkan penyembunyian detail internal suatu objek, sehingga hanya metode atau antarmuka yang ditentukan yang dapat diakses dari luar objek. Atribut objek dienkapsulasi untuk mencegah akses langsung dari luar kelas.

- Setiap kelas mengelompokan perilaku dan data terkait.

  Contoh: Home, GameWin, GameLose, Level, Benda, Bomb, Bread, Cake, Donut, Dynamite, Fire, Keranjang, Macaron, Muffin, Pizza, CountLevel, Counter, Nilai.

- Variabel instans dalam kelas dienkapsulasi dan sering dideklarasikan sebagai privat untuk menyembunyikan data.

  Contoh: Kelas Level memiliki variabel instans privat seperti images, cont, win, dan lose. Kelas Benda tidak memiliki variabel instans privat, tetapi memiliki beberapa metode yang dilindungi.

- Metode dienkapsulasi dalam kelas, menyediakan fungsionalitas khusus.
  Contoh: Metode act dalam berbagai kelas, metode meledak, breadFall, CakeFall, DonutFall, membakar, objectDisappear, moveKeranjang, MacaronFall, MuffinFall, PizzaFall, checkLevel, metode Counting.

- Modifikasi akses seperti public, private, dan protected digunakan untuk mengendalikan akses ke variabel dan metode.
  Contoh: Modifikasi akses private digunakan untuk variabel instans dalam kelas Level.

- Konstruktor mengenkapsulasi logika inisialisasi untuk objek.
  Contoh: Konstruktor public Home() menginisialisasi variabel instans level dan sound1.

- Metode getter dan setter tidak secara eksplisit ada dalam kode, tetapi kelas Counter menyediakan metode (Counting) untuk memodifikasi variabel totalPoin.

e. Interaksi Antar Objek

Interaksi antar objek adalah kemampuan objek-objek dalam pemrograman berbasis objek (PBO) untuk saling berkomunikasi dan memengaruhi satu sama lain dalam konteks suatu program atau permainan.

1. Kelas '**Home**'

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Home extends World {
    Level level = new Level();
    GreenfootSound sound1 = new GreenfootSound("DesiJourney.wav");
    public Home() {
        super(800, 750, 1);
    }

    public void act(){
        if(Greenfoot.isKeyDown("enter") || Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(level);
            sound1.play();
        }
    }
}
```

2. Kelas **'Level'**

```
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Level extends World {
    private GreenfootImage[] images =
    {
        new GreenfootImage("level1.png"),
        new GreenfootImage("level2.png"),
        new GreenfootImage("level3.png")
    };

    Counter cont = new Counter();
    GameWin win = new GameWin();
    GameLose lose = new GameLose();

    public Level() {
        super(640, 480, 1);
        setBackground(images[0]);
        isi();
    }

    public void act() {
        if (getObjects(Bomb.class).isEmpty()) bombJatuh();
```

3. Kelas **'Benda'**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Benda extends Actor {
    public boolean atWorldEdge() {
        if(getX() < 10 || getX() > getWorld().getWidth() - 10)
        return true;
        if(getY() < 10 || getY() > getWorld().getHeight() - 10)
        return true;
        else
        return false;
    }


    public boolean canSee(Class clss) {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }


    public void eat(Class clss) {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        if(actor != null) {
            getWorld().removeObject(actor);
        }
```

4. Kelas Bom **(‘Bomb’)**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bomb extends Benda {
    public void act() {
       setLocation(getX(), getY()+4);
       meledak();
    }

    public void meledak() {
        if (canSee(Keranjang.class)) {
            ((Counter)getWorld().getObjects(Counter.class).get(0)).Counting(-1);
            Greenfoot.playSound("bomb.wav");
        }

        if (atWorldEdge()) {
            getWorld().removeObject(this);
        }
    }

}
```

5. Kelas **‘Keranjang’**

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Keranjang extends Benda {
    public void act() {
        moveKeranjang();
        objectDisappear();
    }

    public void objectDisappear() {
        if (canSee(Muffin.class)) {
            eat(Muffin.class);
            ((Counter)getWorld().getObjects(Counter.class).get(0)).Counting(5);
            Greenfoot.playSound("score.wav");
        }

        if (canSee(Bread.class)) {
            eat(Bread.class);
            ((Counter)getWorld().getObjects(Counter.class).get(0)).Counting(10);
            Greenfoot.playSound("score.wav");
        }

        if (canSee(Macaron.class)) {
            eat(Macaron.class);
            ((Counter)getWorld().getObjects(Counter.class).get(0)).Counting(15);
```

Dengan contoh di atas, dapat terlihat bagaimana setiap kelas memiliki metode act() yang berisi logika permainan dan interaksi antar objek. Misalnya, kelas Bomb mengecek apakah ada objek Keranjang di dekatnya dan mengurangkan poin jika terdeteksi. Kelas Keranjang mengecek apakah objek makanan tertentu terlihat dan memanggil metode eat() untuk mengonsumsinya.

f. Overriding dan Overloading

- Overriding

Overriding terjadi ketika suatu kelas anak (subclass) menyediakan implementasi ulang dari metode yang sudah didefinisikan di kelas induk (superclass). Metode dengan nama yang sama di kelas anak akan menggantikan atau "mengoverride" implementasi metode di kelas induk.

Pada kelas Counter, terdapat metode Counting yang muncul dalam kelas anak CountLevel. Ini disebut overriding karena metode dengan nama yang sama didefinisikan ulang di kelas anak.

Kelas **'Counter' :** Overriding

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Counter extends Actor {
    public int totalPoin = 0;

    public Counter() {
        setImage(new GreenfootImage("0", 30, Color.WHITE, Color.BLACK));
    }

    public void Counting (int hitung) {
        totalPoin += hitung;
        setImage(new GreenfootImage("" + totalPoin, 30, Color.WHITE, Color.BLACK));
    }
    // Metode yang ditumpuk untuk memungkinkan pembaruan counter dengan gambar kustom
    public void Counting(int hitung, GreenfootImage customImage) {
        totalPoin += hitung;
        setImage(customImage);
    }
}
```

Kelas '**CounLevel**' : Overriding

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class CountLevel extends Actor {
    Counter lvl = new Counter();
    String[] levelNow = {"Level 1", "Level 2", "Level 3"};
    GreenfootImage customImage = new GreenfootImage("Custom Image", 30, Color.RED, Color.BLACK);

    public void act() {
        checkLevel();
    }

    public void checkLevel() {
        lvl.Counting(50, customImage); // Menggunakan metode yang ditumpuk dengan gambar kustom
        if (lvl.totalPoin >= 50) {
            lvl.Counting(400, customImage); // Menggunakan metode yang ditumpuk dengan gambar kustom
        }

        if (lvl.totalPoin >= 400) {
            lvl.Counting(0); // Menggunakan metode asli tanpa gambar kustom
        }
    }
}
```

- Overloading

Overloading terjadi ketika terdapat beberapa metode dengan nama yang sama di suatu kelas, tetapi dengan parameter-parameter yang berbeda. Metode overloading memungkinkan kelas untuk menyediakan beberapa cara berbeda untuk memanggil metode dengan nama yang sama.

Pada kelas Counter, terdapat dua metode Counting yang memiliki nama yang sama tetapi jumlah parameter yang berbeda. Ini disebut overloading.

Kelas **'Counter' :** Overloading

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Counter extends Actor {
    public int totalPoin = 0;

    public Counter() {
        setImage(new GreenfootImage("0", 30, Color.WHITE, Color.BLACK));
    }

    public void Counting (int hitung) {
        totalPoin += hitung;
        setImage(new GreenfootImage("" + totalPoin, 30, Color.WHITE, Color.BLACK));
    }
    // Metode yang ditumpuk untuk memungkinkan pembaruan counter dengan gambar kustom
    public void Counting(int hitung, GreenfootImage customImage) {
        totalPoin += hitung;
        setImage(customImage);
    }
}
```

Kelas '**CounLevel**' : Overriding method dengan jumlah parameter yang berbeda

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class CountLevel extends Actor {
    Counter lvl = new Counter();
    String[] levelNow = {"Level 1", "Level 2", "Level 3"};
    GreenfootImage customImage = new GreenfootImage("Custom Image", 30, Color.RED, Color.BLACK);

    public void act() {
        checkLevel();
    }

    public void checkLevel() {
        lvl.Counting(50, customImage); // Menggunakan metode yang ditumpuk dengan gambar kustom
        if (lvl.totalPoin >= 50) {
            lvl.Counting(400, customImage); // Menggunakan metode yang ditumpuk dengan gambar kustom
        }

        if (lvl.totalPoin >= 400) {
            lvl.Counting(0); // Menggunakan metode asli tanpa gambar kustom
        }
    }
}
```

Selanjutnya, pada kelas Level, terdapat beberapa metode isi() yang merupakan overloading karena memiliki nama yang sama tetapi jumlah parameter yang berbeda.

Kelas '**Level**' : Method overloading dengan parameter yang berbeda

```
public void isi(){
    Keranjang keranjang = new Keranjang();
    Nilai nilai = new Nilai();
    addObject(keranjang, 648, 403);
    addObject(cont, 57, 69);
    addObject(nilai, 64, 44);
    nilai.setLocation(59, 44);
    keranjang.setLocation(373, 403);
}
```

BAB III

PENUTUP

A. Kesimpulan

Laporan ini menyimpulkan dengan menekankan pentingnya Greenfoot sebagai alat pembelajaran dan memberikan gambaran umum tentang konsep-konsep kunci pemrograman berbasis objek yang diilustrasikan melalui pengembangan permainan dengan Greenfoot.

Dengan ini, diharapkan pembaca dapat memahami secara menyeluruh konsep dasar pemrograman berbasis objek dan mampu mengaplikasikannya menggunakan platform Greenfoot. Semoga proyek ini memberikan kesempatan bagi Anda untuk mendalami dan menerapkan konsep-konsep yang telah dipelajari selama semester ini.