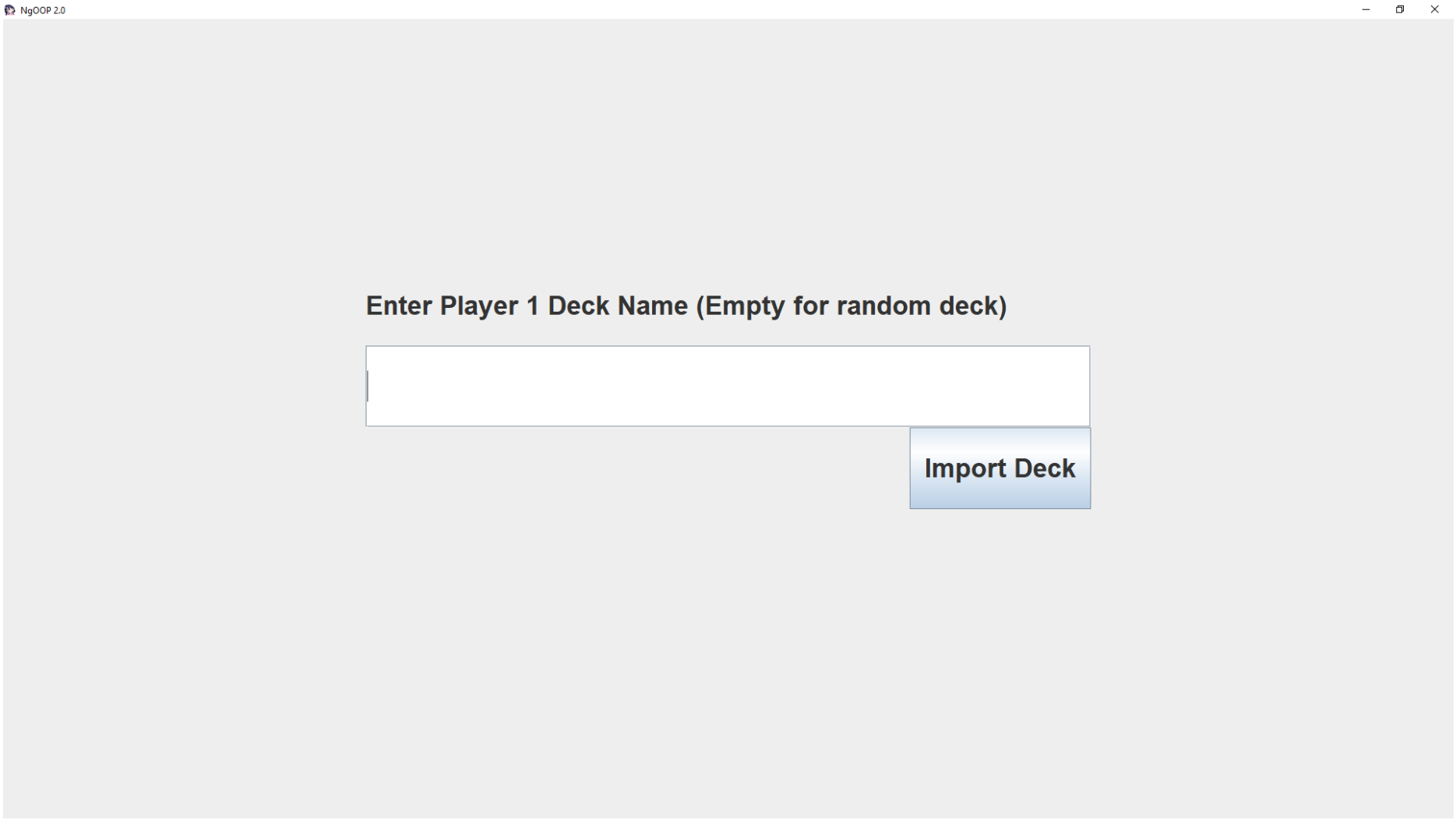
Kelas : 01
Nomor Kelompok : 09
Nama Kelompok : ngOOP2.0
1. 13520019 / Maharani Ayu Putri Irawan
2. 13520034 / Bryan Bernigen
3. 13520040 / Ng Kyle
4. 13520088 / Rio Alexander Audino
5. 13520115 / Maria Khelli
Asisten Pembimbing : Muhammad Fauzan Rafi Sidiq Widjonarto

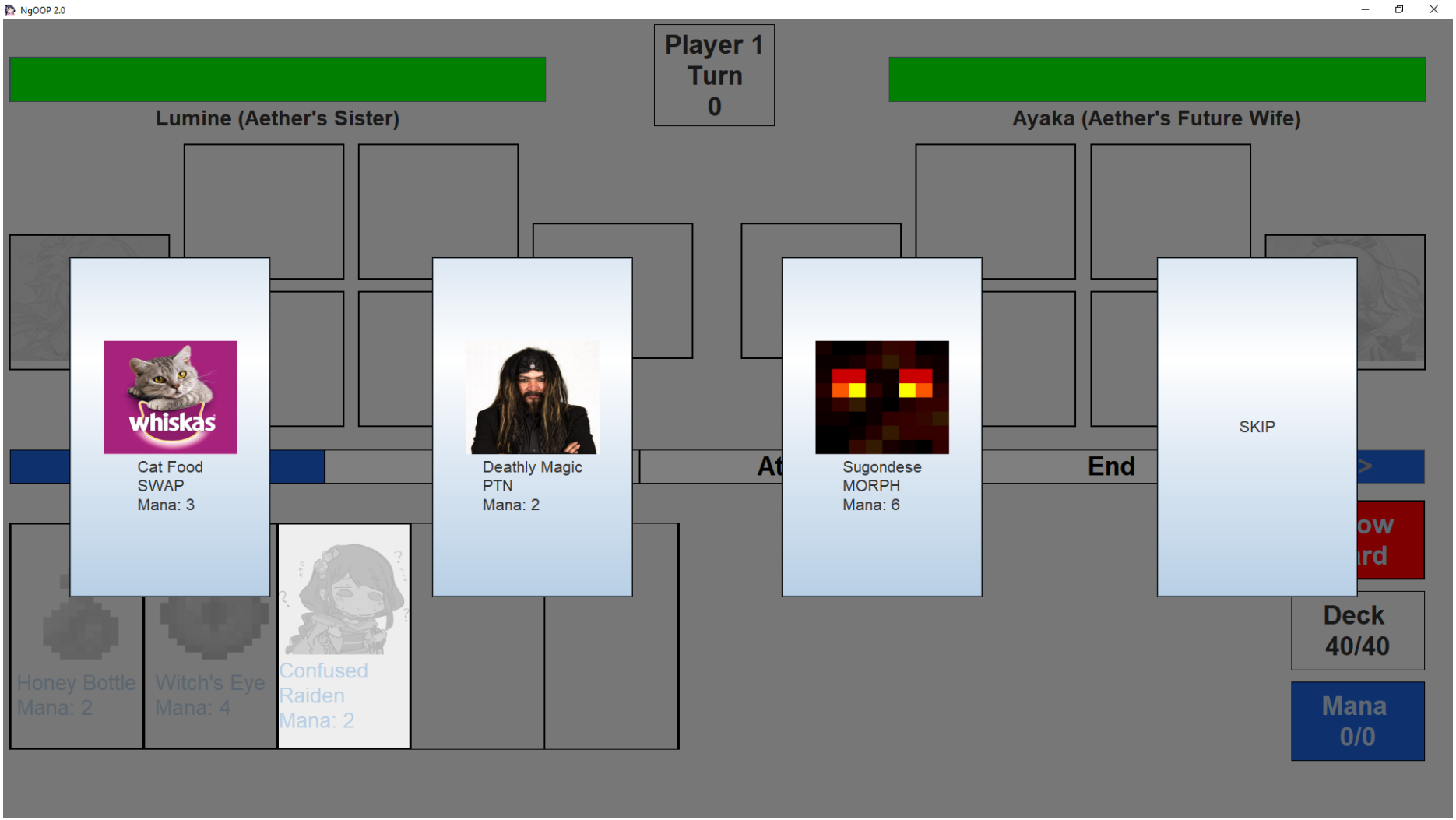
1. Deskripsi Umum Aplikasi

Aplikasi yang kami buat merupakan permainan berbasis GUI (Graphical User Interface) yang berjudul *Minecraft: Aether Realm Wars*. Aplikasi ini merupakan permainan kartu yang dijalankan secara bergilir antara dua pemain. Pada setiap giliran, pemarin akan melakukan fase *draw* dan *attack*. Seorang pemain akan menang ketika berhasil menghabiskan *health point* (HP) dari musuh atau ketika lawan sudah tidak dapat melakukan fase *draw* lagi.

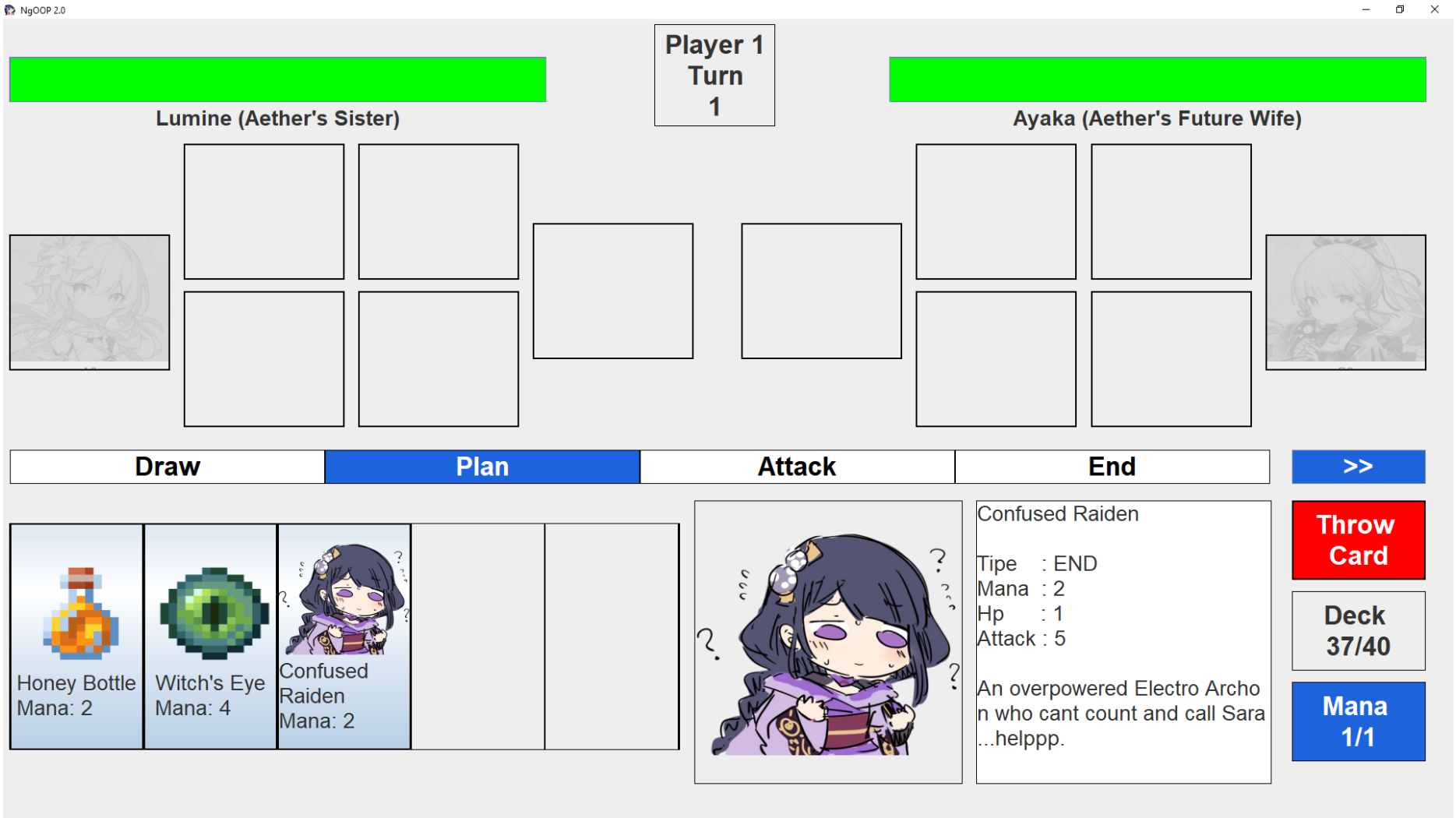
Dalam mengimplementasikan aplikasi ini, kami menggunakan bahasa pemrograman Java dengan pustaka Java Swing untuk GUI dan Gradle untuk membangun aplikasi. Paradigma pemrograman yang kami gunakan adalah pemrograman berorientasi objek. Berikut beberapa tangkapan layar dari aplikasi yang dibuat.



(Import Phase)


NgOOP 2.0
Player 1
Turn
0
Lumine (Aether's Sister)
Ayaka (Aether's Future Wife)
Cat Food
SWAP
Mana: 3
Deathly Magic
PTN
Mana: 2
Sugondese
MORPH
Mana: 6
SKIP
End
Honey Bottle
Mana: 2
Witch's Eye
Mana: 4
Confused
Raiden
Mana: 2
Deck
40/40
Mana
0/0

(Draw Phase)



(Play Phase)

2. Konsep OOP

Pada bagian ini, kami akan menjelaskan konsep pemrograman berorientasi objek yang kami implementasikan. Kelima konsep ini merupakan konsep utama yang diterapkan, meski sebenarnya ada konsep lain yang berperan, misalnya komposisi (setiap Player memiliki Board, Hand, Deck, dan seterusnya). **Perhatikan bahwa setiap kelas tidak akan ditulis nama path relatifnya—terlalu panjang. Path relatif akan dilampirkan pada Bagian 5.**

2.1. Inheritance

Konsep Inheritance digunakan pada kelas-kelas berikut:

2.1.1. Kelas abstrak Card merupakan kelas orang tua dari CharacterCard dan SpellCard.
2.1.2. Kelas abstrak SpellCard merupakan kelas orang tua dari LevelSpellCard, MorphSpellCard, PotionSpellCard, SwapSpellCard, dan ImmuneSpellCard (bonus).

Perlu diperhatikan bahwa kelas Card dan SpellCard merupakan kelas abstrak. Jadi, ada kontrak antara kelas orang tua dan kelas anak.

2.2. Interface

Konsep Interface digunakan pada kelas-kelas berikut:

2.2.1. Kelas CardContainer merupakan interface dari kelas Deck dan Hand. Artinya, Deck dan Hand merupakan penampung dari Card yang bisa diambil (*take*) dan ditambah (*add*).
2.2.2. Kelas ICardBuilder merupakan interface dari Builder Card (CharacterBulder, ImmuneBuilder, LevelSpellBuilder, MorphBuilder, PotionBuilder, SwapBuilder)
2.2.3. Kelas IStatsBuilder merupakan interface dari Builder Card yang perlu set HP dan Attack (CharacterBuilder dan PotionBuilder).
2.2.4. Kelas ITypeBuilder merupakan interface dari Builder Card yang perlu set Type Card (CharacterBuilder)

2.3. Polymorphism

Terdapat dua jenis polymorphism yang kami terapkan, yaitu *method overloading* (*compile-time polymorphism*) dan *method overriding* (*run-time polymorphism*). Berikut aplikasinya.

2.3.1. Method show() pada kelas abstrak Card diimplementasikan oleh setiap kelas anaknya: CharacterCard, LevelSpellCard, MorphSpellCard, PotionSpellCard, SwapSpellCard, dan ImmuneSpellCard. Kasus ini merupakan *run-time polymorphism*.

2.3.2. Method giveEffect() pada semua turunan kelas SpellCard mengembalikan ArrayList of Objects yang merupakan informasi-informasi yang diperlukan untuk mengubah state internal dari SummonedCharacter. Kasus ini merupakan *run-time polymorphism*.
2.3.3. Kelas Board dan Hand memiliki method add() dan add(int). Kasus ini merupakan *compile-time polymorphism*.
2.3.4. Kelas Board dan Hand memiliki method take() dan take(int). Kasus ini merupakan *compile-time polymorphism*.

2.4. SOLID

2.4.1 Single Responsibility Principle
   a. Kelas Card hanya bertugas untuk menyimpan informasi dasar yang dibaca saat permainan dimulai, sedangkan yang bertugas menyimpan adalah kelas yang menerapkan interface CardContainer.
   b. Kelas CharacterCard hanya bertugas untuk menyimpan informasi dasar yang dibaca saat permainan dimulai, sedangkan yang berfungsi untuk bertarung dalam field (kelas Board) adalah kelas SummonedCharacter.

2.4.2 Open-Closed Principle
   a. Pada kelas Card, jika kita ingin menambahkan kartu baru (*open for extension*), kita tidak perlu memodifikasi kelas Card lain (*closed for modification*). Jika ingin menambahkan kartu nonspell, kita bisa melakukan *inherit* dari kelas Card. Sebaliknya, jika ingin menambahkan kartu spell, kita bisa meng-*inherit* kelas SpellCard. Contohnya, pada program ini, kami menerapkan bonus ImmuneSpellCard. Dengan penambahan ini, kami tidak perlu mengubah kelas-kelas lain.

2.4.3 Liskov Substitution Principle
   a. Kelas CharacterCard dan kelas SpellCard (serta anak-anaknya) adalah subtipe kelas Card. Pada program kami, kelas Hand dan Deck memiliki array of Card. Kelas Card tidak mungkin ada instansiasi objeknya (kelas abstrak), maka pastilah isi dari array tersebut adalah subtipe dari Card. Dengan demikian, setiap kelas anak dari Card dapat disubstitusi dengan kelas orang tuanya sehingga memenuhi prinsip ini.

2.4.4 Interface Segregation Principle
   a. Interface ICardBuilder, ITypeBuilde, IStatsBuilder merupakan segregasi dari interface yang bertugas memberikan kontrak terhadap builder Card.

2.4.5 Dependency Inversion Principle
   a. Kelas Player melakukan dependency injection pada kelas Deck menggunakan kelas CardRepo.
   b. Kelas Hand memiliki array of Card. Ketika Card dari Hand dipindahkan ke Board, Hand akan menciptakan SummonedCharacter dengan melakukan dependency injection kelas Card ke kelas SummonedCharacter.

c. Kelas Player (high-level module) tidak bergantung pada kelas CharacterCard, MorphSpellCard, atau sejenisnya (low-level module). Namun, ada kelas abstrak Card (depend on abstraction) yang menghubungkan Player dengan kelas turunan kartu yang bisa diinstansiasi. Dengan demikian, jika ada penambahan kartu, misal ada kartu ZombieCard, maka implementasi kelas Player tidak perlu berubah.

2.5. Design Pattern

Tuliskan daftar penggunaan konsep ini di aplikasi, misalnya:

- Design pattern Repository dengan kelas repository CardRepo yang menyimpan data Card.
- Design pattern Builder untuk setiap turunan kelas Card/SpellCard. Contoh: CharacterCard memiliki builder CharacterBuilder.
- Design pattern Adapter digunakan oleh kelas Hand untuk mengonversi kelas Card menjadi SummonedCharacter.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Import Deck

Import Deck dibuat menggunakan dua buah file csv, yaitu file player1.csv dan player2.csv. Konfigurasi id card pada deck player yang ingin digunakan dapat diubah pada salah satu file tersebut yang masing-masing secara berurutan untuk player 1 dan player 2. Konfigurasi dibaca menggunakan modul CSVReader lalu diimplementasikan dalam class ImportDeck yang diimplementasikan pada aetherwars/utils/. Class Import Deck memiliki method static read yang memiliki parameter Hashmap<String,Card> yang merupakan dictionary cards yang telah dibaca sebelumnya dan parameter integer player yang menentukan konfigurasi yang digunakan player1.csv atau player2.csv untuk nilai 1 dan 2 respectively.

```
public class ImportDeck {
    private static final String PLAYER1_FILE_PATH = "card/player/player1.csv";
    private static final String PLAYER2_FILE_PATH = "card/player/player2.csv";

    public static Stack<Card> read(Map<String, Card> dict,int player) throws IOException, URISyntaxException {
        Stack<Card> cards = new Stack<>();
        File cardCSVFile;
        if(player == 1) {
            cardCSVFile = new File(Objects.requireNonNull(AetherWars.class.getResource(PLAYER1_FILE_PATH)).toURI());
        }
        else{
            cardCSVFile = new File(Objects.requireNonNull(AetherWars.class.getResource(PLAYER2_FILE_PATH)).toURI());
        }
        CSVReader cardReader = new CSVReader(cardCSVFile, "\t");
        cardReader.setSkipHeader(true);
        List<String[]> cardRows = cardReader.read();
        for(String[] row : cardRows){
            cards.push((Card) dict.get(row[0]));
        }
        return cards;
    }

}
```

Method read pada ImportDeck ini digunakan pada pembacaan konfigurasi dengan constructor overloading pada class player dan pada class Deck

```
public Player(String name, HashMap<String, Card> cdict, int player) throws IOException, URISyntaxException {
        this.name = name;
        this.hp = 80;
        this.mana = 0;
        this.board = new Board();
        this.hand = new Hand();
        this.deck = new Deck(cdict,player);
    }
```

```
Deck(HashMap<String, Card> cdict, int player) throws IOException, URISyntaxException{
        this.cards = ImportDeck.read(cdict, player);
    }
```

Sehingga, program dapat digunakan menggunakan konfigurasi dari file csv menggunakan ImportDeck maupun dirandom dari kumpulan card yang ada.

3.1.2. Spell Baru

Spell baru yang dibuat adalah ImmuneSpellCard. Cara kerja dari ImmuneSpellCard adalah memberikan immunity terhadap SummonedCharacter sehingga akan immune terhadap beberapa attack yang bersifat permanen (tidak memiliki expiration time) sehingga efek dari kartu hanya akan hilang ketika setelah banyaknya immune attack habis dari karakter tersebut (semi permanent).

Implementasi dari ImmuneSpellCard pada class ImmuneSpellCard

```java
public class ImmuneSpellCard extends SpellCard {
    int immunity;
    public ImmuneSpellCard(ImmuneBuilder builder) {
        super(builder.name, builder.desc, Type.IMMUNE, builder.imagepath, builder.mana, 0);
        this.immunity = builder.immunity;
    }

    public void show() {
        super.show();
        System.out.println("Is Immune");
    }

    public ArrayList<Object> giveEffect() {
        ArrayList<Object> effect = new ArrayList<Object>();
        effect.add(this.immunity);
        return effect;
    }

    @Override
    public String getDescription() {
        // TODO Auto-generated method stub
        String text;
        text = this.getName();
        text += "\n\nTipe       : " + this.getType();
        text += "\nMana     : " + this.getMana();
        text += "\nDuration : " + this.getDuration();
        text += "\n\n"+this.getDesc();
        return text;
    }
```

giveEffect akan mengembalikan nilai banyaknya immunity.

3.2. Bonus Kreasional

Membuat pembacaan konfigurasi dari file csv yang diinput dari GUI nama dari deck (csv).

4. Pembagian Tugas

- 13520019 / Maharani Ayu Putri Irawan
    - Pembuatan kelas Spell dan Card
- 13520034 / Bryan Bernigen
    - Pembuatan GUI dengan Swing
- 13520040 / Ng Kyle
    - Pembuatan kelas Spell dan Card
- 13520088 / Rio Alexander Audino
    - Pembuatan kelas Player
- 13520115 / Maria Khelli
    - Pembuatan kelas Board, Deck, dan Hand

5. Lampiran

5.1 Path relatif kelas

Semua class implementasi pada package com.aetherwars dengan relative path src/main/java/com/aetherwars. Penjabaran Path kelas dimulai dari relative path package, misal Path : card/Card.java memiliki relative path src/main/java/com/aetherwars/card/Card.java.

Nama Kelas	Relative Path
Card	card/Card.java
CharacterCard	card/CharacterCard.java
SpellCard	card/SpellCard.java
ImmuneSpellCard	card/ImmuneSpellCard.java
LevelSpellCard	card/LevelSpellCard.java
MorphSpellCard	card/MorphSpellCard.java
PotionSpellCard	card/PotionSpellCard.java
SwapSpellCard	card/SwapSpellCard.java
SummonedCharacter	card/SummonedCharacter.java
ICardBuilder	card/ICardBuilder.java

ITypeBuilder	card/ITypeBuilder.java
IStatsBuilder	card/IStatsBuilder.java
EmptyContainerException	exceptions/EmptyContainerException.java
FullContainerException	exceptions/FullContainerException.java
NoCardChosenException	exceptions/NoCardChosenException.java
NotCharacterCardException	exceptions/NotCharacterCardException
SpaceFilledException	exceptions/SpaceFilledException
Define	GUI/Define.java
GUI	GUI/GUI.java
Board	model/Board.java
CardContainer	model/CardContainer.java
Deck	model/Deck.java
Hand	model/Hand.java
Player	model/Player.java
CardReader	util/CardReader.java
CardRepo	util/CardRepo.java
CommandType	util/CommandType.java
CSVReader	util/CSVReader.java
ImportDeck	util/ImportDeck.java
Status	util/Status.java
Type	util/Type.java

5.2 Form Asistensi

Tugas Besar 2 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022

Kelas : 01
Nomor Kelompok : 09
Nama Kelompok : ngOOP2.0
1. 13520019 / Maharani Ayu Putri Irawan
2. 13520034 / Bryan Bernigen
3. 13520040 / Ng Kyle
4. 13520088 / Rio Alexander Audino
5. 13520115 / Maria Khelli

Asisten Pembimbing : Muhammad Fauzan Rafi Sidiq Widjonarto

1. Konten Diskusi

Class diagram dibuat di awal, sekalian biar tidak kerja dua kali.

Design pattern creational: misal untuk character card, bagusnya bagaimana?
Awalnya sudah dibuat card reader. Kalau ada card yang dibuat, diclone dari HashMap.
String apa?
String itu ID nya.
Jadi langsung dibuat semua?
Ya. Sudah dibuat objek di hashmap. Misal perlu character A. ID nya 1. Kalau begitu creational apa?
Design pattern repository. Baca semua, punya DAO. Kalau masalah oke atau nggak kayak gitu, oke oke saja. Apa gunanya character card?
Semua di character card, nanti ada container.
Saran dari desain OOP nya. Itu jadi base info buat kalau misalkan kartu-kartu. Kalau sudah aktif, jangan character card, misal active card, yang punya atribut character card. Kalau kena serang, yang kena bukan character card nya. Anggap character card itu program, active character itu proses. Character card sama active character itu beda. Class nya wrapper. Adapter atau decorator. Wrap character card, namanya active character. Yang kena spell active card. Infonya misalkan health basis, attack basis, gitu doang Kalau exp, level, masuk active character.
Soalnyaa kalau disimpan di hand nggak punya level, exp. Single responsibility.
Kenapa tidak boleh di clone?
Yang clone tidak masalah. Yang dipermasalahkan, classnya terlalu gendut, tidak single responsibility.
Hand itu card, kalau aktif, instantiate baru. Jadi hand dan active character itu terpisah.

Tugas Besar 2 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022

Kenapa tidak extend dari card?
Tidak memenuhi solid. Kan didesain sebagai card, card dan active character itu berbedaa, sesuai responsibility masing-masing. Kalau misalkan langsung bikin begini, dia langsung aktif.
Di board itu bukan card. Kalian harus jelaskan kenapa itu card. Kalau pakai desain gue tadi, bukan card, tapi yang aktif. Bisa bikin getter.
Nggak boleh desain dipaksakan hanya utk convenience. Cari cara bedakan card dan active card. Kalau semua card disatukan. Exp, swap duration, seharusnya tidak ada di card. Kalau masih ada, infonya redundan.
Seperti pokemon.

Kalau di java clone bagaimana?
Maksudnya deepcopy ya? Lupa. Cari di internet. Ada abstraksinya. Tidak ada deepcopy.

Kalau active character diimplementasikan, instansiasi bagaimana?
Dependency injection jadinya. Dependency injection itu kalau punya atribut, jangan bikin manual, pakai constructor. Kalau bikin constructor, contoh active character punya attr char card, nggak boleh bikin new, mending masukin objekny ake konstruktornya, this.charactercard = character card yg di parameternya. Jangan create di dalam constructor. Itu bad design. Kalau mau new dari luar. Namanya dependency injection, soalnya depend terhadap yg dimasukkan.

Yang vector tidak masalah, karena collection bukan objek baru. Pakai pattern builder soalnya banyak attribut nya. Lebih readable, kalau ada null tidak perlu satu-satu. Lebih readable dan robust desainnya. Tidak perlu kotor nulis null. Bisa beri default function, misal none atau nol. Kalau dipisah, tidak terlalu banyak, tidak perlu

Companion class apa?
Class di dalam class, nested class, daripada bikin class card builder.

Semua atribut tidak perlu setter getter nggak sih?
Iya, Cuma ada konvensi OOP harus ada setter getter untuk semua atribut.
Itu kan melanggar private atribut itu, harusnya Cuma bisa di get aja nggak boleh di set.
Kalau begitu, workaround nya getter nya private.
Melanggar info encapsulation. Boleh protected, pastikan akses anak-anaknya. Sangat tergantung use case nya.

Hati hati chaos diagram. Versioning aja. Pakai draw.io atau lucid chart. Soalnya makin lama program makin besar. Kalau ada tool buat dari Java nya boleh dipakai.

Tugas Besar 2 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022

2. Screenshot Bukti