

Kode Kelompok : AMP

Nama Kelompok : Aland Mulia Pratama

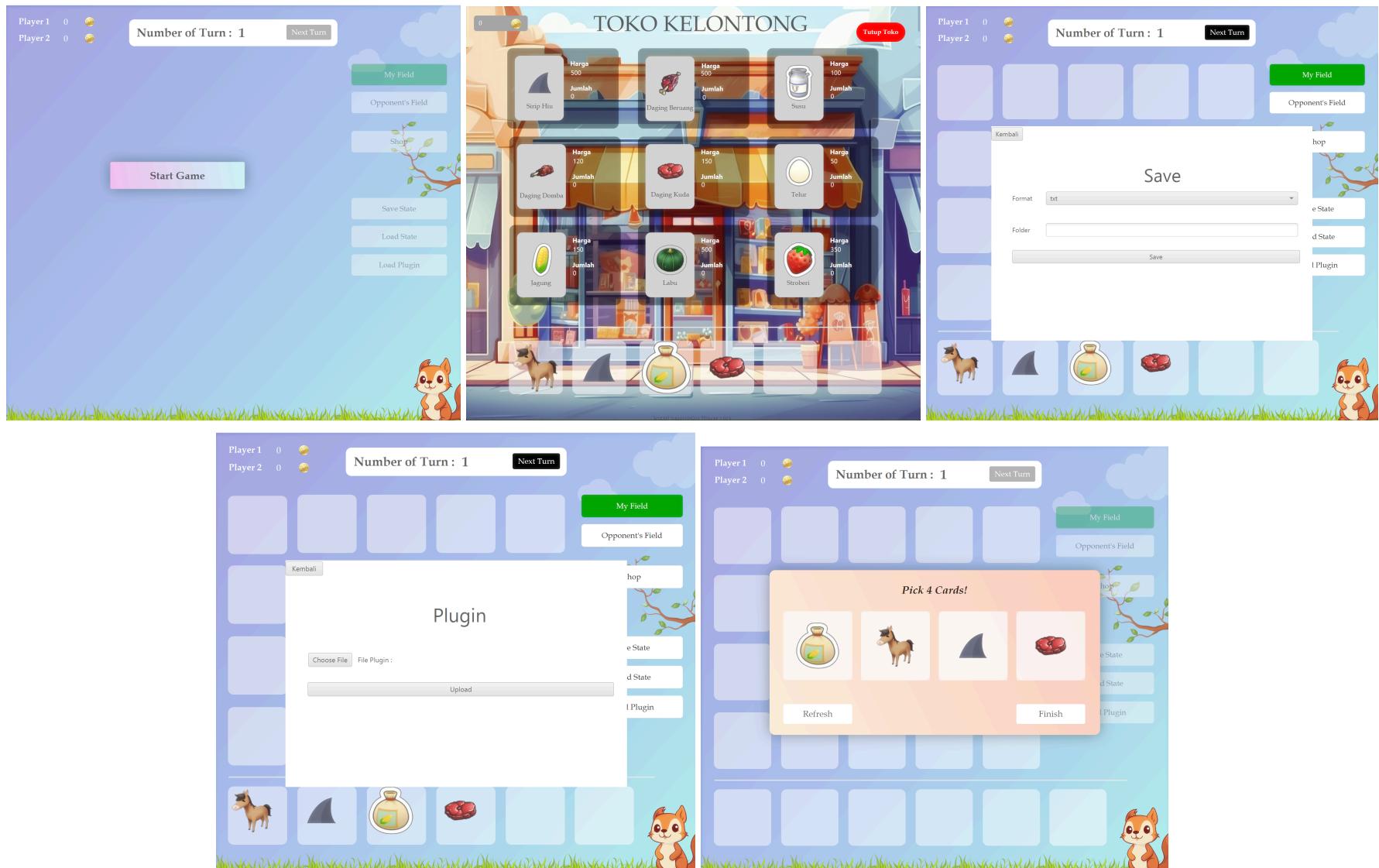
1. 13522124 / Aland Mulia Pratama
2. 13522138 / Andi Marihot Sitorus
3. 13522150 / Albert Ghazaly
4. 13522161 / Mohammad Akmal Ramadan
5. 13522163 / Atqiyah Haydar Luqman

Asisten Pembimbing : Marcellus Michael Herman Kahari

1. Deskripsi Umum Aplikasi

Aplikasi **Remidi Kelola Kerajaan** adalah sebuah permainan simulasi yang menggabungkan konsep pengelolaan ladang dengan elemen permainan kartu. Dalam permainan ini, pemain akan menjadi pengelola ladang dan bertugas menanam tanaman serta merawat hewan di dalamnya. Ladang direpresentasikan dalam bentuk matriks petak-petak, di mana setiap petak dapat ditanami tanaman atau ditempatkan hewan. Pemain memiliki sebuah deck kartu yang berisi berbagai jenis tanaman dan hewan. Mereka dapat menempatkan kartu-kartu ini pada petak ladang mereka atau menjualnya di toko jika berada pada deck aktif. Selain itu, pemain juga harus waspada terhadap serangan beruang yang dapat merusak objek-objek dalam ladang secara acak.

Fitur save & load memungkinkan pemain untuk menyimpan dan memuat progres permainan mereka dalam format teks, sehingga mereka dapat melanjutkan permainan dari titik terakhir. Selain itu, aplikasi ini bersifat ekstensibel dengan adanya dukungan untuk plugin, memungkinkan pengguna untuk menambahkan format file save & load lainnya dengan mudah. Dalam setiap permainan, terdapat dua pemain yang bersaing untuk mengumpulkan sebanyak mungkin uang dalam 20 turn. Pemenang akan ditentukan berdasarkan jumlah uang yang berhasil dikumpulkan oleh masing-masing pemain. Aplikasi ini juga menampilkan antarmuka grafis (GUI) yang menarik dan mudah dipahami, mempermudah pemain dalam mengelola ladang mereka.



2. Kekas GUI: Java FX

JavaFX adalah toolkit GUI (Graphical User Interface) yang memungkinkan pengembangan aplikasi desktop dengan tampilan modern dan interaktif. Menggunakan JavaFX, kita dapat menciptakan antarmuka aplikasi yang profesional dan menarik, lengkap dengan berbagai komponen GUI interaktif.

Siklus hidup aplikasi JavaFX terdiri dari beberapa tahap:

1. Membuat Kelas Utama Aplikasi

Langkah pertama adalah membuat kelas utama yang meng-extend kelas Application dari JavaFX. Kelas ini akan menjadi titik awal aplikasi dan mengatur siklus hidupnya.

2. Override Metode start()

Setelah kelas utama dibuat, metode start() dari kelas Application harus di-override. Metode ini dipanggil saat aplikasi dijalankan dan di sini kita dapat menambahkan komponen GUI yang ingin ditampilkan.

3. Membuat Stage dan Scene

Untuk menampilkan komponen GUI, kita perlu membuat Stage dan Scene. Stage merepresentasikan window aplikasi, sementara Scene adalah konten yang akan ditampilkan di window tersebut. Kita bisa menentukan ukuran dan judul Stage serta menambahkan Scene ke dalamnya.

4. Menambahkan Layout

Setiap komponen GUI dalam JavaFX membutuhkan layout yang mengatur posisinya di window. JavaFX menyediakan berbagai jenis layout seperti BorderPane, GridPane, StackPane, HBox, dan VBox, yang bisa disesuaikan sesuai kebutuhan.

Untuk menampilkan GUI di JavaFX, beberapa komponen utama yang harus ada dalam kode meliputi:

1. Stage

Stage merepresentasikan window aplikasi. Pengembang harus membuat Stage dan menentukan judul serta ukurannya.

2. Scene

Scene adalah konten yang akan ditampilkan di Stage. Setiap Scene memiliki root node, yang menjadi induk dari semua komponen di Scene.

3. Layout

Setiap komponen dalam Scene memerlukan layout untuk menentukan bagaimana posisi dan ukuran dari komponen tersebut. JavaFX menyediakan berbagai jenis layout seperti BorderPane, GridPane, HBox, VBox, dan lain-lain.

4. Komponen GUI

Setelah menentukan Stage, Scene, dan Layout, kita bisa menambahkan komponen GUI seperti tombol, label, kotak teks, checkbox, radio button, dan lain-lain. Komponen ini dapat diatur tampilan dan aksinya.

Beberapa komponen yang disediakan oleh JavaFX meliputi:

1. Button

Tombol yang bisa diberi label, gambar, dan aksi saat ditekan.

2. TextField

Kotak teks untuk memasukkan data.

3. Label

Komponen untuk menampilkan teks informasi pada GUI.

4. CheckBox

Kotak centang untuk memilih satu atau beberapa opsi.

5. RadioButton

Tombol radio untuk memilih satu opsi dari beberapa opsi.

6. StackPane

Layout untuk overlay satu item di atas item lainnya, misalnya meng-overlay tombol di atas ikon.

7. ScrollPane

Daftar item yang dapat digulir tetapi tidak bisa diklik, untuk menampilkan daftar item atau view.

3. Plugin & Class Loader

Cara Kerja Class PluginLoader

1. Menerima Lokasi File JAR:

Metode loadAndInvoke menerima sebuah string jarPath yang merupakan path ke file JAR yang berisi kelas plugin yang ingin dijalankan.

2. Normalisasi Path:

Path yang diberikan dinormalisasi untuk memastikan penggunaan slash yang konsisten (mengganti backslash dengan slash). File uploader mereturn absolute path sehingga perlu dilakukan konversi path yang dapat dibaca oleh program.

3. Membuat URLClassLoader:

URLClassLoader adalah mekanisme di Java yang memungkinkan Anda memuat kelas dari lokasi luar (seperti file JAR) pada runtime. URL ke file JAR dibuat dan digunakan untuk menginisialisasi URLClassLoader, memungkinkan kelas yang ada di dalam JAR untuk dimuat.

4. Ekstraksi Nama Kelas:

Nama kelas diekstrak dari nama file JAR menggunakan metode extractClassName. Metode ini mengasumsikan bahwa nama file JAR sama dengan nama kelas yang ingin dimuat, sebuah pendekatan yang umum tetapi bisa dibatasi. Ekstraksi dilakukan dengan menghapus ekstensi .jar dan path direktori dari nama file JAR.

5. Memuat Kelas:

Kelas dimuat menggunakan URLClassLoader dengan nama yang telah diekstrak.

6. Instansiasi Objek Kelas:

Instance dari kelas yang dimuat dibuat. Ini dilakukan dengan memanggil konstruktor default kelas tersebut.

7. Pencarian dan Pemanggilan Metode:

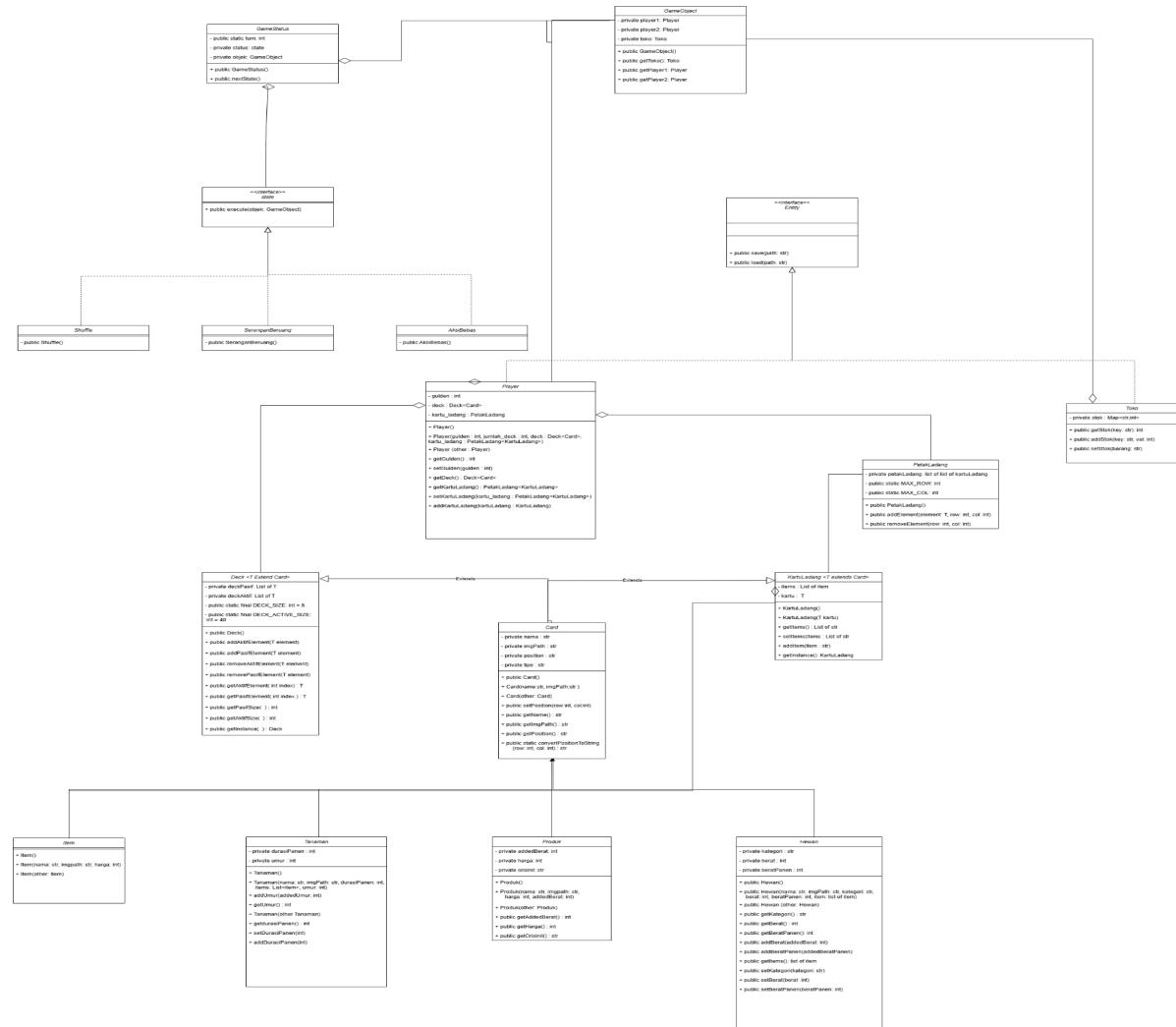
Metode findMethodReturningString mencari melalui semua metode yang dideklarasikan di kelas untuk menemukan metode pertama yang mengembalikan tipe String. Jika metode tersebut ditemukan, metode tersebut dipanggil pada instance yang baru dibuat. Jika tidak, sebuah exception dilempar.

8. Mengembalikan Hasil:

Hasil dari pemanggilan metode yang mengembalikan String dikembalikan ke pemanggil.

Plugin diimplementasikan ke dalam user interface tampilan program dengan menggunakan kelas PluginController. PluginController membantu menjembatani antara logika aplikasi dan antarmuka pengguna, memungkinkan aplikasi untuk secara efektif memanfaatkan plugin untuk memperluas atau menyesuaikan fungsionalitasnya sesuai kebutuhan pengguna. PluginController memiliki kemampuan untuk memuat dan mengintegrasikan plugin secara dinamis meningkatkan fleksibilitas dan skalabilitas aplikasi, memungkinkan penambahan fitur atau fungsi baru tanpa perlu mengubah kode sumber aplikasi inti.

4. Class Diagram



Berdasarkan Implementasi Konsep-Konsep OOP yang telah dipelajari dan spesifikasi Tugas Besar yang telah diberi, kami mendesain struktur kelas yang memadai untuk memenuhi kriteria spesifikasi program seperti gambar di atas. Kelas-kelas yang dibuat memaksimalkan konsep-konsep OOP seperti kelas Items, Hewan, Tanaman, dan Produk inherit ke kelas Card agar dapat digunakan sebagai elemen dari ladang dan deck. Selain itu, kelompok ini juga mengimplementasikan kelas GameState dan GameObject sebagai abstraksi lebih tinggi dari kelas-kelas yang lain. Dengan begitu, penggunaan program seperti command-command seperti shuffle dan serangan beruang lebih mudah untuk dijalankan. Selain itu, desain diagram kelas kami juga mengimplementasikan interface agar memudahkan penggunaan fungsi-fungsi dan sebagai abstraksi lebih tinggi dari modul-modul dengan level abstraksi rendah. Untuk lebih jelasnya lagi, di bawah merupakan penjelasan konsep OOP yang digunakan.

Class diagram dapat diakses di https://drive.google.com/file/d/1-VNLhkTN0_F-n19hXGhuDezS91xzyhbS/view?usp=sharing

5. Konsep OOP

5.1. Inheritance

- Kelas Hewan (src/main/java/card/Hewan.java) merupakan anak dari kelas Card (src/main/java/card/Card.java)
- Kelas Item (src/main/java/card/Item.java) merupakan anak dari kelas Card (src/main/java/card/Card.java)
- Kelas Tanaman (src/main/java/card/Tanaman.java) merupakan anak dari kelas Card (src/main/java/card/Card.java)
- Kelas Produk (src/main/java/card/Produk.java) merupakan anak dari kelas Card (src/main/java/card/Card.java)

5.2. Composition

- Kelas Deck (src/main/java/deck/Deck.java) memiliki banyak kelas Card (src/main/java/card/Card.java)
- Kelas GameObject (src/main/java/gameobject(GameObject.java) memiliki 2 kelas Player (src/main/java/player/Player.java), satu kelas Toko (src/main/java/toko/Toko.java), banyak kelas Hewan (src/main/java/card/Hewan.java), banyak kelas Item (src/main/java/card/Item.java), banyak kelas Tanaman (src/main/java/card/Tanaman.java), banyak kelas Produk (src/main/java/card/Produk.java)

- Kelas KartuLadang (src/main/java/petakladang/KartuLadang.java) memiliki satu kelas Card (src/main/java/card/Card.java) dan banyak kelas Item (src/main/java/card/Item.java)
- Kelas Player (src/main/java/player/Player.java) memiliki satu kelas Deck (src/main/java/deck/Deck.java) dan satu kelas PetakLadang (src/main/java/petakladang/PetakLadang.java)
- Kelas GameStatus (src/main/java/gamestatus/GameStatus.java) memiliki satu kelas GameObject (src/main/java/gameobject(GameObject.java), satu kelas State (src/main/java/state/State.java), dan satu kelas Entity(src/main/java/entity/Entity.java)

5.3. Interface

- Interface Entity (src/main/java/entity/Entity.java)
- Interface State (src/main/java/state/State.java)

5.4. Method Overriding dan Method Overloading

- Method save pada kelas Player (src/main/java/player/Player.java)
- Method addAktifElement pada kelas Deck (src/main/java/deck/Deck.java)
- Method start pada kelas Main (src/main/java/com/tubesoop/tubes2oop/Main.java)
- Method load pada kelas saveloadJSON (src/main/java/entity/saveloadJSON.java)
- Method execute pada kelas SeranganBeruang (src/main/java/state/SeranganBeruang.java)

5.5. Polymorphism

- Method addAktifElement pada kelas Deck (src/main/java/deck/Deck.java) menerima kelas T, tapi pada method ini dipanggil dengan kelas Produk di kelas Toko (src/main/java/toko/Toko.java)

5.6. Java API Collection

Tuliskan daftar penggunaan konsep ini di aplikasi, misalnya:

- Kelas Deck (src/main/java/deck/Deck.java) memiliki List of Card
- Kelas Toko (src/main/java/toko/Toko.java) memiliki Map of (String, Integer)
- Kelas PetakLadang (src/main/java/petakladang/PetakLadang.java) memiliki List of List of KartuLadang
- Kelas Shuffle (src/main/java/state/Shuffle.java) memiliki List of Card
- Kelas KartuLadang (src/main/java/petakladang/KartuLadang.java) memiliki List of Item

5.7. SOLID

1. Single-Responsibility Principle

- Kelas Item (src/main/java/card/Item.java) hanya bertugas untuk mengatur behaviour dari Item, sedangkan penggunaannya diatur oleh kelas-kelas lain seperti kelas KartuLadang (src/main/java/petakladang/KartuLadang.java) yang mengatur items pada kelas PetakLadang (src/main/java/petakladang/PetakLadang.java)
- Kelas Card (src/main/java/card/Card.java) hanya bertugas untuk mengatur behaviour dari Card, sedangkan penggunaannya diatur oleh kelas-kelas lain seperti kelas KartuLadang (src/main/java/petakladang/KartuLadang.java) yang mengatur Card pada kelas PetakLadang (src/main/java/petakladang/PetakLadang.java)
- Kelas Toko (src/main/java/toko/Toko.java) hanya bertugas untuk mengatur interaksi jual beli di dalam toko, sedangkan objek mana yang melakukan jual-beli diatur oleh kelas GameStatus (src/main/java/gamestatus/GameStatus.java)

2. Open-Closed Principle

- Kelas PluginLoader (src/main/java/plugin/PluginLoader.java) terbuka (open) untuk menambah fitur load player untuk ekstensi lainnya, tetapi tertutup (closed) dalam pengubahan program atau fungsi load di dalam kelas Player (src/main/java/player/Player.java)
- Kelas PluginLoader (src/main/java/plugin/PluginLoader.java) terbuka (open) untuk menambah fitur load state untuk ekstensi lainnya, tetapi tertutup (closed) dalam pengubahan program atau fungsi load di dalam kelas Toko (src/main/java/toko/Toko.java)

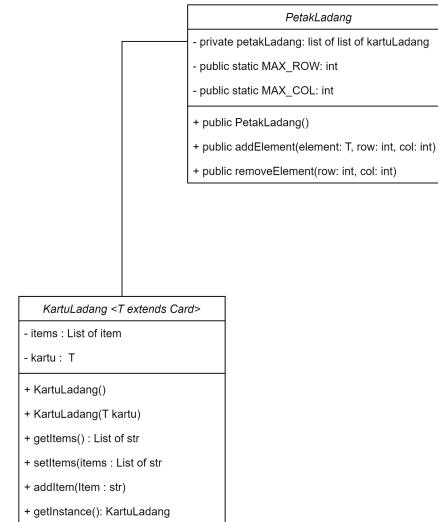
3. Liskov Substitution Principle

- Kelas Hewan (src/main/java/card/Hewan.java) dapat memiliki method-method yang dimiliki kelas Card (src/main/java/card/Card.java) meskipun merupakan jenis kelas yang berbeda.
 - Kelas Tanaman (src/main/java/card/Tanaman.java) dapat memiliki method-method yang dimiliki kelas Card (src/main/java/card/Card.java) meskipun merupakan jenis kelas yang berbeda.
 - Kelas Item (src/main/java/card/Item.java) dapat memiliki method-method yang dimiliki kelas Card (src/main/java/card/Card.java) meskipun merupakan jenis kelas yang berbeda.
4. Interface Segregation Principle
- Kelas Player (src/main/java/player/Player.java) tidak perlu mengimplementasi save & load player karena Interface Entity (src/main/java/entity/Entitiy.java) memiliki interface sebagai entitas yang dapat dijalankan save & load.
 - Kelas Toko (src/main/java/toko/Toko.java) tidak perlu mengimplementasi save & load gamestate karena Interface Entity (src/main/java/entity/Entitiy.java) memiliki interface sebagai entitas yang dapat dijalankan save & load.
 - Kelas Shuffle (src/main/java/state/Shuffle.java) tidak perlu mengimplementasi execute gamestate karena Interface State (src/main/java/state/State.java) memiliki interface sebagai entitas yang dapat dijalankan execute.
5. Dependency Inversion Principle
- Interface State (src/main/state/State.java) merupakan abstraksi tinggi yang dapat dipanggil oleh kelas GameStatus (src/main/gamestatus/GameStatus.java) (abstraksi level tinggi) dalam menjalankan fungsi execute shuffle dalam kelas Shuffle (src/main/java/state/Shuffle.java) (abstraksi level rendah).
 - Interface State (src/main/state/State.java) merupakan abstraksi tinggi yang dapat dipanggil oleh kelas GameStatus (src/main/gamestatus/GameStatus.java) (abstraksi level tinggi) dalam menjalankan operasi serangan beruang dalam kelas SeranganBeruang (src/main/state/SeranganBeruang.java) (abstraksi level rendah).

5.8. Design Pattern

- **Creational design pattern (Singleton)**

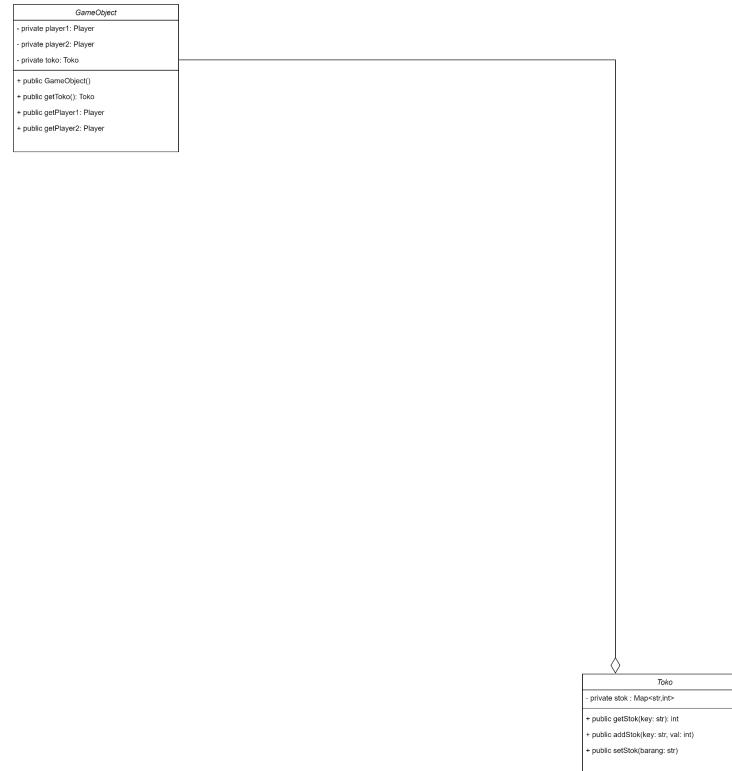
Creational design pattern digunakan pada kelas KartuLadang (src/main/java/petakladang/KartuLadang.java). Kami menggunakan getInstance untuk mendapatkan deepcopy dari list of items. Implementasi KartuLadang.getInstance() menggunakan pattern Singleton untuk membuat salinan dalam dari objek KartuLadang tanpa memperhatikan kelas kelas yang terlibat.



- Structural Design Pattern (Facade)

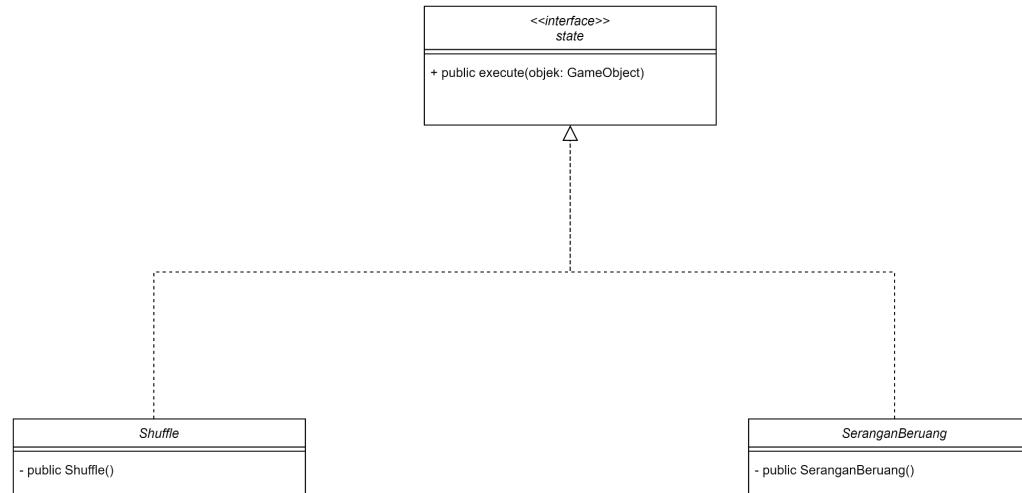
Dalam struktur program, GameStatus berperan sebagai abstraksi untuk menyederhanakan interaksi dengan kelas-kelas yang lebih kompleks dalam sistem game. Fungsi utama dari GameStatus adalah untuk mengelola status permainan yang berbeda, seperti Shuffle dan Serangan Beruang, dan mengontrol peralihan antar status ini melalui sebuah mekanisme yang mudah diikuti.





- **Behavioral design pattern (State)**

Dalam tugas besar 2 ini kami menggunakan Behavioral Design Pattern yaitu state yang di implementasi pada atribut status dalam kelas GameStatus. Pola State sangat efektif untuk sistem seperti game ini, di mana game dapat memiliki berbagai fase atau status yang berbeda seperti Shuffle, Serangan Beruang, atau Aksi Bebas. Atribut status pada GameStatus terhubung dengan suatu interface bernama state. Masing-masing fase ini memiliki perilaku yang berbeda dan cara interaksi dengan pemain yang unik dan diimplementasikan berdasarkan interface state tersebut.



5.9. Reflection

- Method `loadAndInvoke` pada kelas `PluginLoader` (`src/main/java/PluginLoader.java`) menggunakan reflection untuk membaca method dari file (.jar) yang diupload melalui program utama.

5.10. Threading

- Threading (`src/main/java/com/tubesoop/tubes2oop/FieldController.java`) terjadi dalam dua bagian fungsi `attackOnBeruang()`:
Pertama, `startCountdown()` menggunakan Timeline dari JavaFX untuk memulai countdown timer. Ini memungkinkan pembaruan UI secara periodik dengan menggunakan KeyFrame setiap 0.1 detik.

Kedua, scheduler.schedule() menggunakan ScheduledExecutorService untuk menjadwalkan tugas-tugas tertentu, seperti mengeksekusi serangan beruang atau menangani respons setelahnya. Platform.runLater() digunakan di dalamnya untuk memastikan bahwa tugas-tugas yang memengaruhi UI dieksekusi di thread UI utama, mencegah kesalahan dan memastikan konsistensi tampilan.

6. Bonus Yang dikerjakan

6.1. Unit Testing

Unit testing untuk Tugas Besar 2 kami dilaksanakan dengan memanfaatkan library JUnit5 dan Jacoco yang bertujuan untuk menghasilkan laporan coverage yang detail. Kami berhasil menjalankan unit testing dengan baik dan setidaknya menghandle diatas 90 persen untuk method dan baris yang telah dilakukan unit testing menggunakan JUnit. Kami berhasil mencapai coverage sekitar 66% untuk method dan 22% untuk line. Meskipun kami telah berhasil menerapkan unit testing, masih banyak aspek dari kode yang belum sempat teruji terutama saat kelas tersebut merupakan base class.

⌚ com.tubeso 6% (1/15)	2% (2/99)	1% (12/937)	0% (0/301)
⌚ ActionsC 0% (0/1)	0% (0/10)	0% (0/73)	0% (0/4)
⌚ Application 0% (0/1)	0% (0/2)	0% (0/6)	100% (0/0)
⌚ BearAttack 0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
⌚ FieldControl 0% (0/1)	0% (0/18)	0% (0/325)	0% (0/178)
⌚ LoadControl 0% (0/1)	0% (0/6)	0% (0/31)	0% (0/4)
⌚ Main 100% (1/1)	66% (2/3)	22% (12/54)	0% (0/18)
⌚ ObjectInfo 0% (0/1)	0% (0/6)	0% (0/107)	0% (0/29)
⌚ PlayerState 0% (0/1)	0% (0/5)	0% (0/15)	0% (0/4)
⌚ PluginControl 0% (0/1)	0% (0/6)	0% (0/36)	0% (0/8)
⌚ SaveControl 0% (0/1)	0% (0/6)	0% (0/31)	0% (0/4)
⌚ ShuffleCards 0% (0/1)	0% (0/10)	0% (0/124)	0% (0/28)
⌚ StartGame 0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)
⌚ TokoControl 0% (0/1)	0% (0/20)	0% (0/98)	0% (0/6)
⌚ TurnControl 0% (0/1)	0% (0/4)	0% (0/30)	0% (0/18)
⌚ WinnerCondition 0% (0/1)	0% (0/1)	0% (0/3)	100% (0/0)
⌚ exception 100% (10/10)	100% (10/10)	100% (10/10)	100% (0/0)
⚡ BeliE 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Deck 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Field 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ FileNumber 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Inapp 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Inapp 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Insert 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ JualE 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ KeyNumber 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
⚡ Select 100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)

▼ card	100% (5/5)	82% (32/39)	76% (58/76)	100% (0/0)
🕒 Card	100% (1/1)	45% (5/11)	39% (11/28)	100% (0/0)
🕒 Hewan	100% (1/1)	100% (11/11)	100% (21/21)	100% (0/0)
🕒 Item	100% (1/1)	80% (4/5)	85% (6/7)	100% (0/0)
🕒 Produk	100% (1/1)	100% (5/5)	100% (9/9)	100% (0/0)
🕒 Tanaman	100% (1/1)	100% (7/7)	100% (11/11)	100% (0/0)

6.2. Memperindah UI

Dalam Tugas Besar 2 ini kami telah melakukan implementasi tambahan terhadap user interface yang diberikan di dalam [spesifikasi Tugas Besar 2 OOP](#). Dalam bonus ini, kami mengkreasikan *color pallette* dan layout toko yang diberikan di dalam spesifikasi. Kami menambahkan beberapa components vector seperti awan, tupai, dan rumput dalam tampilan user interface ini. Kami juga memberikan background musik yang senada dengan tampilan user interface yang kami miliki. Selain itu, kami juga mengimplementasikan sound effect saat pengguna berinteraksi dengan program seperti mengclick barang atau saat melakukan panen dan meletakkan hewan atau tumbuhan pada ladang.

7. Pembagian Tugas

- 13522124 / Aland Mulia Pratama
 - Membuat unit testing untuk package Exception dan package Card
 - Membuat Kelas Toko, Controller Toko, dan GUI Toko
 - Menampilkan Background Music pada permainan dan sound effect
 - Membuat mekanisme plugin
 - Laporan Bab 1, Bab 2, dan Bab 3 dan Bab 6.
- 13522138 / Andi Marihot Sitorus
 - Membuat Fungsi Save dan implementasi pada GUI permainan
 - Membuat Fungsi Load dan implementasi pada GUI permainan
 - Membuat Fungsi Load dan Save untuk format Json dan XML dan implementasi pada GUI permainan
 - Membuat implementasi plugin pada GUI permainan
- 13522150 / Albert Ghazaly

- Membuat dokumentasi seluruh kelas dasar
- Implementasi design pattern
- Implementasi Drag & Drop
- Implementasi fungsi makan
- Implementasi efek-efek item seperti protection, destroy, delay
- Implementasi Tanam di petakLadang
- 13522161 / Mohammad Akmal Ramadan
 - Implementasi serangan beruang pada permainan
 - Implementasi mekanisme clock pada serangan beruang
 - Membuat unit testing untuk package Card
- 13522163 / Atqiyah Haydar Luqman
 - Inisialisasi struktur program
 - Inisialisasi user interface dari program (set up project)
 - Debugging masalah yang terdapat pada sistem program
 - Membuat GUI halaman utama dari permainan / program
 - Fungsionalitas dari User Interface serta memperbaiki user experience dari program permainan

8. Foto Kelompok



9. Link Terlampir

Dokumen asistensi: [W IF2210_TB2_Asistensi_AMP.docx](#)

Kode Kelompok : AMP

Nama Kelompok : Aland Mulia Pratama

1. 13522124 / Aland Mulia Pratama
2. 13522138 / Andi Marihot Sitorus

3. 13522150 / Albert Ghazaly
4. 13522161 / Mohammad Akmal Ramadan
5. 13522163 / Atqiyah Haydar Luqman

Asisten Pembimbing : Marcellus Michael Herman Kahari

1. Konten Diskusi

Q: Apakah aman untuk membuat loader dengan memasukan ke dalam objek turunan card untuk deck element dan card element?

A: Sejauh ini aman, karena tidak ada melanggar peraturan untuk OOP. Kakaknya tapi tetap perlu lihat class diagramnya

Q: "Apakah 40 deck saat awal perlu disimpan atau hanya di-generate 6 tiap turn secara random agar tidak perlu dibuat instansi nya"

A: "40 kartu pada deck perlu diinstansi saat awal permainan dan kartu tidak akan bisa habis karena jumlah round terbatas ≤ 10 "

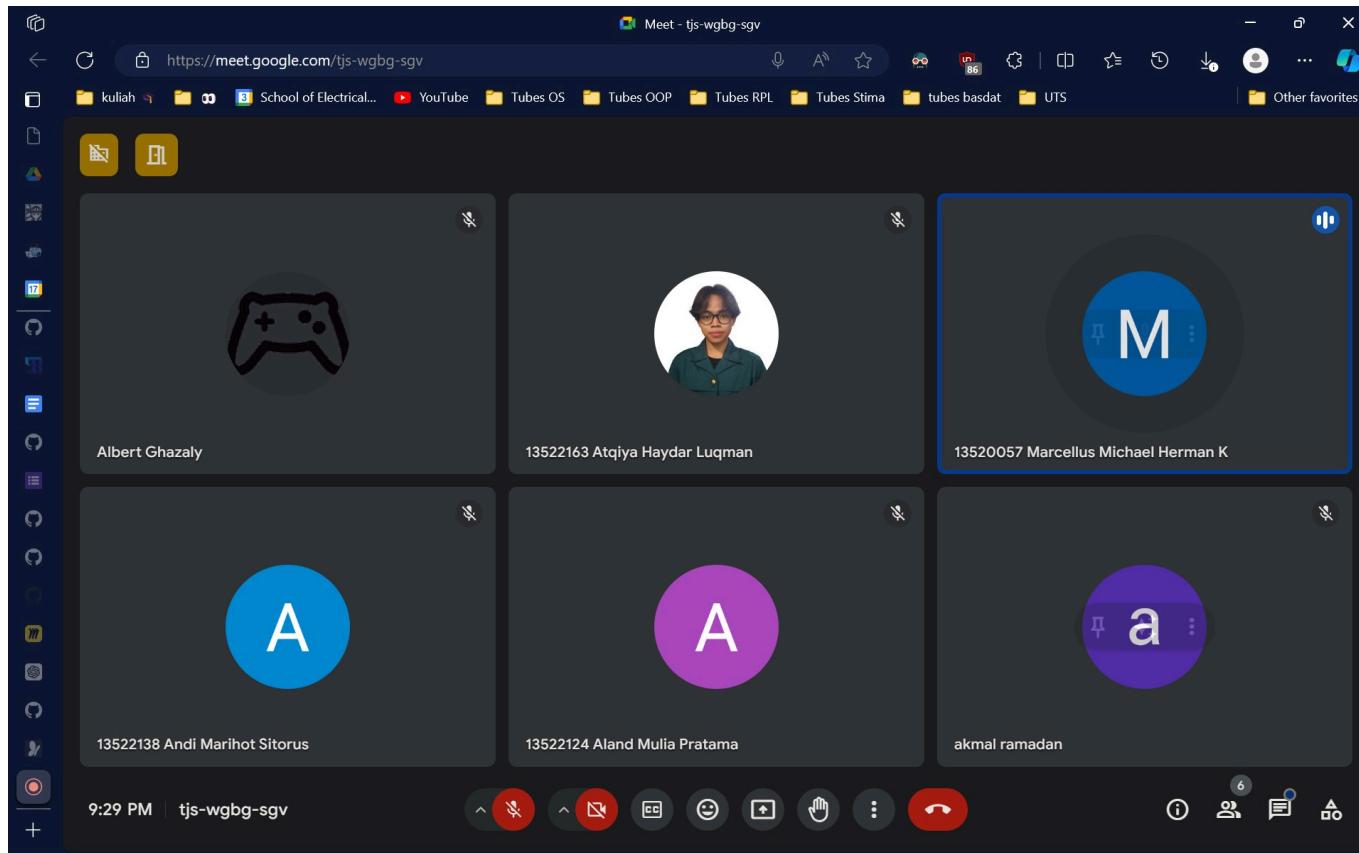
Q: "Kalau deck pasif kita ada 1 sementara deck aktif ada > 1 (misal 3), apakah kita hanya bisa mengambil 1 kartu yang tersisa saat awal turn"

A: "ya, hanya bisa ambil 1 kartu yang tersisa"

2. Tindak Lanjut

- Melakukan revisi class diagram sesuai dengan design pattern.
- Melakukan adjustment pada deck sesuai dengan jawaban di atas.

3. Foto Dokumentasi



Repository: https://github.com/AlbertGhazaly/IF2210_TB2_AMP

Draw io: https://drive.google.com/file/d/1-VNLhkTN0_F-n19hXGhuDezS91xzyhbS/view