

Kode Kelompok : OPO

Nama Kelompok : Objectioners

1. 13521047 / Muhammad Equilibrie Fajria
2. 13521048 / M. Farrel Danendra Rachim
3. 13521098 / Fazel Ginanda
4. 13521105 / Haidar Hamda
5. 13521141 / Edia Zaki Naufal Ilman

Asisten Pembimbing : Hafid Abi Daniswara

## 1. Diagram Kelas

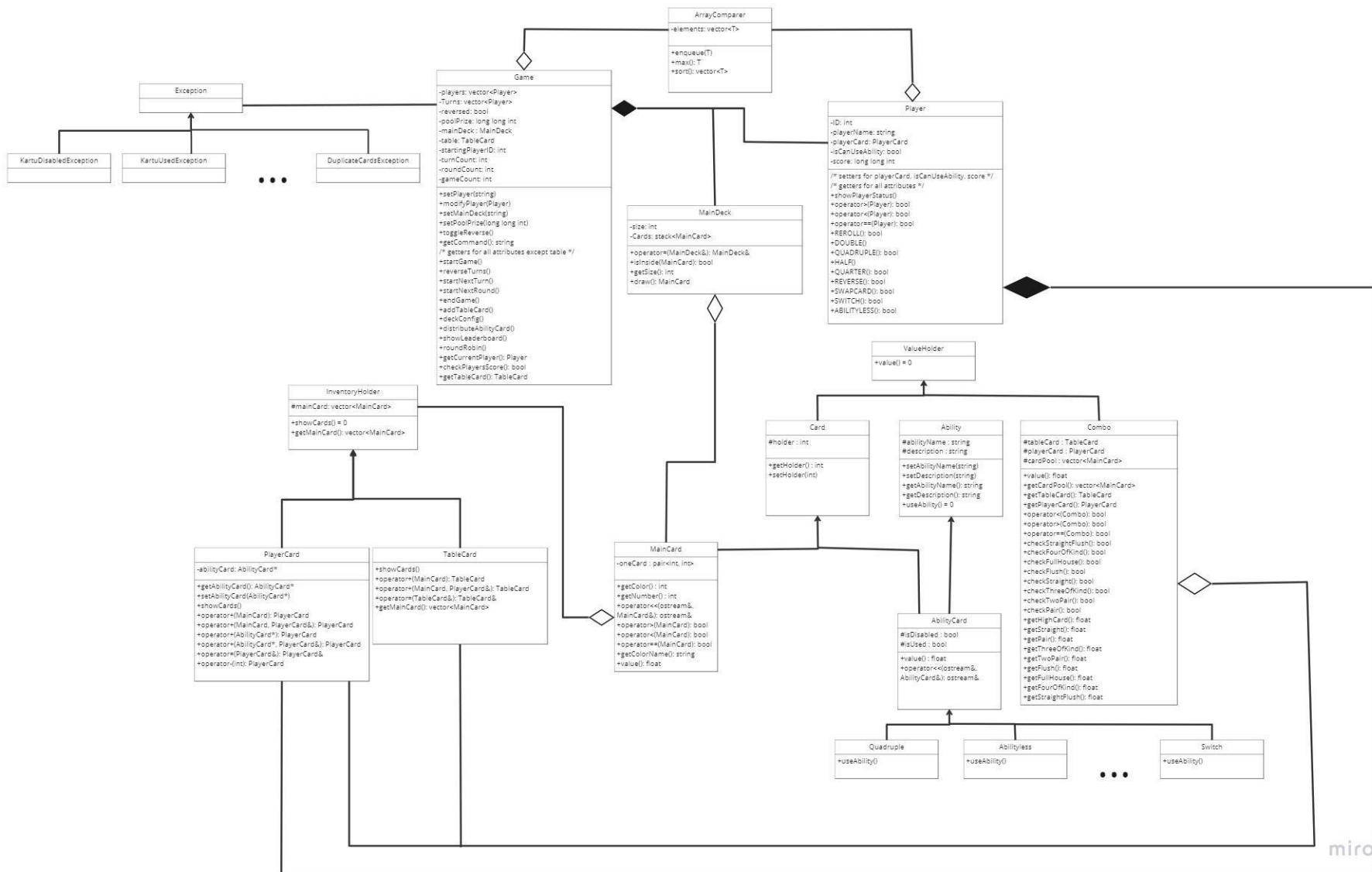
Dalam bab ini, kami menggunakan diagram kelas dengan notasi UML. Kami memilih desain diagram kelas dengan bahasa ini karena notasi UML memiliki berbagai macam notasi seperti indikasi sifat atribut dan relasi antar kelas yang dapat menjelaskan desain program kami dengan lebih baik. Kami sengaja tidak memasukkan constructor, copy constructor, dan destructor dalam diagram ini, serta setter dan getter bagi beberapa kelas tertentu untuk menyederhanakan penampilan diagram secara keseluruhan - mereka diimplementasi dengan cara yang telah dipahami secara umum. Selain itu, kami juga hanya memasukkan metode virtual bagi kelas yang meng-override metode tersebut. Misalnya, pada kelas Card, metode value() tidak dimasukkan karena metode tetap bersifat virtual dari kelas di atasnya, ValueHolder, namun pada kelas MainCard dan AbilityCard metode value() dimasukkan karena pada kelas-kelas tersebut value() meng-override metode di atasnya.

Dari diagram kelas kami, dapat dilihat bahwa desain diagram kelas ini memiliki beberapa kelebihan. Salah satunya yaitu implementasi kelas Game yang membuat abstraksi saat main program dijalankan. Dengan kata lain, program utama tidak perlu mengetahui mekanisme kompleks yang dilakukan Game dan kelas-kelas lainnya. Selain itu, konsep inheritance dengan menginisialisasi sebuah metode dengan nilai nol pada base class seperti pada InventoryHolder dan ValueHolder dapat membuat implementasi yang berbeda-beda untuk tiap kelas yang mewarisinya. Namun, selain kelebihan yang disebutkan, terdapat beberapa kelemahan pada desain diagram kelas ini. Kelemahan pertama yaitu pembagian tugas antara tiap anggota kelompok menjadi sulit. Hal ini dikarenakan semua class yang ada pada diagram class saling membutuhkan (depends) class lain, sehingga tidak mungkin pengimplementasian class-class dilakukan murni per-individu. Untuk menerapkan class-class sangat diperlukan komunikasi dan koordinasi yang baik antar anggota kelompok. Kelemahan kedua, yaitu desain program lebih rumit. Desain menggunakan OOP perlu ada pembagian yang detail untuk tiap class-classnya. Atribut dan metode harus dapat dienkapsulasi. Isi class harus sebisa mungkin diurus oleh class itu sendiri. Kelemahan yang ketiga yaitu untuk mengetahui suatu metode/mengakses suatu

atribut harus menelusuri class-class parentnya (tidak langsung pakai). Hal ini dapat terjadi jika terjadi inheritance berturut-turut yang banyak (a diturunkan ke b, b diturunkan ke c, c diturunkan ke d, dst).

Selama pembuatan desain ini, kami menghadapi berbagai kendala. Pertama, mengingat alur permainan yang lumayan kompleks (termasuk implementasi ability, ganti round, dan perbandingan kombinasi) kami tidak memiliki pilihan selain memuat banyak metode dalam sebuah kelas tertentu sehingga organisasi kelas tersebut serta hubungannya dengan kelas-kelas lain makin sulit. Selain itu, kami kebingungan di kelas mana kami perlu mengimplementasikan function dan operator overloading yang sesuai dengan peran masing-masing kelas.

Untuk tampilan diagram kelas yang lebih jelas: [https://miro.com/app/board/uXjVMe3oW5E=?share\\_link\\_id=371568491760](https://miro.com/app/board/uXjVMe3oW5E=?share_link_id=371568491760)



## 2. Penerapan Konsep OOP

### 2.1. Inheritance & Polymorphism

Inheritance atau Pewarisan/Penurunan adalah konsep pemrograman dimana sebuah class dapat ‘menurunkan’ property dan method yang dimilikinya kepada class lain. Konsep inheritance digunakan untuk memanfaatkan fitur ‘code reuse’ untuk menghindari duplikasi kode program. Konsep inheritance membuat sebuah struktur atau ‘hierarchy’ class dalam kode program. Class yang akan ‘diturunkan’ bisa disebut sebagai class induk (parent class), super class, atau base class. Sedangkan class yang ‘menerima penurunan’ bisa disebut sebagai class anak (child class), sub class, derived class atau heir class ([Inheritance – School of Information Systems \(binus.ac.id\)](https://www.sis.binau.ac.id/~inher.htm)).

Polimorfisme terbagi menjadi dua suku kata yaitu, Poly yang berarti banyak dan Morfisme yang berarti bentuk. Dalam ilmu sains, Polimorfisme (polymorphism) adalah sebuah prinsip dalam biologi di mana organisme atau spesies memiliki banyak bentuk serta tahapan (stages). Prinsip tersebut diterapkan juga pada bahasa Java. Polimorfisme dalam OOP merupakan sebuah konsep OOP di mana class memiliki banyak “bentuk” method yang berbeda, meskipun namanya sama. Maksud dari “bentuk” adalah isinya yang berbeda, namun tipe data dan parameternya berbeda ([Pengertian Polimorfisme Dalam Pemrograman Java - Dicoding Blog](https://www.dicoding.com/id/blog/pengertian-polimorfisme-dalam-pemrograman-java/)).

Untuk membuat beberapa class pada program ini digunakan konsep inheritance. Class-class yang menggunakan konsep inheritance antara lain AbilityCard, MainCard, PlayerCard, dan TableCard. AbilityCard merupakan turunan dari class Card dan Ability, MainCard merupakan turunan dari class Card, PlayerCard merupakan turunan dari class InventoryHolder, dan TableCard merupakan turunan dari class InventoryHolder. Dengan menggunakan inheritance, atribut umum (seperti holder, abilityName, description pada AbilityCard) dan metode umum yang digunakan pada beberapa class anak cukup dideklarasikan sekali saja pada class induk. Dengan begini, kode program yang sama dapat direuse pada setiap class anak dapat, sehingga dapat terhindar dari duplikasi yang dapat mengakibatkan ketidakkonsistenan dan ketidaksinkronan.

Cuplikan kode berikut merupakan bukti bahwa beberapa kelas tersebut melakukan inheritansi.

```
class AbilityCard : public Card, public Ability
```

```
class MainCard : public Card
```

```
class PlayerCard : public InventoryHolder
```

```
class TableCard : public InventoryHolder
```

Sementara itu, konsep polymorphism digunakan pada class Ability dan ValueHolder. Ability diturunkan menjadi class AbilityCard yang diturunkan lagi menjadi tujuh class, yaitu ReRoll, Abilityless, Quadruple, Quarter, ReverseDirection, Swap, dan Switch. Setiap class dari tujuh class anak tersebut mengimplementasikan metode useAbility(). Sedangkan ValueHolder diturunkan class menjadi Card dan Combo. Metode yang diimplementasikan dari valueHolder yaitu value(). Karena setiap ability dari abilityCard (ReRoll, Quadruple, Quarter, dst) memiliki useAbility() yang berbeda-beda, maka pengimplementasianya juga harus berbeda-beda sesuai dengan classnya. Begitu juga dengan value() pada ValueHolder. Oleh karena itu, penggunaan polymorphism diperlukan untuk mengimplementasikan metode yang implementasinya berbeda-beda setiap class, sehingga kode implementasi dapat terenkapsulasi untuk setiap class dan pada base class hanya terdapat satu saja metode yang dijadikan pure virtual.

Berikut cuplikan kode useAbility() dan value() yang diinisialisasi nol pada base class masing-masing.

```
virtual void useAbility() = 0;
```

```
virtual float value()=0;
```

## 2.2. Method/Operator Overloading

Function overloading adalah sebuah kasus pada bahasa C++ di mana dua atau lebih fungsi dapat memiliki nama yang sama jika angka dan/atau tipe argumen di dalamnya berbeda, sedangkan operator overloading yaitu mendefinisikan ulang cara kerja operator agar operator dapat disesuaikan dengan suatu aksi pada kelas.

Function overloading diimplementasikan di kelas PlayerCard, di mana fungsi operator (+) didefinisikan empat kali, masing-masing dengan tipe parameter yang berbeda. Berikut implementasinya.

```
PlayerCard PlayerCard::operator+(MainCard mc)
{
    PlayerCard pc;
    pc.setAbilityCard(this->abilityCard);
    pc.mainCard.push_back(mc);
    return pc;
}

PlayerCard operator+(MainCard mainCard, const PlayerCard &playerCard)
{
    PlayerCard pc;
    pc.setAbilityCard(playerCard.abilityCard);
    pc.mainCard = playerCard.mainCard;
    pc.mainCard.push_back(mainCard);
    return pc;
}

PlayerCard PlayerCard::operator+(AbilityCard *ac)
{
    PlayerCard pc;
    pc.mainCard= this->mainCard;
    pc.abilityCard = ac;
    return pc;
}

PlayerCard operator+(AbilityCard *abilityCard, const PlayerCard &playerCard)
{
    PlayerCard pc;
    pc = playerCard;
    pc.setAbilityCard(abilityCard);
    return pc;
}
```

Operator overloading digunakan di beberapa kelas pada program yaitu TableCard, PlayerCard, MainCard, Combo, AbilityCard, dan Player. Kelas PlayerCard menggunakan operator overloading pada (-) untuk memudahkan pengurangan elemen pada buffer atau menarik sebuah kartu. Terdapat operator overloading (<<) pada kelas AbilityCard dan MainCard untuk memudahkan output berupa informasi kartu pada

layer dengan menggunakan ostream daripada membuat metode baru. Operator overloading (+) pada kelas PlayerCard dan TableCard berguna untuk menambahkan kartu. Operator overloading (<, >, ==) digunakan pada kelas Combo dan Player untuk membandingkan dua buah kombo atau dua buah Player.

Berikut contoh implementasi operator overloading perbandingan pada MainCard:

```
bool MainCard::operator>(MainCard other) {
    return this->value()>other.value();
}

bool MainCard::operator<(MainCard other) {
    return this->value()<other.value();
}

bool MainCard::operator==(MainCard other) {
    return this->value()==other.value();
}
```

Berikut implementasi operator overloading (-) pada PlayerCard.

```
PlayerCard PlayerCard::operator-(int num){  
    PlayerCard pc;  
    pc.mainCard= this->getMainCard();  
    pc.setAbilityCard(this->getAbilityCard());  
    for (int i = 0; i < num; ++i) {  
        if (this->getMainCard().size()==0){  
            return pc;  
        }  
        pc.mainCard.pop_back();  
    }  
    return pc;  
}
```

Berikut implementasi operator overloading (<<) pada MainCard.

```
ostream &operator<<(ostream &os, MainCard mc) {  
    //os << "Holder : " << mc.getHolder() << endl;  
    os << "Color : " << mc.getColorName() << endl;  
    os << "Number : " << mc.getNumber() << endl;  
    return os;  
}
```

## 2.3. Template & Generic Classes

Template dan generic class dalam bahasa C++ berguna untuk mengoper tipe data sebagai parameter agar kita tidak perlu menulis kode yang sama untuk tipe data yang berbeda. Dalam program ini, kami menerapkan konsep template dan generic class dalam kelas ArrayComparer yang nantinya digunakan pada kelas Game dan kelas Combo. ArrayComparer yang kami buat berupa sebuah vektor yang diimpor dari STL. Dengan template ini, ArrayComparer dapat mencari nilai maksimum sebuah elemen T dalam vektor dan mengurutkan elemen-elemen T secara membesar. Kami mengimplementasikan konsep ini agar ArrayComparer dapat digunakan dengan tipe elemen yang bervariasi - dalam kasus ini untuk objek Card dan Player. Implementasi ini juga dibuat dalam satu file header agar kami tidak perlu mendefinisikan tipe data tiap kali menggunakan kelas ArrayComparer.

Berikut implementasi template dan generic class pada ArrayComparer.

```
template <class T>
class ArrayComparer {
private:
    vector<T> elements;
public:
    ArrayComparer(vector<T> elements){
        this->elements=elements;
    }
    ~ArrayComparer() {}
    void enqueue(const T q){
        this->elements.push_back(q);
    }
    T max() {
        int max = this->elements.at(0);
        for (int i=1; i<this->nEff; i++) {
            if (this->elements.at(i) > max) {
                max = this->elements.at(i);
            }
        }
        return max;
    }
    vector<T> sort(){
        // cout << "MULAI\n";
        for (int i = 0; i < this->elements.size(); ++i) {
            for (int j = 0; j < this->elements.size() - i-1; ++j) {
                if (this->elements.at(j) > this->elements.at(j+1)){
                    T temp= this->elements.at(j);
                    this->elements.at(j)= this->elements.at(j+1);
                    this->elements.at(j+1)=temp;
                }
            }
        }
        // cout << "SELESAI\n";
        return this->elements;
    }
}
```

## 2.4. Exception

Kami juga menerapkan konsep ~~HFSWIRQ~~ dengan konsep polimorfisme untuk mengatasi berbagai kasus error yang dapat terjadi saat permainan berlangsung. Kami membuat berbagai kelas yang diturunkan dari kelas basis exception sesuai kasus errornya, dengan setiap kelas memiliki metode what() untuk mengoper pesan error kepada pengguna. Program akan melakukan aksi ~~WURZ~~ saat terjadi sebuah error pada

kelas exception dengan error yang sesuai, sehingga exception dapat di-~~FM~~ Dengan exception, program tetap berjalan dengan baik, dan pemrogram dapat mengetahui letak error dalam programnya agar ~~GHEXJJLQJ~~ jauh lebih mudah.

Berikut implementasi kelas-kelas exception.

```

struct KartuDisabledException : public exception {
    const char* what() const throw() {
        return "LOL kena disabled";
    }
};

struct KartuUsedException : public exception {
    const char* what() const throw() {
        return "Tingkatkan literasi! udah dipake kartunya";
    }
};

struct InvalidCommandException : public exception {
    const char* what() const throw() {
        return "Perintah invalid! Silahkan coba lagi!";
    }
};

struct InvalidChoiceException : public exception {
    const char* what() const throw() {
        return "Pilihan invalid! Silahkan coba lagi!";
    }
};

struct FileNotFoundException : public exception {
    const char* what() const throw() {
        return "File tidak ditemukan atau tidak dapat dibuka! Silahkan coba lagi!";
    }
};

struct InvalidFileNotFoundException : public exception {
    const char* what() const throw() {
        return "File tidak ditemukan atau tidak dapat dibuka! Silahkan coba lagi!";
    }
};

struct InvalidCardException : public exception {
    const char* what() const throw() {
        return "Terdeteksi penulisan kartu invalid pada file! Silahkan coba lagi!";
    }
};

struct TooManyCardsException : public exception {
    const char* what() const throw() {
        return "Jumlah kartu lebih dari 52 kartu! Silahkan coba lagi!";
    }
};

struct TooFewCardsException : public exception {
    const char* what() const throw() {
        return "Jumlah kartu kurang dari 52 kartu! Silahkan coba lagi!";
    }
};

struct DuplicateCardsException : public exception {
    const char* what() const throw() {
        return "Terdeteksi duplikat kartu pada file! Silahkan coba lagi";
    }
};

```

dan berikut penerapan exception-nya di Main.cpp:

```
while(flag){  
    try{  
        string choice;  
        cout << "Apakah ingin bermain lagi?\n1. Main lagi\n2. Exit\n";  
        cin >> choice;  
  
        //Error  
        if(choice.compare("1") != 0 && choice.compare("2") != 0){  
            throw InvalidChoiceException();  
        }  
  
        flag = false;  
        if(choice.compare("2") != 0){  
            play = false;  
        }  
    }catch(InvalidChoiceException err){  
        cout << err.what() << endl;  
    }  
}
```

## 2.5. C++ Standard Template Library

Standard Template Library (STL) adalah sebuah pustaka dalam bahasa pemrograman C++ yang membantu pengorganisasian kode dalam suatu program dan mempercepat penggerjaan program dengan memanfaatkan berbagai struktur data, fungsi, dan kelas dalam sebuah

jenis STL. Kami memilih empat jenis STL untuk memudahkan penggerjaan kami: vektor, pair, map, dan stack, masing-masing memiliki peran tersendiri dalam permainan kartu ini.

### 2.5.1. Vektor

Vektor berperan sebagai ~~FRONTIER~~ untuk menyimpan MainCard yang digunakan dan Player yang berpartisipasi selama permainan. STL ini digunakan untuk mengorganisasikan atribut yang terdapat pada kelas InventoryHolder. Kelas-kelas yang diturunkan dari kelas InventoryHolder yaitu PlayerCard dan TableCard juga menggunakan vektor dalam beberapa method yang dibutuhkan untuk memenuhi tanggung jawabnya. Kami menggunakan vektor dari STL sebagai ganti array default untuk pengelolaan elemen kartu yang lebih dinamis dan praktis (misal iterasi vektor lebih sederhana dibanding iterasi array secara manual) dan tingkat kesalahan yang lebih kecil (misal array biasa pada umumnya memiliki batas kapasitas tertentu yang akan membuat penanganan lebih kompleks, hal ini tidak terjadi pada vektor).

Berikut salah satu implementasi vektor MainCard dalam kelas Combo:

```
bool Combo::checkStraight(){
    vector<MainCard> sorted= (new ArrayComparer<MainCard>(this->cardPool))->sort();
    for (int i = sorted.size()-1; i >= 4; i--){
        for (int j = i-1; j >=3; j--) {
            if (sorted.at(i).getNumber()==sorted.at(j).getNumber()+1){
                for (int k = j-1; k >=2 ; k--) {
                    if (sorted.at(j).getNumber()==sorted.at(k).getNumber()+1){
                        for (int l = k-1; l >=1 ; l--) {
                            if (sorted.at(k).getNumber()==sorted.at(l).getNumber()+1{
                                for (int m = l-1; m >=0; m--) {
                                    if (sorted.at(l).getNumber()==sorted.at(m).getNumber()+1){
                                        return true;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return false;
}
```

Berikut salah satu implementasi vektor Players dalam kelas Game:

```
//Reverse turns (only affects remaining players haven't played this turn)
void Game::reverseTurns(){
    vector<Player> temp = Turns;
    int len = Turns.size();
    for(int i = 0; i < Turns.size(); i++){
        Turns[i] = temp[len - 1];
        len--;
    }
}
```

## 2.5.2. Pair

Pair berperan sebagai penyimpan informasi berupa kombinasi warna dan angka dari sebuah MainCard dengan warna elemen pertama dan angka elemen kedua. Tiap warna akan dienumerasi terlebih dahulu menjadi sebuah angka: Hijau = 0, biru = 1, kuning = 2, dan merah = 3. Kami lebih memilih menggunakan pair daripada membuat atribut tersendiri untuk angka dan warna untuk memudahkan akses informasi terhadap angka dan warna kartu tertentu, serta membuat identitas tiap kartu menjadi jelas dan unik dibandingkan kartu lain.

Berikut dua implementasi constructor pair dalam MainCard.

```
MainCard::MainCard() : Card() {
    this->oneCard = make_pair(0, 0);
}
```

```
MainCard::MainCard(int _color, int _number) : Card() {
    this->oneCard = make_pair(_color, _number);
}
```

## 2.5.3. Stack

Stack berperan sebagai kontainer MainDeck yang berisi kumpulan MainCard. Kami memilih struktur data ini karena tumpukan kartu mengikuti konsep Stack, yakni elemen yang dapat diakses hanyalah elemen di puncak tumpukan kartu. Operasi yang dapat dilakukan pada MainDeck hanyalah push dan pop. Misalnya, dalam program kami, metode draw() memanfaatkan metode pop().

Berikut salah satu implementasi Stack di MainDeck.

```
MainCard MainDeck::draw(){
    MainCard drew = Cards.top();
    cout << "Color : " << drew.getColorName() << " number: " << drew.getNumber() << endl;
    Cards.pop();
    size--;
    return drew;
}
```

## 2.6. Konsep OOP lain

Dalam pengerjaan tugas ini, dimanfaatkan konsep DEWFEDMH FOW Konsep kelas abstrak diterapkan pada kelas InventoryHolder. Berdasarkan analisis yang dilakukan terhadap spesifikasi tugas besar ini, dibutuhkan suatu kelas untuk menyatakan kelas PlayerCard dan TableCard dalam suatu kelas yang lebih umum. Alasannya adalah dua kelas tersebut memiliki kesamaan, yaitu berperan sebagai kelas yang merepresentasikan kartu yang sedang dimainkan dalam suatu instansiasi waktu atau giliran tertentu. Selain itu, PlayerCard dan TableCard juga memiliki kesamaan dalam hal atribut yaitu MainCard. Oleh karena itu dibutuhkan suatu kelas untuk menampung kesamaan-kesamaan tersebut, yaitu kelas InventoryHolder.

Kelas InventoryHolder termasuk kelas abstrak dikarenakan memiliki sebuah method yang bersifat SXUH YLWDOyaitu showCards. Method tersebut bertugas untuk menampilkan kartu-kartu yang dimiliki oleh pemain maupun kartu yang berada pada WEOI Akan tetapi, langkah-langkah atau implementasi dari tugas tersebut berbeda tergantung objeknya yang dalam hal ini adalah PlayerCard dan TableCard. Oleh karena itu, metode showCards dideklarasikan sebagai method yang bersifat SXUH YLWDOarena perilaku dari method tersebut hanya terdefinisi apabila objek yang menginvokasinya bersifat konkret bukan abstrak.

Selain penggunaan konsep DEWFEDMVH FOW digunakan juga konsep FRP SRVIRQ Konsep FRP SRVIRQditerapkan pada kelas Combo. Kelas Combo yang berperan dalam pengujian kombinasi kartu berserta penghitungan nilai dari kombinasi tersebut memiliki tiga atribut, yaitu tableCard, playerCard, dan cardPool. Dua atribut pertama bertipe PlayerCard dan TableCard. Alasan dari komposisi ini adalah pengujian serta penghitungan nilai combo memerlukan dua objek tersebut, yaitu kartu yang dimiliki oleh pemain serta kartu yang berada di meja. Sehingga konsep FRP SRVIRQdianggap tepat dalam menyelesaikan persoalan tersebut.

Berikut adalah header file abstract base class InventoryHolder.

```
#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP

#include <iostream>
#include <vector>
#include "MainCard.hpp"
#include "Ability_Cards/AbilityCard.hpp"
using namespace std;

class InventoryHolder
{
protected:
    vector<MainCard> mainCard;

public:
    InventoryHolder() {}
    InventoryHolder(const InventoryHolder &) {}
    void setMainCards(vector<MainCard>);
    virtual ~InventoryHolder() {}
    virtual void showCards() = 0;
    virtual vector<MainCard> getMainCard() const;
};

#endif
```

### 3. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
-------------------------	-------------	--------

Ability, AbilityCard	13521047	13521047
MainDeck	13521098	13521098
ValueHolder, MainCard	13521048	13521048
Player	13521105	13521105
ArrayComparer	13521048	13521048
Card	13521141	13521141
Combo	13521105, 13521048	13521105, 13521048
InventoryHolder, PlayerCard, TableCard	13521098	13521098
Game	13521048, 13521141, 13521105	13521048, 13521141, 13521105
Exception	13521047	13521047
Main.cpp	13521047, 13521141, 13521105	13521048, 13521141, 13521098

#### 4. Lampiran



Link repository: [https://github.com/Breezy-DR/Tubes1\\_OOP\\_Objectioners](https://github.com/Breezy-DR/Tubes1_OOP_Objectioners)

Kode Kelompok : OPO

Nama Kelompok : Objectioners

1. 13521047 / Muhammad Equillibrie Fajria
2. 13521048 / Muhammad Farrel Danendra Rachim
3. 13521098 / Fazel Ginanda
4. 13521105 / Haidar Hamda
5. 13521141 / Edia Zaki Naufal Ilman

Asisten Pembimbing : Kak Abi

## 1. Konten Diskusi

- Table card: Semua bisa lihat. Kombinasi terdiri dari kartu player dan kartu table (tidak bisa kartu player saja)
- Apa poin kombinasi akan digabung dengan poin hadiah untuk pemenang, dan apakah poin kombinasi ini akan diakumulasi untuk masing2 pemain untuk permainan-permainan selanjutnya? Tidak, nilai kombinasi akan dimulai dari awal lagi
- Bagaimana jumlah kartu yang dimiliki pemain bisa mencapai 5 buah? Padahal pemain hanya memiliki 2 buah kartu dan jumlahnya tidak berubah sepanjang sebuah permainan. Dua buah kartu tersebut akan dibandingkan dengan table card dan akan diperoleh kombinasi
- Bagaimana jika kombinasi seperti pair hanya dapat diperoleh dari player card dan bukan dari table card? Tidak bisa, karena table card harus dimasukkan dalam kombinasi.

## 2. Tindak Lanjut

- Membagi tugas masing-masing kelompok
- Memahami kelas-kelas apa saja yang perlu diimplementasikan
- Menyetup CLI Linux

### 3. Screenshot Bukti

The screenshot shows a Google Docs page with the following content:

**Pengumpulan**

1. Softcopy laporan dan source code program dikumpulkan pada [LINK FORM](#). Submission **ditutup hari Kamis, 16 Maret 2023, 22:10 WIB**. Di luar waktu pengumpulan tersebut, deliverables tidak akan diterima.
- Untuk penyusunan Laporan Tugas Besar, akan disediakan template laporan. Laporan dikumpulkan dalam format pdf.
- Spesifikasi pengumpulan yaitu:
  - 1 kelompok 1x pengumpulan saja!** Jika ada revisi, silakan submit ulang, akan dipakai submissi paling terakhir.
  - File source code (semua modul dan program utama) dikompres ke dalam zip.
  - Hapus semua file *executable* atau *binary* dari zip pengumpulan.
  - File diberi nama "IF2210\_TB1\_XXX.zip" dengan penjelasan:
    - XXX: Kode Unik Kelompok (ditulis dalam 3 huruf) yang ada pada Sheet Kelompok
    - Format untuk nama PDF Laporan sama seperti nama zip, **Ekstensi saja yang diubah menjadi .pdf**
  - Contoh : "IF2210\_TB1\_XYZ.zip" adalah file untuk kelompok XYZ.

**Penilaian**

Penilaian Tugas Besar 1 akan dilakukan oleh asisten sesuai kriteria yang tertera pada dokumen Spesifikasi Tugas Besar 1.

Secara umum, penilaian akan sama seperti tugas besar sebelumnya (Algoritma Struktur Data) dengan mensimulasikan *test case* dari Asisten. Pastikan hal-hal krusial yang dilakukan untuk simulasi tidak rusak. Contoh: Membaca urutan kartu dari *deck*.

Pastikan sebelum pengumpulan, submisi yang kalian kumpulkan sudah benar. Asisten berhak untuk tidak menilai apabila program tidak jalan meskipun kesalahan pengumpulan sesuai prosedur. **Pastikan juga program sudah memenuhi ketentuan yang sudah dituliskan diatas**