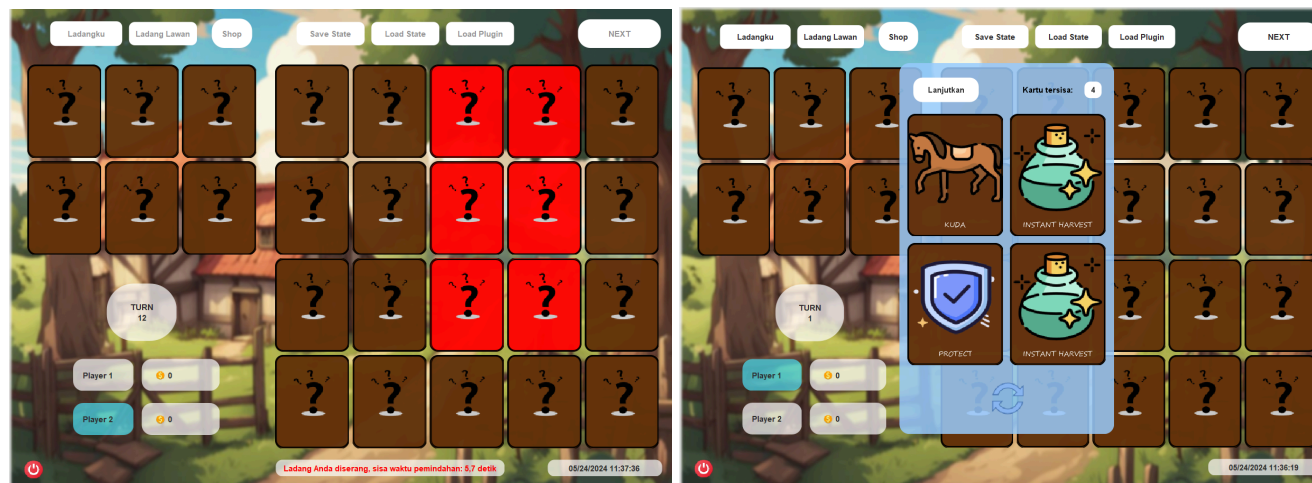


Kelompok : GTG

1. 13522029 / Ignatius John Hezekiel Chan
  2. 13522043 / Daniel Mulia Putra Manurung
  3. 13522050 / Kayla Namira Mariadi
  4. 13522093 / Matthew Vladimir Hutabarat
  5. 13522098 / Suthasoma Mahardika Munthe
- Asisten Pembimbing : 13520022 / Primanda Adyatma Hafiz

## 1. Deskripsi Umum Aplikasi

Aplikasi Remidi Kelola Kerajaan adalah permainan mengelola ladang dan berlomba mengumpulkan uang melawan pemain lain. Aplikasi ini berbasis desktop dan dibuat menggunakan bahasa pemrograman Java dan *library* Java Swing untuk bagian antarmuka. Aplikasi ini memiliki beberapa *game-mechanic* seperti manajemen ladang dengan *drag-and-drop*, kartu kartu efek yang dapat diterapkan ke hewan atau tanaman, seruan beruang yang dapat menghancurkan ladang, sistem toko untuk membeli dan menjual barang. Aplikasi kami juga memungkinkan untuk menyimpan dan memuat kondisi di tengah permainan menggunakan beberapa plugin atau menambah plugin.



## 2. Kakas GUI: Java Swing

Java Swing merupakan bagian dari *library* Java untuk membuat GUI pada aplikasi desktop. Java Swing menawarkan komponen GUI yang lebih canggih dan fleksibel ketimbang AWT (Abstract Window Toolkit). Kelebihan dari kakas ini, pengguna memungkinkan membuat tema *custom* tanpa mengubah kode dasar, komponen-komponen GUI yang lengkap dibanding AWT, penanganan peristiwa seperti gerakan mouse, penekanan tombol, dan masih banyak lagi.

Untuk menggunakan Java Swing dapat mengikuti langkah berikut:

- a. Pastikan sudah menginstall Ekstensi Java dan JDK (Java Development Kit) yang sesuai
- b. JavaSwing dapat digunakan dengan melakukan *import javax.swing.\**
- c. Membuat kelas yang merupakan turunan dari *javax.swing.JFrame*

Komponen-komponen utama yang dibutuhkan ialah:

- a. Swing Containers: Komponen yang dapat menampung komponen Swing lainnya seperti JFrame, JPanel, JScrollPane
- b. Swing Controls: Komponen interaktif untuk pengguna berinteraksi dengan aplikasi seperti JButton, JLabel, JCheckBox
- c. Swing Menus: Komponen untuk membuat menu di aplikasi seperti JMenu, JMenuBar
- d. Swing Windows: Komponen untuk membuat *window* tambahan selain *window* utama seperti JDialog
- e. Swing Filler: Komponen untuk mengatur ruang tata letak GUI seperti Glue, Box.Filler

Beberapa komponen yang disediakan Java Swing antara lain:

- a. JFrame: berfungsi menjadi *window* utama yang menampung komponen komponen didalamnya
- b. JButton: berfungsi untuk membuat tombol dan berinteraksi dengan tombol
- c. JLabel: berfungsi menjadi label untuk menampilkan tulisan atau gambar
- d. JCheckBox: berfungsi untuk menjadi tombol yang pilihannya Ya atau Tidak
- e. JTextField: berfungsi untuk menampung input user berupa text
- f. JComboBox: berfungsi menjadi dropdown input user
- g. JMenuBar: berfungsi untuk membuat menu di aplikasi

### 3. Plugin & Class Loader

Class Loader merupakan salah satu bentuk Reflection Java, yang memungkinkan program untuk memeriksa dan mengubah struktur dan perilaku aplikasi secara dinamis saat runtime. Class Loader merupakan program Java bertanggung jawab untuk memuat kelas-kelas ke dalam memori saat diperlukan. Class Loader mengelola proses pemuatan, verifikasi, dan linking antar kelas Java. Saat terdapat program java yang menggunakan class ClassLoader dan meminta untuk melakukan load sebuah kelas, JVM akan menerima request dan memulai pencarian class dengan nama bersesuaian mulai dari Bootstrap Class Loader, Extension Class Loader, hingga akhirnya Application Class Loader. Jika class ditemukan, maka JVM akan memuat kelas tersebut ke dalam memori. Prinsip Class Loader ini digunakan pada penerapan fitur plugin game, yang memungkinkan pengguna untuk menambahkan Loader untuk jenis file load/save yang baru. Class Loader akan mencari kelas loader yang ingin ditambahkan saat runtime, dan kemudian menambahkan loader baru tersebut pada memory sehingga bisa dipakai program.

Fitur plugin yang kami buat menerapkan konsep reflection yakni Class Loader, yang akan me-*load* file yang ditemukan pada jar yang sesuai dan valid untuk ditambahkan sebagai plugin loader baru. Fitur ini akan menerima path pada jar file dan men-traverse jar file tersebut untuk menemukan class-class yang dapat ditambahkan. Berikut langkah-langkah lebih jelas mengenai alur penambahan plugin.

1. Membuat objek baru JarFile, yang akan menyimpan konten-konten jar file
2. Melakukan pencarian pada konten JarFile untuk menemukan file yang sesuai yakni entry dengan suffix “.class” dan bukan merupakan suatu directory
3. Jika ditemukan class yang valid, load class tersebut pada suatu variabel.
4. Lakukan pengecekan class apakah sesuai dengan ketentuan suatu class plugin loader yang sesuai. Class yang sesuai yakni class yang bukan merupakan interface dan class yang mengimplement interface / API yang kami sediakan untuk plugin, yakni interface FileLoader. Jika class tidak valid, lanjutkan pencarian
5. Jika menemukan class yang sesuai, lakukan instansiasi objek kelas tersebut dan tambahkan objek tersebut pada daftar FileLoader milik game.

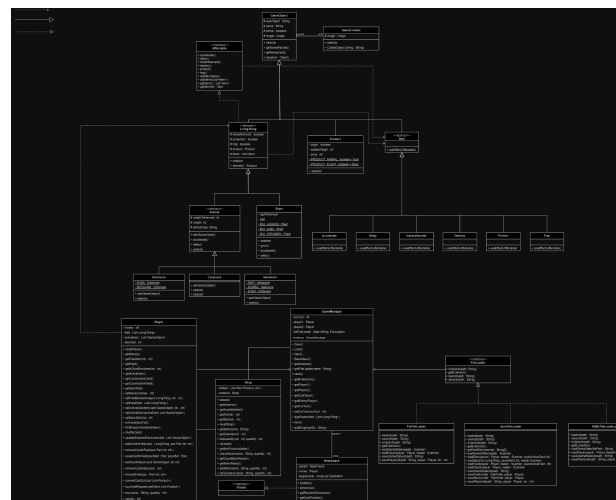
Dengan reflection, plugin bisa ditambahkan secara dinamis. Jika ingin menambahkan plugin baru, cukup memuat JAR yang berisi kelas yang mengimplementasikan FileLoader, lalu memanggil addPlugin pada gamemanager.

## 4. Class Diagram

Diagram kelas terdiri dari berbagai komponen utama yaitu `GameObject`, `GameContext`, `GameManager`, dan `Shop`. `GameContext` sebagai kelas yang memetakan `GameObject` terhadap objek-objeknya. `GameObject` sebagai kelas induk bagi objek permainan yaitu kelas `LivingThing` (kelas abstrak yang akan di-*inherit* oleh `Animal` dan `Plant`), `Product` (hasil panen dari `Animal` dan `Plant`), dan `Item` (objek yang dapat memakai efek seperti `delay`, `instant harvest`, `accelerate`, `destroy`, `trap`, dan `protect`). Dengan meng-*inherit* `GameObject`, objek pada permainan memiliki informasi tentang nama, tipe, serta ikonnya (untuk keperluan GUI). Sementara itu, `GameManager` sebagai kelas yang mengontrol jalannya permainan. Kelas ini memiliki kontrol atas pemain, serangan beruang, dan *loader* file. Kelas `Shop` sendiri nantinya berhubungan dengan `Player` dan `GameManager`.

Interface `Affectable` berfungsi untuk mendefinisikan fungsi fungsi yang memberi efek terhadap objek, seperti efek `delay`, `instant harvest`, `accelerate`, `destroy`, `trap`, dan `protect`. `Item` memiliki fungsi abstrak `useEffect` yang menerima interface `Affectable` ini sebagai parameter. Objek turunan dari kelas `Livingthing` yang memiliki kumpulan `Item` kemudian dapat mengimplementasikan fungsi pada interface sesuai `Item`-nya. Juga terdapat Interface `FileLoader` yang mendefinisikan fungsi untuk memuat plugin `Txt`, `Json`, dan `YAML`.

Sumber : [Class Diagram](#)



## 5. Konsep OOP

### 5.1. Inheritance

No	Parent Class (src/main/java/models/)	Child Class (src/main/java/models/)
1	GameObject.java	LivingThing.java
		Product.java
		Item.java
2	LivingThing.java	Animal.java
		Plant.java
3	Animal.java	Omnivore.java
		Carnivore.java
		Herbivore.java
4	Item.java	Accelerate.java
		Delay.java
		InstantHarvest.java
		Destroy.java
		Protect.java
		Trap.java

5	FileLoader.java	TxtFileLoader.java
		JsonFileLoader.java
		YamlFileLoader.java

## 5.2. Composition

No	Kelas (src/main/java/models/)	Komposisi (src/main/java/models/)
1	GameManager.java	Player.java (2)
		FileLoader.java (banyak)
2	Player.java	LivingThing (banyak)
		GameObject (banyak)
3	LivingThing.java	Product (1)
		Item (banyak)

### 5.3. Interface

No	Interface (src/main/java/models/)	Implementator (src/main/java/models/)
1	Affectable.java	Accelerate.java
		Delay.java
		InstantHarvest.java
		Destroy.java
		Protect.java
		Trap.java
2	FileLoader.java	JSONFileLoader.java
		TxtFileLoader.java
		YAMLFileLoader.java

### 5.4. Method Overriding dan Method Overloading

#### A. Method Overriding

No	Method	Lokasi (src/main/java/models/)	Penggunaan
1	destroy	LivingThing.java	Method-method ini <i>override</i> method pada interface Affectable
	protect		

	trap		
2	accelerate	Animal.java	Method ini <i>override</i> method pada interface Affectable
		Plant.java	
3	delay	Animal.java	Method ini <i>override</i> method pada interface Affectable
		Plant.java	
4	useEffect	Accelerate.java	Method-method ini <i>override</i> method pada class Item untuk memanggil efek efek berbeda sesuai dengan jenis itemnya pada kelas LivingThing
		Delay.java	
		InstantHarvest.java	
		Destroy.java	
		Protect.java	
		Trap.java	
5	eat	Herbivore.java	Method ini <i>override</i> method pada class Animal untuk mengecek makanan tersebut bisa dimakan atau tidak
		Carnivore.java	
		Omnivore.java	

## B. Method Overloading

No	Method	Lokasi (src/main/java/models/)	Penggunaan
1	getQuantity(int)	Shop.java	Method ini <i>overloading</i> untuk memudahkan developer



	getQuantity(String)		mengakses objek Shop
2	setQuantity(int, int)	Shop.java	Method ini <i>overloading</i> untuk memudahkan developer mengakses objek Shop
	setQuantity(String,int)		
3	getItem(int)	Shop.java	Method ini <i>overloading</i> untuk memudahkan developer mengakses objek Shop
	getItem(String)		
4	Constructor	Seluruh Class	Method ini <i>overloading</i> untuk memudahkan developer membuat objek dengan parameter berbeda-beda

## 5.5. Polymorphism

No	Lokasi (src/main/java/models/)	Kelas atau Method	Penggunaan
1	Player.java	Method placeLiving	Method ini menerima kelas LivingThing namun akan dikembalikan objek yang berbeda-beda sesuai dengan atribut nama dari kelas LivingThing
2	Player.java	Method harvestField	Method ini akan menerima <i>List of LivingThing</i> . Untuk mengetahui apakah objek bisa di-harvest perlu <i>dynamic casting</i> menjadi kelas Animal atau Plant
3	Carnivore.java	Method eat	Method ini menerima objek GameObject. GameObject ini yang harus merupakan <i>instance</i> Product atau method akan <i>throw</i> GameException
	Omnivore.java		
	Herbivore.java		
4	Player.java	Method placeDeckToField	Method ini akan memindahkan objek GameObject dari

			<i>deck</i> ke <i>field</i> . Objek dari <i>deck</i> merupakan kelas <i>GameObject</i> tapi bentuknya bisa beranekaragam . Untuk melakukan validasi, objek <i>GameObject</i> harus di- <i>check</i> merupakan <i>instance</i> dari kelas <i>LivingThing</i> / <i>Product</i> / <i>Item</i>
5	GameContext.java	Method createObject	Method ini menerima string nama. Method ini akan menghasilkan kelas <i>GameObject</i> yang sesuai dengan input string tersebut.

## 5.6. Java API Collection

No	Kelas (src/main/java/models/)	API Collection
1	LivingThing.java	List of Item
2	TxtFileLoader.java	Stream
3	GameContext.java	HashMap of String, String
4	GameContext.java	ArrayList

## 5.7. SOLID

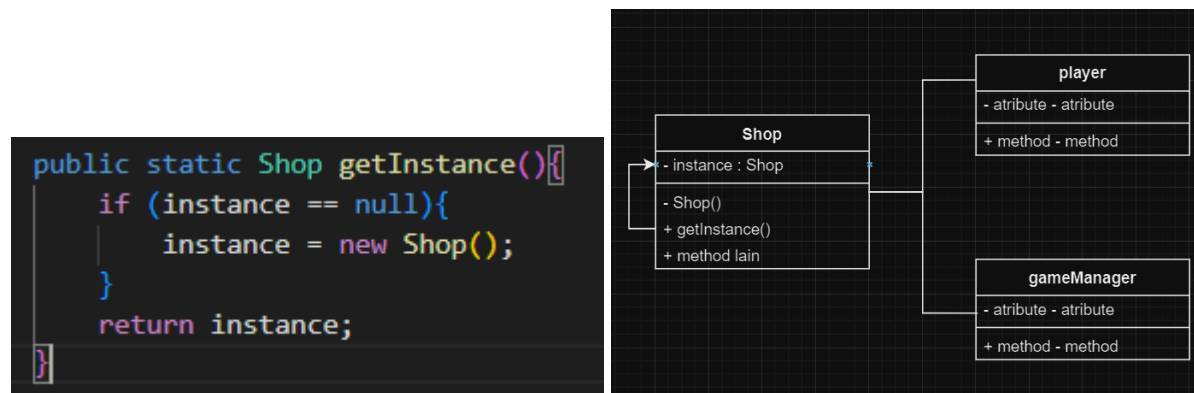
PRINSIP SOLID	PENJELASAN
S	Kelas Accelerate (src/main/modals/Accelerate.java) bertugas hanya untuk menangani efek item accelerate saja, dan efek item lain seperti Delay, InstantHarvest akan ditangani oleh kelas yang bersangkutan.
	Kelas Carnivore (src/main/modals/Carnivore.java) bertugas hanya untuk menangani hewan karnivora (hiu darat) secara spesifik dan bukan hewan lain, atau fungsionalitas yang lebih mengeneral lainnya, seperti fungsi makan spesifik hewan karnivora, tetapi bukan fungsi penggunaan efek item yang dapat digunakan oleh hewan lain.
	Kelas GameObject dan GameContext (src/main/modals/GameObject.java dan src/main/modals/GameContext.java). GameObject merepresentasikan sebuah instansi objek dalam game beserta atribut dan metode yang di dalamnya. Sedangkan GameContext menyediakan sebuah <i>mapping</i> antara gameobject dan instansi mereka masing - masing. GameContext juga menyediakan metode yang menginstantiasi gameobject menggunakan reflection dan handle berbagai exception dalam pembuatan gameobject.
O	Kelas Item dan turunannya (src/main/modals/Item.java). Item memiliki sebuah metode useEffect yang tidak diimplementasikan dalam item itu sendiri, melainkan diimplementasikan dalam kelas - kelas turunannya seperti accelerate, delay, instanharvest, dan lainnya, sehingga jika akan ditambahkan kelas turunan Item baru, kelas Item tak perlu diubah. Kelas turunan baru hanya perlu meng- <i>extend</i> kelas Item.
	Kelas Animal dan turunannya (src/main/modals/Animal.java). Animal memiliki sebuah metode eat yang tidak diimplementasikan dalam kelas animal itu sendiri, melainkan diimplementasikan dalam kelas - kelas turunannya yaitu carnivore, herbivore, dan omnivore.

	Kelas LivingThing yang mengimplementasikan Affectable dan turunannya (src/main/modals/LivingThing.java). Living thing memiliki metode accelerate yang diimplementasikan berbeda untuk animal dan plant.
L	Kelas Item dan turunannya (src/main/modals/Item.java). Setiap kelas turunan item yang melingkupi accelerate, delay, instant harvest, destroy, protect, trap, dapat menggantikan class Item itu sendiri, dengan tambahan atribut atau metode mereka masing - masing. Setiap Item dapat menggunakan metode useEffect yang berbeda efeknya tiap item.
	Kelas LivingThing dan turunannya (src/main/modals/LivingThing.java). Setiap kelas turunan LivingThing yang melingkupi animal dan plant, dapat menggantikan kelas LivingThing itu sendiri dengan tambahan atribut atau metode mereka masing - masing. Setiap turunan livingthing dapat menggunakan metode harvest yang untuk tiap instance dari animal dan plant itu memiliki efek yang berbeda - beda.
	Kelas GameObject dan turunannya (src/main/modals/GameObject.java). Setiap kelas turunan GameObject yang melingkupi LivingThing, Product, dan Item, dapat menggantikan kelas GameObject itu sendiri dengan tambahan atribut atau metode mereka masing - masing.
I	Kelas Item dan turunannya (src/main/modals/LivingThing.java). Setiap kelas turunan LivingThing (accelerate, delay, instant harvest, destroy, protect, trap) tidak mengimplementasikan metode interface untuk item lainnya yang tidak dapat digunakan di item tersebut. Contohnya kelas accelerate tidak mengimplementasikan metode delay, instant harvest, dll.
D	Interface Affectable (src/main/modals/Affectable.java). Kelas LivingThing bergantung pada Affectable dan tidak langsung bergantung pada kelas item dan turunannya.

## 5.8. Design Pattern

- Creational Design Patterns
  - Singleton Pattern

Kelas Shop (src/main/java/models/Shop/java) merupakan kelas yang mengimplementasikan pattern *singleton*. Kelas ini berfungsi untuk menyimpan mengelola produk produk yang dijual / dibeli player. Dengan menerapkan *singleton*, kelas ini hanya akan diinstansiasi sekali saja sehingga meskipun penggunaan kelas ini tersebar di seluruh program, mereka akan menggunakan objek yang sama. Selain kelebihanannya, pattern ini punya kelemahan bahwa dibutuhkan *special treatment* jika kita mengharapkan kelas shop bekerja di lingkungan *multithread*. Jika pattern ini tidak diterapkan, akan sulit untuk menjaga konsistensi data karena data yang krusial tersebar di beberapa objek dan dibutuhkan *special treatment* untuk menyinkronisasikan.



- Factory Pattern

Factory Design Pattern adalah salah satu bagian dari creational design pattern. Design pattern ini berfokus dan bertujuan untuk memisahkan penggunaan sebuah objek dan pembuatan atau creation sebuah objek. Hal ini membuat kita mendapatkan fleksibilitas dalam creation sebuah objek tanpa harus bergantung pada suatu implementasi khusus. Konsep factory design pattern bertanggung jawab untuk membentuk menciptakan objek - objek dari kelas - kelas yang bersesuaian dengan kebutuhan dan keinginan pengguna.

Hal ini membuat kita dapat melakukan creation objek - objek tanpa mengatakan secara eksplisit tipe dari objek tersebut. Pengguna menggunakan factory untuk mendapatkan objek yang diinginkan.

Dalam program ini kami menerapkan design pattern Factory. Hal ini kami terapkan dalam class dan turunan dari class GameObject. Animal, Plant, dan Product yang adalah kelas turunan dari GameObject, masing - masing memiliki sebuah fungsi konstruktor yang dapat melakukan construct terhadap class yang diinginkan secara reflection. Dengan menggunakan parameter nama objek yang ingin di-construct, kita dapat melakukan construct terhadap gameobject yang kita inginkan.

Dalam kasus kami, creator class kami adalah fungsi metode untuk melakukan creation objek yang menerima string untuk melakukan construct objek yang diinginkan user. Sedangkan product class kami adalah kelas hasil itu sendiri, yaitu Animal, Plant, Product, dll.

```
private static Map<String, Herbivore> HERBIVORES = new HashMap<>();

static {
    HERBIVORES.put(key:"SAPI", new Herbivore(name:"SAPI", active:true, Icon.COW, instantHarvest:false, protection:false, t...false);
    HERBIVORES.put(key:"DOMBA", new Herbivore(name:"DOMBA", active:true, Icon.SHEEP, instantHarvest:false, protection:false, t...false);
    HERBIVORES.put(key:"KUDA", new Herbivore(name:"KUDA", active:true, Icon.HORSE, instantHarvest:false, protection:false, t...false);
}

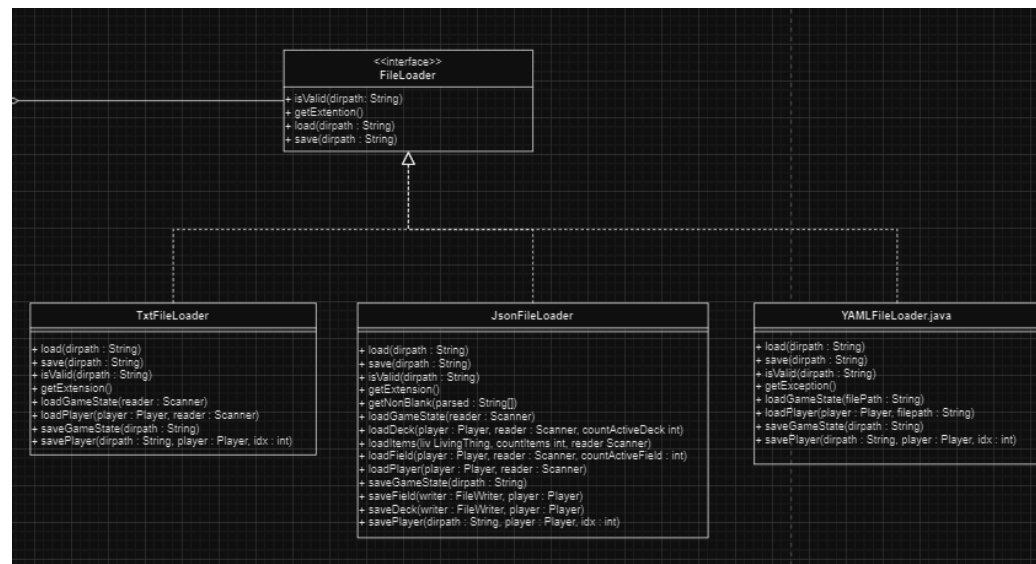
private Herbivore(String name, boolean active, Image image, boolean instantHarvest, boolean protection, boolean trap, Product p...
    super(name, active, image, instantHarvest, protection, trap, product, weightToHarvest, weight, animalType:"HERBIVORE");
}

public Herbivore(Herbivore other) {
```

Sebagai contoh pada Herbivores, yang adalah class turunan dari Animal, di mana fungsi konstruktor Herbivore ini, yang memanfaatkan HashMap, kita dapat melakukan construct of herbivore hanya dengan memanfaatkan sebuah string saja dengan atribut lainnya telah didefinisikan.

- Structural Design Patterns
- Adapter Pattern

Adapter Pattern adalah Design pattern yang memungkinkan objek dengan interface yang tidak kompatibel untuk bekerjasama. Adapter berfungsi sebagai perantara yang mengelola interface satu objek dengan lainnya. Pada konteks program kami, kami membuat *interface* FileLoader yang bekerja sebagai adapter. Developer yang ingin menambah plugin seperti CSV atau XML bisa menambahkan plugin ke program dengan interface FileLoader. Hal ini membuat plugin plugin yang tidak kompatibel menjadi bekerja dengan interface kami.



- Behavioral Design Pattern
  - Mediator

Mediator Design Pattern adalah behavioral design pattern yang membuat loose coupling antar objek, yang membatasi tiap objek untuk berkomunikasi satu sama lain secara eksplisit. Melainkan, mereka berkomunikasi melalui suatu objek mediator yang menjadi perantara antara objek. Objek mediator ini akan mengenkapsulasi bagaimana sekumpulan objek berinteraksi dan menjadi pusat komunikasi kompleks dan control logic.

Pada design kelas kami, bagian yang mengimplementasi design pattern ini merupakan bagian penggunaan item pada suatu hewan atau tanaman, yang diperantarai oleh suatu interface Affectable yang menandakan suatu objek bisa dikenai oleh item-item pada game, seperti Accelerate, Delay, dsb. Penggunaan item pada suatu hewan akan terdapat pada method item yakni useEffect, yang akan menerima parameter class yang implement Affectable. Hewan atau tanaman merupakan kelas yang mengimplement affectable ini, dan bisa kita pass sebagai parameter pada fungsi useEffect. Disinilah interface Affectable bertindak sebagai mediator / perantara antara komunikasi objek Item dengan objek Hewan dan Tanaman.

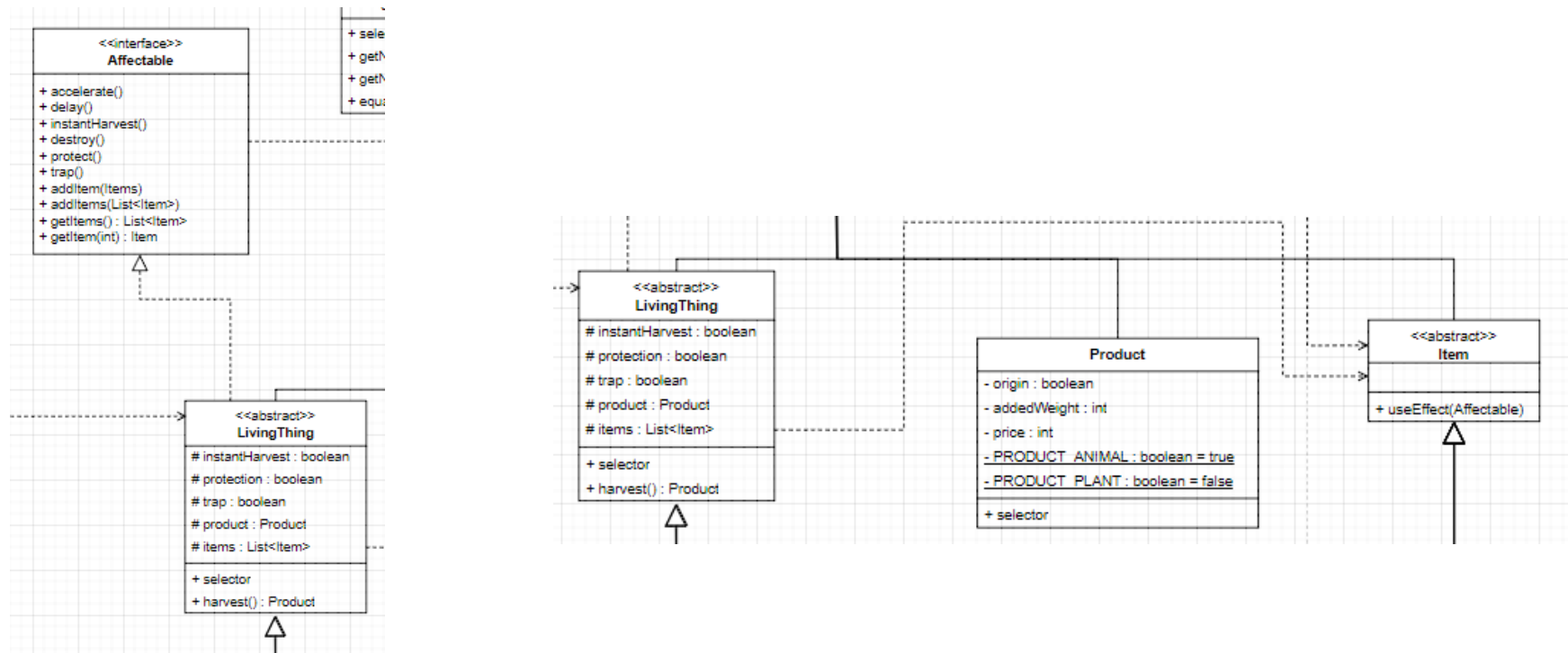
```
public interface Affectable {  
    public void accelerate();  
    public void delay();  
    public void instantHarvest();  
    public void destroy();  
    public void protect();  
    public void trap();  
    public void addItem(Item item);  
    public void addItem(List<Item> items);  
    public List<Item> getItems();  
    public Item getItem(int index);  
}
```

Interface Affectable berperan sebagai mediator, menyediakan fungsi-fungsi sebagai media komunikasi antara item dan object Hewan atau Tanaman.



```
public abstract void useEffect(Affectable affectable);
```

Pada fungsi ini, item menambahkan efek-nya pada Hewan atau Tanaman melalui interface Affectable. Disini, hewan mengimplement interface Affectable dan mengimplementasi efek item masing-masing sesuai jenis atau state Hewan/Tanaman-nya. Berikut class design dari implementasi design pattern ini.



Dapat dilihat bahwa komunikasi antara LivingThing (superclass Hewan dan Tanaman) dengan item dihubungkan oleh suatu interface bernama Affectable. Affectable disini berperan sebagai mediator antara kedua objek.

## 5.9. Reflection

- Method addPlugin pada kelas GameManager (src/main/GameManager.java)

menggunakan reflection untuk menambahkan FileLoader baru ke List of FileLoader yang dimiliki GameManager. Penambahan dilakukan dengan mendapatkan kelas yang bersangkutan dengan file terlebih dahulu, lalu jika valid kelas yang akan di-load akan diekstrak methodnya dengan getDeclaredMethod untuk diinvoke kemudian. Loader dari kelas juga di instansiasi dengan method reflection newInstance().

```
if (!entry.isDirectory() && entry.getName().endsWith(suffix:".class")) {
    String classPath = entry.getName().replace(target:"/", replacement:".").replace(target:".class", replacement:"");
    Class<?> loadedClass = loader.loadClass(classPath);
    Class<?> interfaceClass = Class.forName(className:"models.FileLoader");
    if (!interfaceClass.isAssignableFrom(loadedClass) || loadedClass.isInterface()) {
        continue;
    }
    jarFile.close();
    Method getExt = loadedClass.getDeclaredMethod(name:"getExtension");
    FileLoader fileLoader = (FileLoader) loadedClass.getDeclaredConstructor().newInstance();
    listFileLoader.put((String) getExt.invoke(fileLoader), fileLoader);
    return;
}
```

Dapat dilihat penggunaan reflection untuk mendapatkan objek class berdasarkan input classPath, mendapatkan method dari kelas tersebut, menginvoke method yang didapatkan sebelumnya, mendapatkan constructornya, dan melakukan instantiasi objek.

- Method createObject pada GameContext

Method ini bertindak sebagai layer abstraksi, dimana method ini hanya menerima parameter string nama dan mengembalikan objek baru yang sesuai tanpa perlu mengetahui jenis class dari nama tersebut. Method ini menerapkan design pattern factory berbasis Reflection. Method ini menerima parameter input string berupa nama objek, dan kemudian melakukan pencarian jenis class objek

tersebut berdasarkan nama . Kemudian setelah didapat jenis class objek, digunakan reflection untuk mendapatkan constructor kelas tersebut, menginstantiate objek baru dengan konstruktor tersebut, dan mereturn objek tersebut.

```
public static GameObject createObject(String name){
    try{
        String className = OBJECT_TYPE.get(name);
        Class<?> classObj = Class.forName(className);
        if(className.equals(anObject:"models.Carnivore")){
            return (GameObject)classObj.getDeclaredConstructor().newInstance();
        }
        if(classObj.getSuperclass().getName().equals(anObject:"models.Item")){
            return (GameObject)classObj.getDeclaredConstructor().newInstance();
        }
        return (GameObject)classObj.getDeclaredConstructor(...parameterTypes:String.class).newInstance(name);
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```

Dapat dilihat penggunaan reflection untuk mendapatkan objek kelas berdasarkan input className, mendapatkan konstruktor dari kelas bersangkutan, dan menginstantiasi objek dari kelas tersebut.

## 5.10. Threading

Kelas (src/main/java/gui/)	Penggunaan
BearAttack.java	Threading digunakan untuk <i>handle</i> timer agar program dapat menjalankan beberapa timer dalam satu waktu. Setiap thread memegang timer yang berbeda satu sama lain

## 6. Bonus Yang dikerjakan

### 6.1. Unit Testing

Implementasi Unit Testing dilakukan dengan menggunakan JUnit dan Jacoco. JUnit adalah framework unit testing yang dapat memberikan metode - metode pengujian dan assertion untuk memastikan dan melakukan verifikasi pada kode yang sedang diuji. Jacoco digunakan untuk menghasilkan sebuah laporan yang memberikan hasil coverage dari unit test yang telah dibuat, coverage ini menunjukkan seberapa banyak kode yang telah teruji oleh unit test yang telah dibuat.





































Dalam membuat unit test, pendekatan awalnya adalah dengan menentukan fungsi mana saja yang akan dilakukan uji. Untuk kepentingan penanganan setiap branch, maka unit testing akan dilakukan secara menyeluruh dan sangat diusahakan untuk menyediakan sebuah kondisi uji untuk setiap kondisi yang mungkin terjadi dalam fungsi tersebut. Hal ini akan sering didapatkan pada method - method yang memiliki conditional. Unit testing juga dilakukan pada method - method yang tidak memiliki conditional seperti setter dan getter untuk memastikan bekerjanya method tersebut.

Untuk setiap pembuatan unit test akan dilakukan pendekatan sebagai berikut. Pertama tentukan conditional yang akan dicakup dalam sebuah unit test, hal ini melingkupi branching - branching yang akan dipenuhi dengan unit test yang akan dibuat. Kedua adalah menyiapkan kondisi yang memenuhi kondisi tersebut, hal ini melingkupi setup objek - objek yang akan digunakan. Ketiga adalah melakukan panggilan method yang akan diuji, hal ini akan dilakukan untuk setiap branching dari method yang diuji. Dan yang terakhir adalah assertion, atau memeriksa bahwa hasil yang dikembalikan dari method yang diuji sesuai dengan hasil yang diharapkan.

Dalam berjalannya proses unit testing, untuk setiap FAILED yang ditemukan, dengan asumsi kondisi unit testing sudah benar, maka kesalahan yang harus diperbaiki terdapat dalam kode yang kita miliki, dan bukan mengubah unit testing untuk memenuhi kondisi yang kita inginkan. Setelah seluruh unit testing berjalan tanpa FAILED, maka akan dilakukan pemeriksaan pada coverage unit test yang dihasilkan oleh Jacoco, dengan hasil coverage yang baik, maka objek yang telah diuji terbukti dapat berjalan dengan baik di skenario - skenario yang telah dicover.

KAYLA-DAN-4-ORANG-GANTENG &gt; models

## models

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">Product</a>		100%		n/a	0	7	0	26	0	7	0	1
<a href="#">Herbivore</a>		100%		100%	0	11	0	25	0	6	0	1
<a href="#">Omnivore</a>		100%		100%	0	10	0	22	0	6	0	1
<a href="#">Animal</a>		100%		100%	0	12	0	26	0	10	0	1
<a href="#">Carnivore</a>		100%		100%	0	9	0	17	0	4	0	1
<a href="#">Accelerate</a>		100%		n/a	0	3	0	7	0	3	0	1
<a href="#">Trap</a>		100%		n/a	0	3	0	7	0	3	0	1
<a href="#">Protect</a>		100%		n/a	0	3	0	7	0	3	0	1
<a href="#">Delay</a>		100%		n/a	0	3	0	7	0	3	0	1
<a href="#">Destroy</a>		100%		n/a	0	3	0	6	0	3	0	1
<a href="#">Item</a>		100%		n/a	0	2	0	4	0	2	0	1
<a href="#">GameContext</a>		98%		87%	2	8	1	42	1	4	0	1
<a href="#">LivingThing</a>		97%		100%	1	21	2	49	1	20	0	1
<a href="#">Plant</a>		97%		70%	3	16	0	33	0	11	0	1
<a href="#">GameObject</a>		90%		92%	4	24	7	52	3	17	0	1
<a href="#">Player</a>		87%		83%	21	87	33	213	5	37	0	1
<a href="#">Shop</a>		81%		65%	13	40	20	75	3	18	0	1
<a href="#">GameManager</a>		79%		90%	2	25	15	73	0	14	0	1
<a href="#">InstantHarvest</a>		66%		n/a	1	3	2	6	1	3	0	1
<a href="#">YAMLFileLoader</a>		0%		0%	29	31	162	164	10	12	0	1
<a href="#">TxtFileLoader</a>		0%		0%	30	31	126	127	11	12	0	1
<a href="#">JSONFileLoader</a>		0%		0%	65	65	234	234	18	18	1	1
Total	2,652 of 5,474	51%	209 of 402	48%	171	417	602	1,222	53	216	1	22

## 6.2. Memperindah UI

Fitur	Tujuan
Menu Awal Game	Bagaikan Film di bioskop. Sebuah aplikasi perlu intro dan outro untuk menjelaskan keadaan
Kondisi Menang dan Seri	

Gambar Object berubah	Perubahan ini dibuat untuk mengembangkan nuansa <i>cartoon</i> sehingga enak dilihat oleh pengguna
Background	
Label berat / umur pada kartu di Field	Hal ini bertujuan agar pengguna lebih mudah melihat perkembangan kartu dan tidak perlu melihat detail kartu untuk melihat berat atau umur
Perubahan display tanaman seiring bertambah umur	Untuk memperindah tampilan
Lagu	Untuk meningkatkan semangat dalam bermain

## 7. Pembagian Tugas

NIM	Nama	Pembagian Tugas
13522029	Ignatius John Hezkien Chan	Plugin Plugin, GameManager, logic game, Improve program, Debugger, laporan
13522043	Daniel Mulia Putra Manurung	Unit test, laporan, readme, class diagram, cari asset tampilan
13522050	Kayla Namira Mariadi	Unit test, laporan, GUI button
13522093	Matthew Vladimir Hutabarat	Desain, bantu GUI, initiator program, laporan, class diagram, cari asset tampilan
13522098	Suthasoma Mahardika Munthe	Desain, ALL GUI, Drag and Drop, bearAttack, Improve program, Debugger, class diagram, cari asset tampilan

## 8. Foto Kelompok



## 9. Lampiran

- Form Asistensi : [Form Asistensi](#)
- Repositori : [Repositori](#)
- Class Diagram : [Class Diagram](#)