

Kode Kelompok : HPC

Nama Kelompok : HiperC++

1. 13521118 / Ahmad Ghulam Ilham

2. 13521152 / Muhammad Naufal Nalendra

3. 13521158 / Muhammad Dhiwaul Akbar

4. 13521159 / Sulthan Dzaky Alfaro

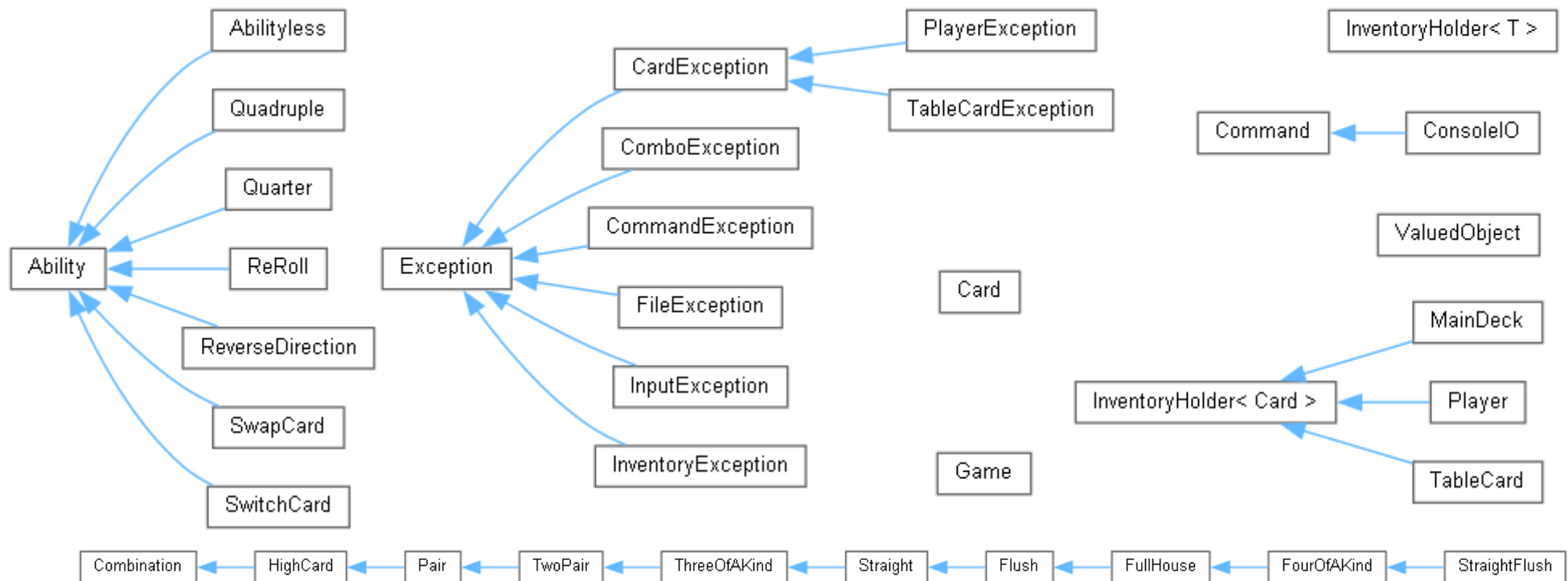
5. 13521166 / Mohammad Rifqi Farhansyah

6. 13521169 / Muhammad Habibi Husni

Asisten Pembimbing : 13519105 / Widya Anugrah Putra

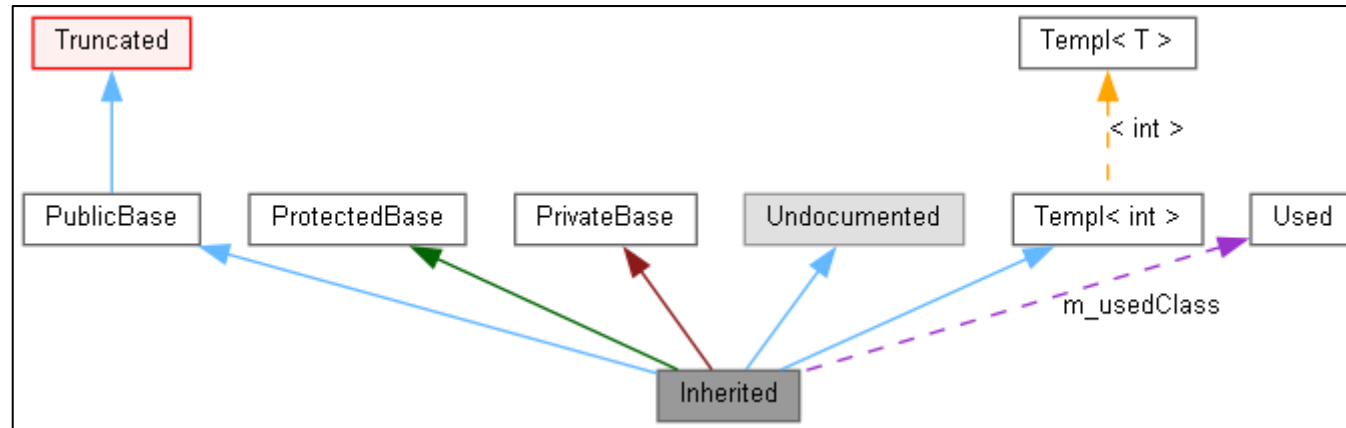
1. Diagram Kelas

a. Class Hierarchy



Gambar 1.1 Class Hierarchy untuk Permainan Kartu ala Kerajaan Permen

b. Class Diagram



Gambar 2.2 Graph Legend

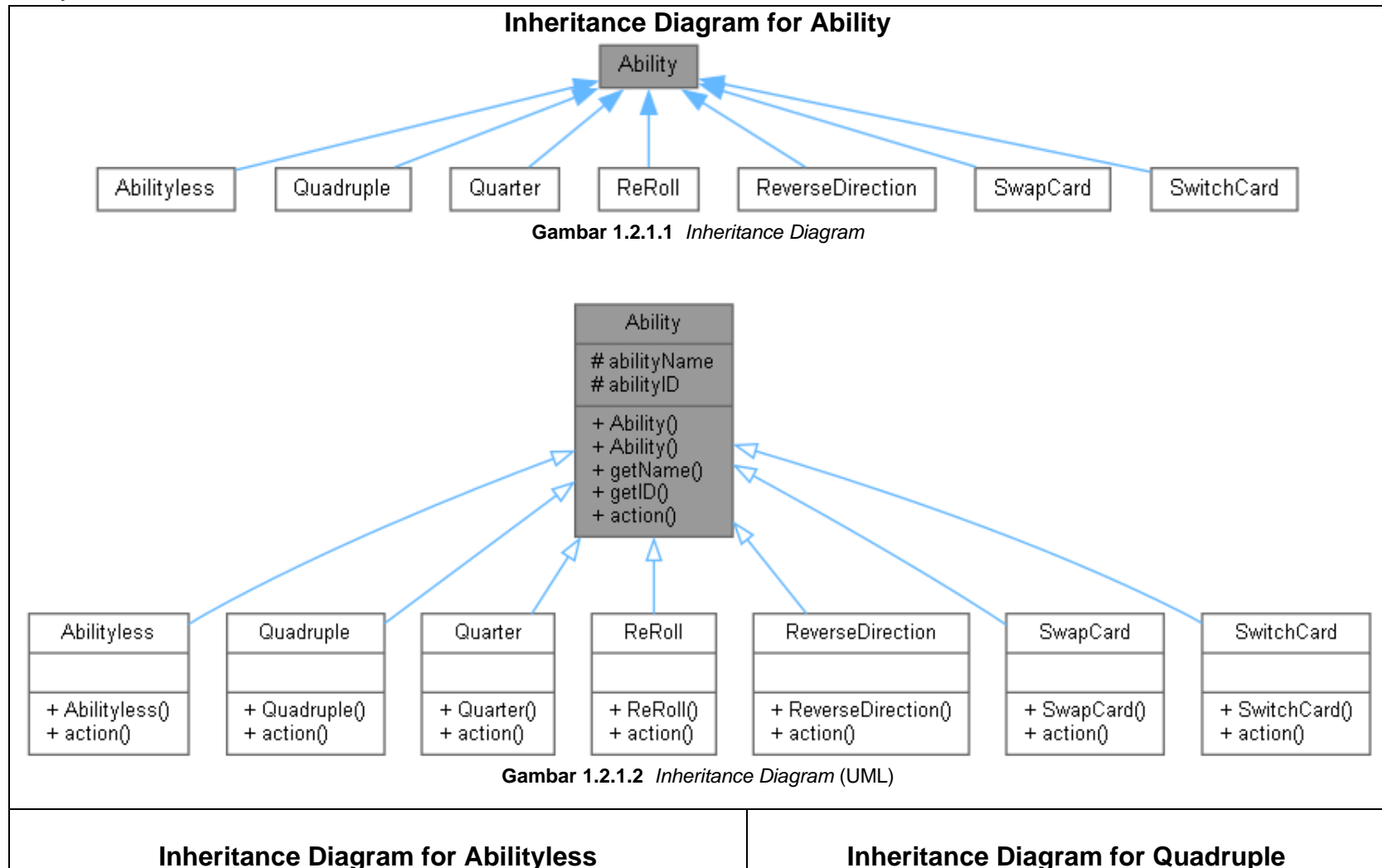
The boxes in the above graph have the following meaning:

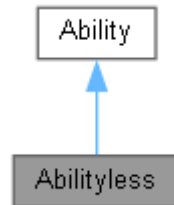
- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a gray border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class for which not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

The arrows have the following meaning:

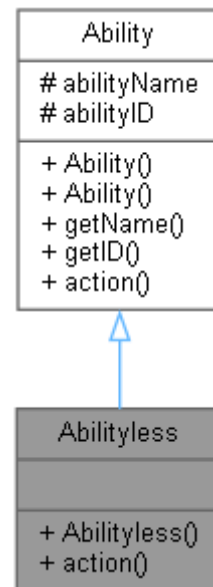
- A blue arrow is used to visualize a public inheritance relation between two classes.
- A dark green arrow is used for protected inheritance.
- A dark red arrow is used for private inheritance.
- A purple dashed arrow is used if a class is contained or used by another class. The arrow is labelled with the variable(s) through which the pointed class or struct is accessible.
- A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labelled with the template parameters of the instance.

i. Ability



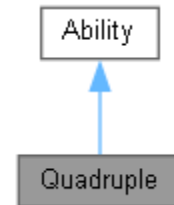


Gambar 1.2.1.3 *Inheritance Diagram*

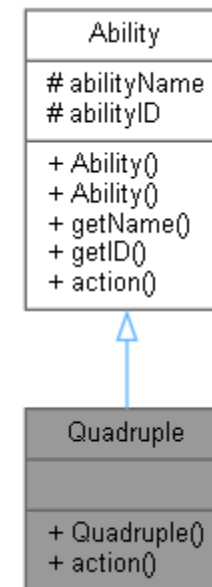


Gambar 1.2.1.4 *Inheritance Diagram (UML)*

Inheritance Diagram for Quarter



Gambar 1.2.1.5 *Inheritance Diagram*

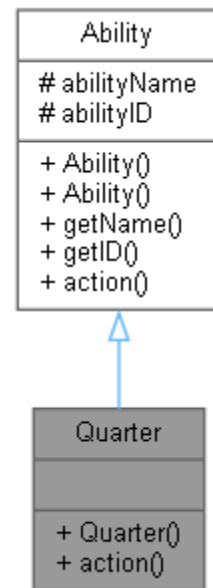


Gambar 1.2.1.6 *Inheritance Diagram (UML)*

Inheritance Diagram for ReRoll



Gambar 1.2.1.7 *Inheritance Diagram*

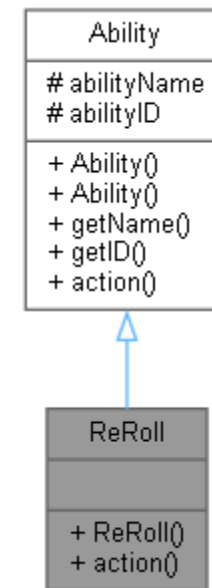


Gambar 1.2.1.8 *Inheritance Diagram (UML)*

Inheritance Diagram for ReverseDirection

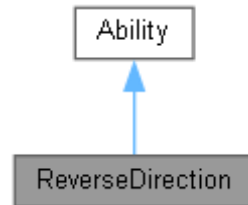


Gambar 1.2.1.9 *Inheritance Diagram*

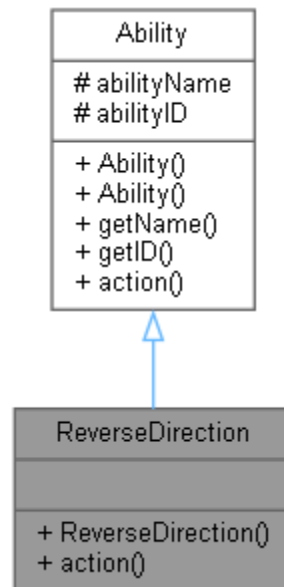


Gambar 1.2.1.10 *Inheritance Diagram (UML)*

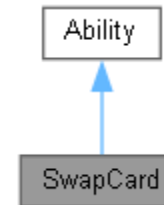
Inheritance Diagram for Swap



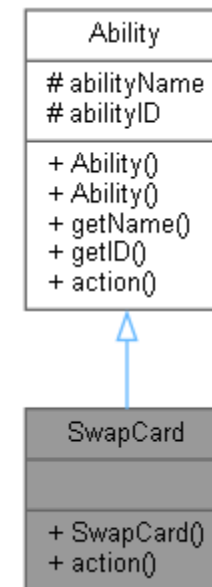
Gambar 1.2.1.11 *Inheritance Diagram*



Gambar 1.2.1.12 *Inheritance Diagram (UML)*

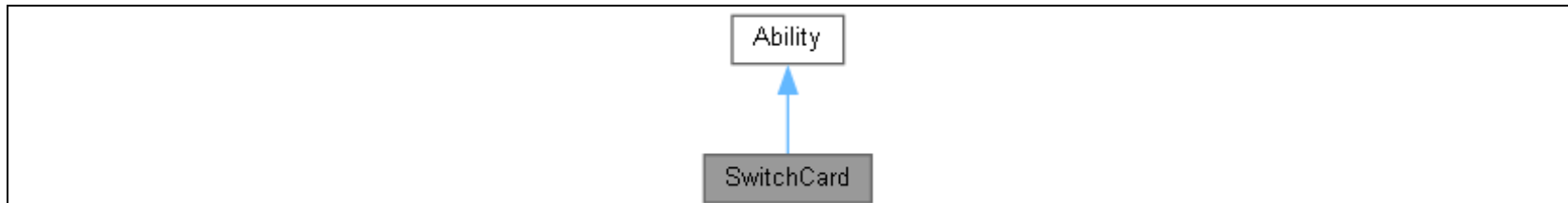


Gambar 1.2.1.13 *Inheritance Diagram*

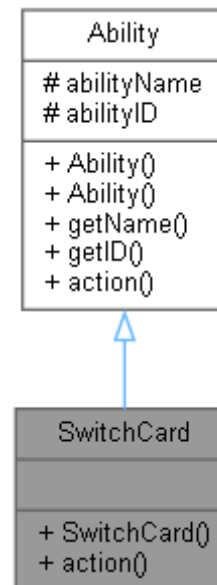


Gambar 1.2.1.14 *Inheritance Diagram (UML)*

Inheritance Diagram for Switch



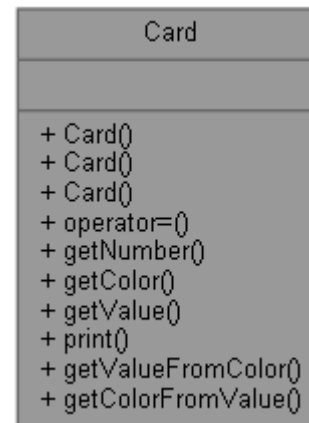
Gambar 1.2.1.15 *Inheritance Diagram*



Gambar 1.2.1.16 *Inheritance Diagram (UML)*

ii. Card

Collaboration Diagram for Card



Gambar 1.2.2.1 Collaboration Diagram (UML)

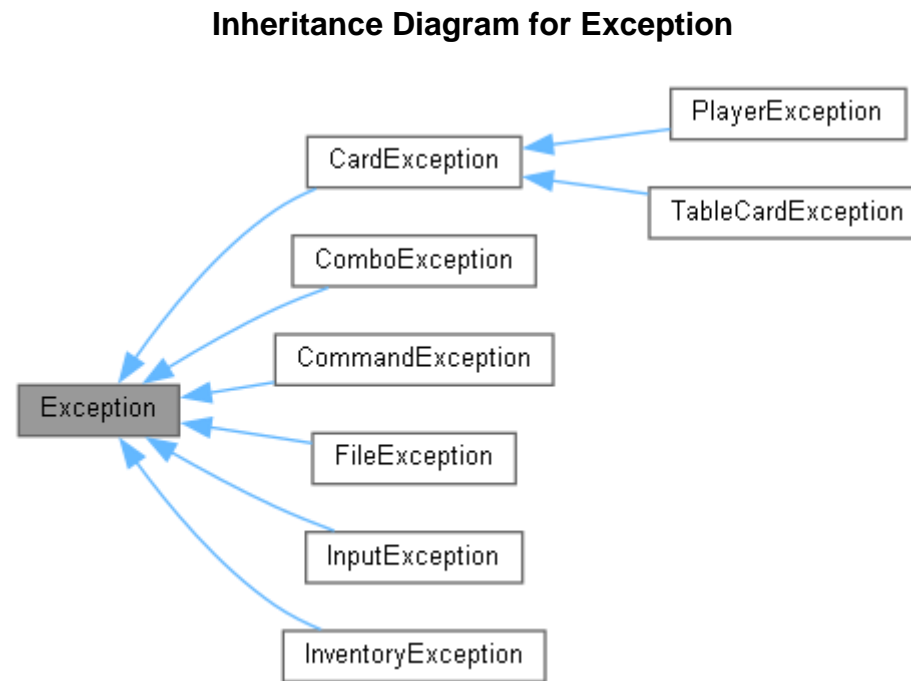
iii. Combination

Inheritance Diagram for Combination

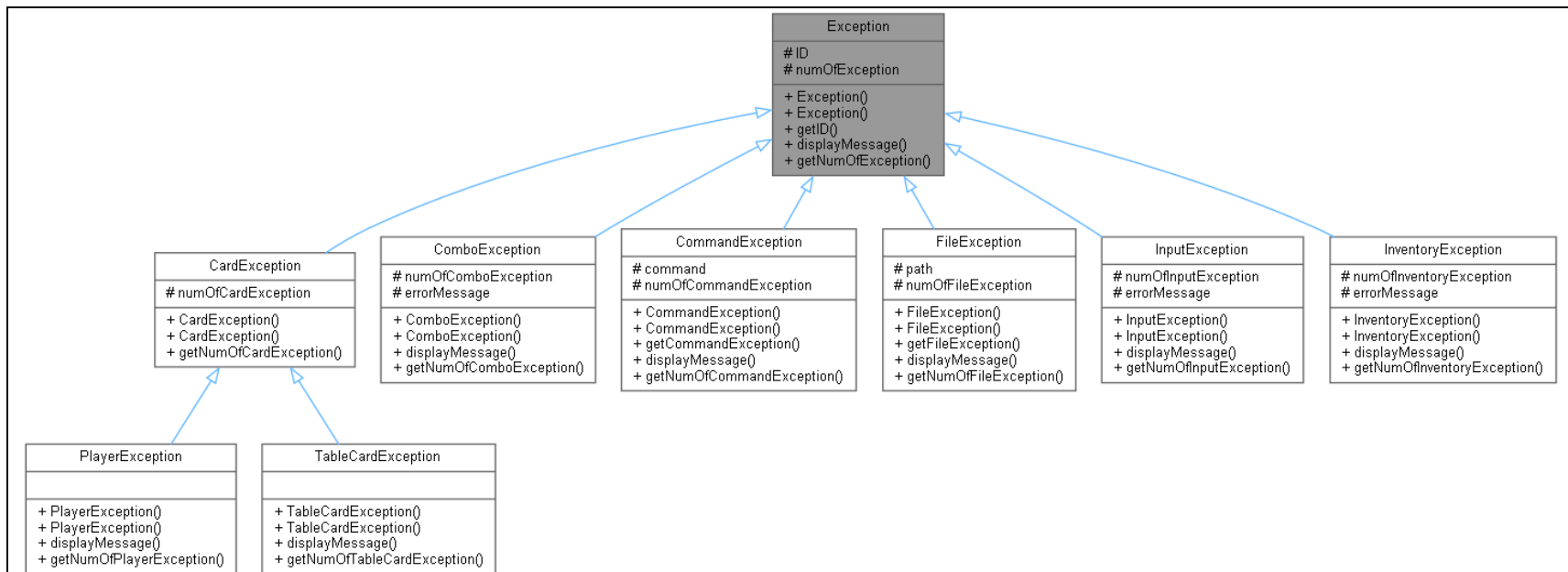


Gambar 1.2.3.1 *Inheritance Diagram*

iv. Exception

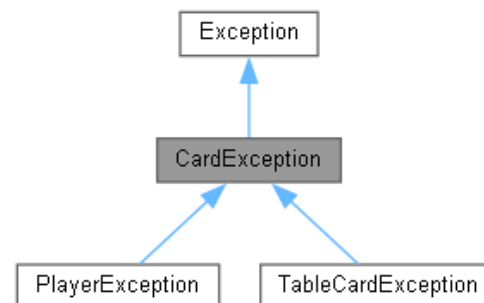


Gambar 1.2.4.1 *Inheritance Diagram*

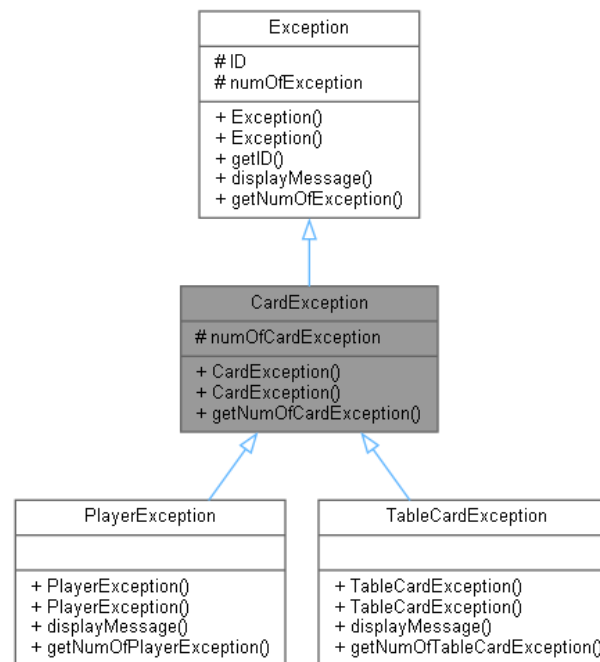


Gambar 1.2.4.2 Inheritance Diagram (UML)

Inheritance Diagram for CardException

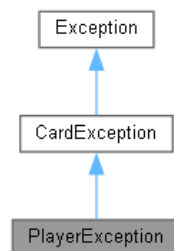


Gambar 1.2.4.3 Inheritance Diagram



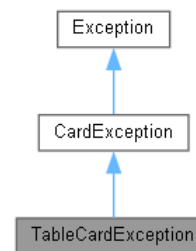
Gambar 1.2.4.4 *Inheritance Diagram (UML)*

Inheritance Diagram for PlayerException

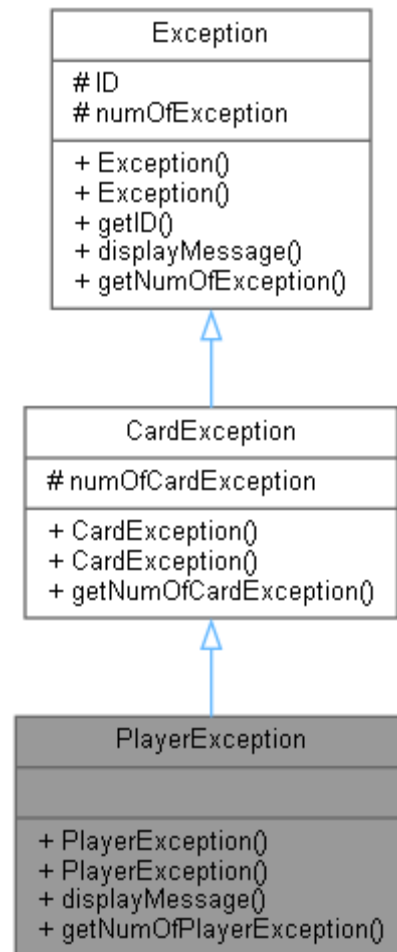


Gambar 1.2.4.5 *Inheritance Diagram*

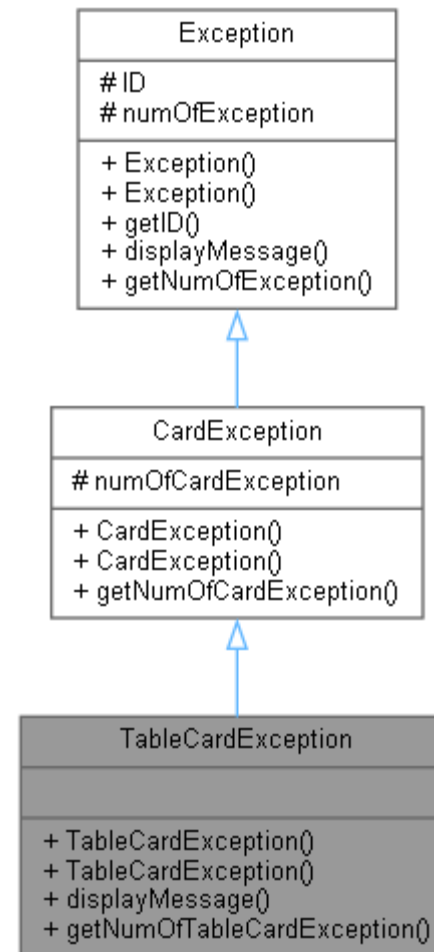
Inheritance Diagram for TableCardException



Gambar 1.2.4.7 *Inheritance Diagram*



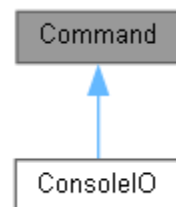
Gambar 1.2.4.6 Inheritance Diagram (UML)



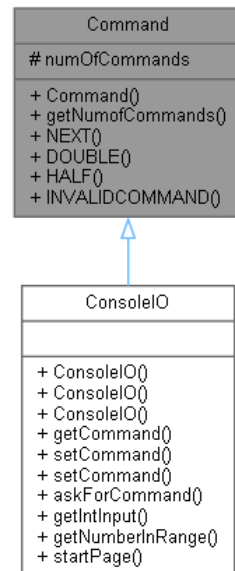
Gambar 1.2.4.8 Inheritance Diagram (UML)

v. IO

Inheritance Diagram for Command

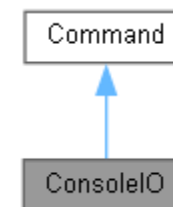


Gambar 1.2.5.1 *Inheritance Diagram*

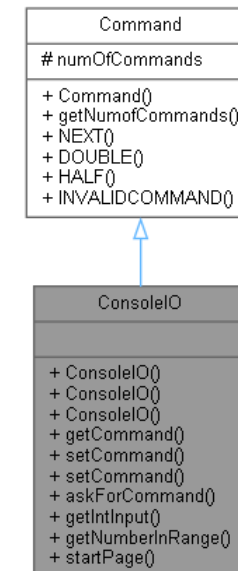


Gambar 1.2.5.2 *Inheritance Diagram (UML)*

Inheritance Diagram for ConsoleIO



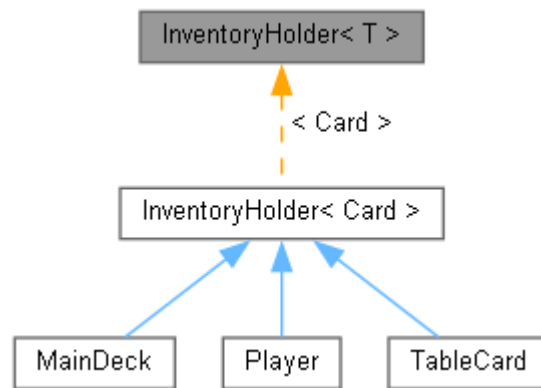
Gambar 1.2.5.3 *Inheritance Diagram*



Gambar 1.2.5.4 *Inheritance Diagram (UML)*

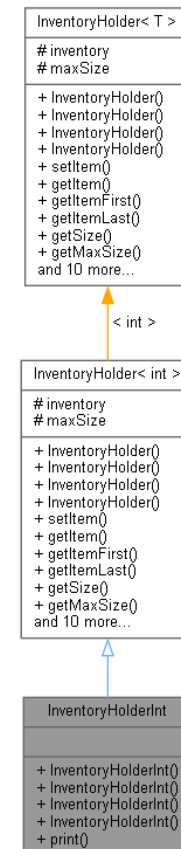
vi. InventoryHolder

Inheritance Diagram for InventoryHolder< T >

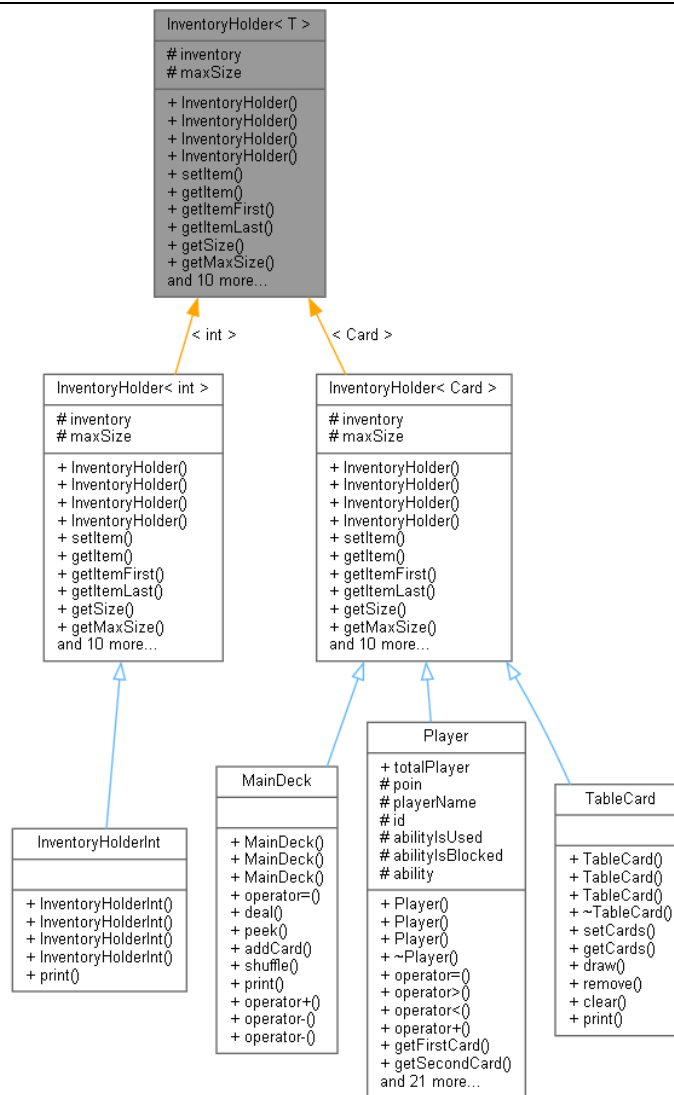


Gambar 1.2.6.1 Inheritance Diagram

Inheritance Diagram for InventoryHolder< int >



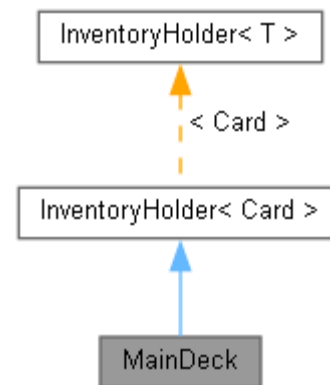
Gambar 1.2.6.3 Inheritance Diagram (UML)



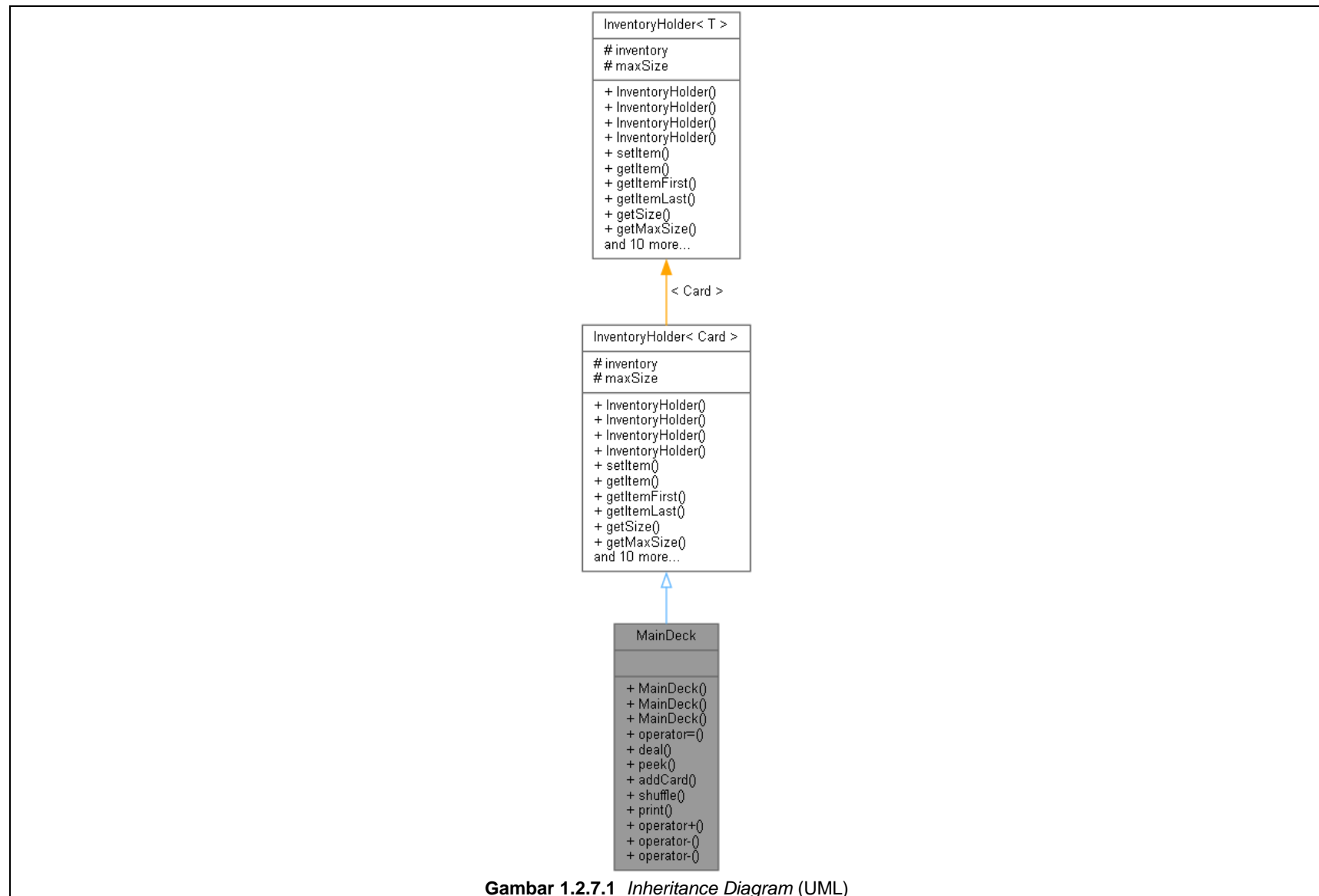
Gambar 1.2.6.2 *Inheritance Diagram (UML)*

vii. MainDeck

Inheritance Diagram for MainDeck



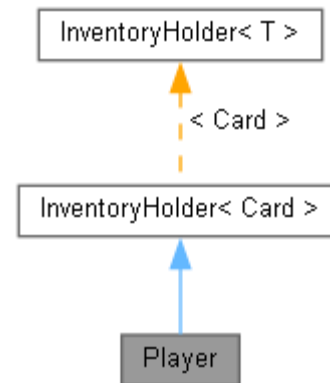
Gambar 1.2.7.1 *Inheritance Diagram*



Gambar 1.2.7.1 Inheritance Diagram (UML)

viii. Player

Inheritance Diagram for Player

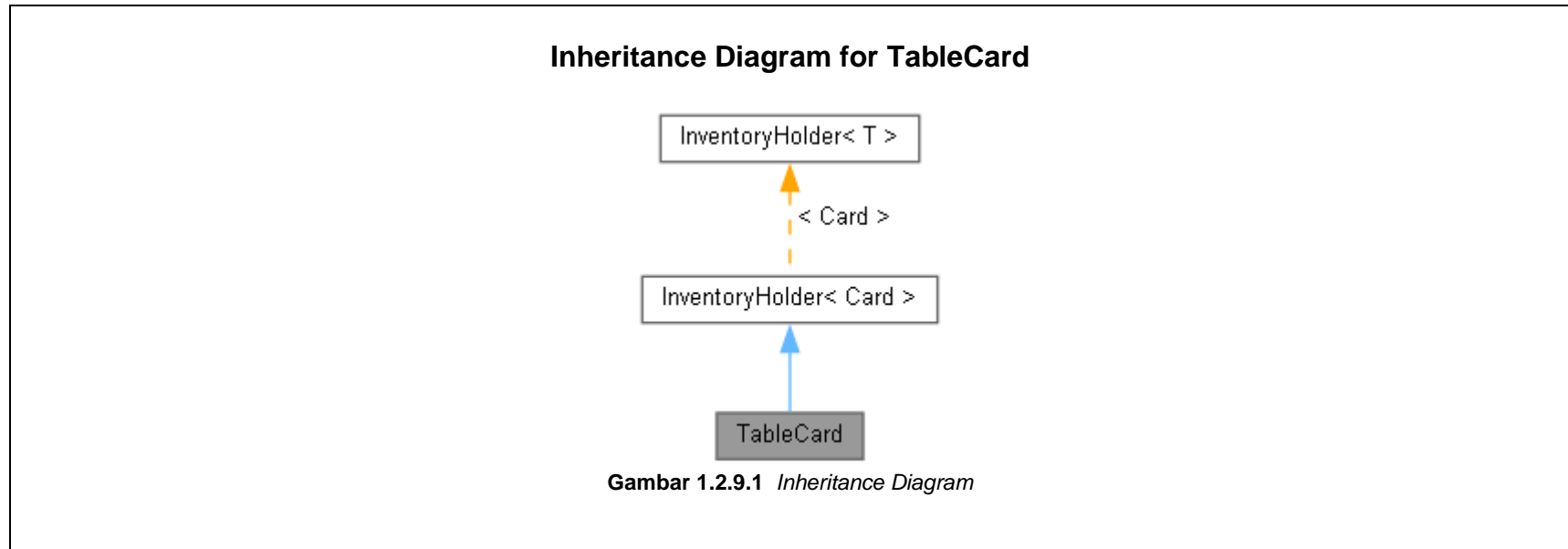


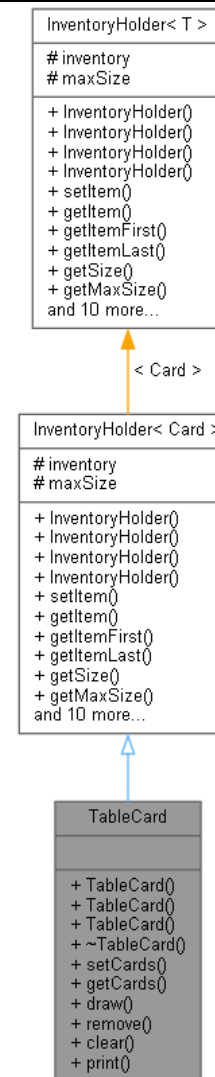
Gambar 1.2.8.1 *Inheritance Diagram*



Gambar 1.2.8.2 Inheritance Diagram (UML)

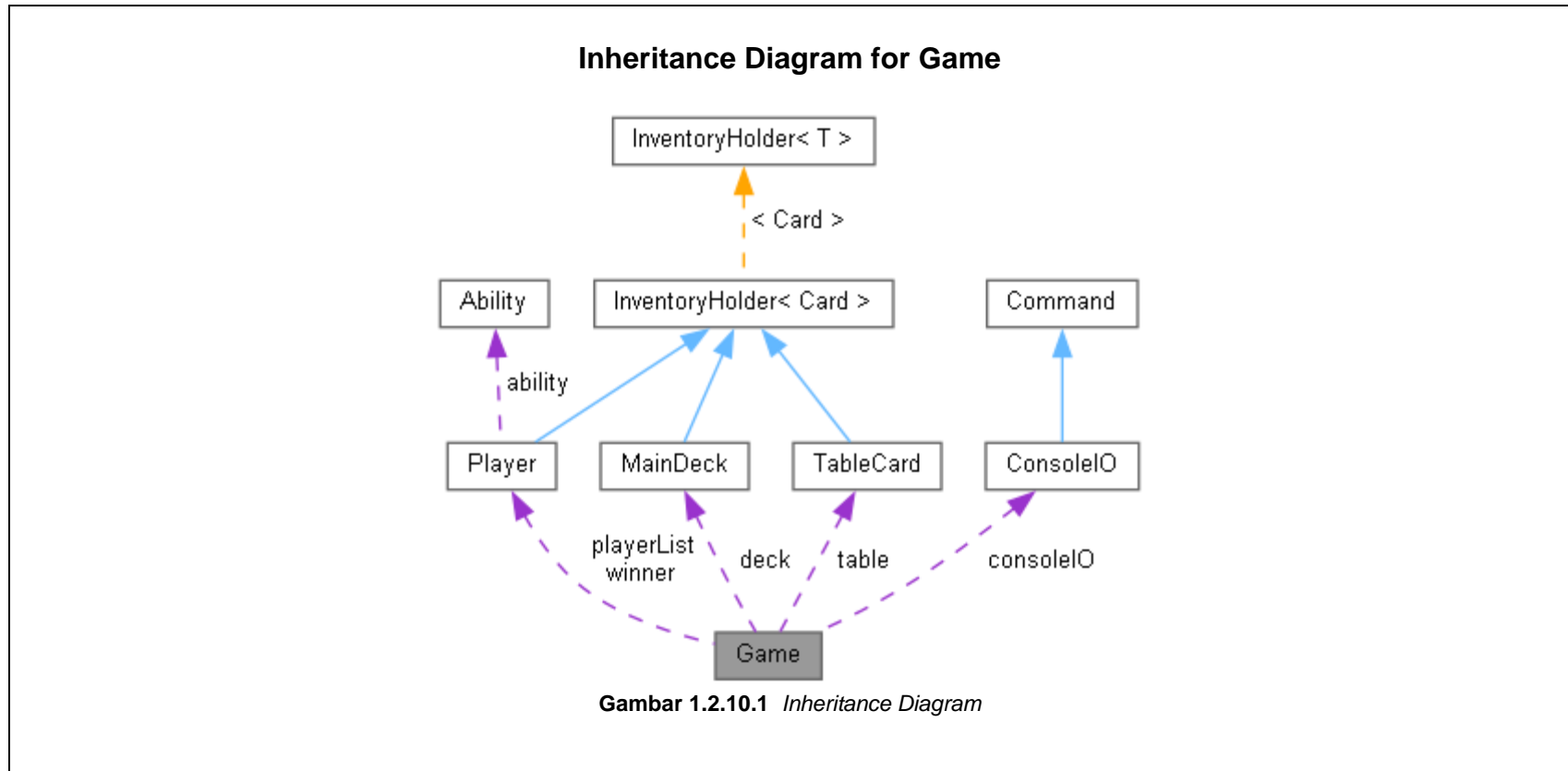
ix. TableCard

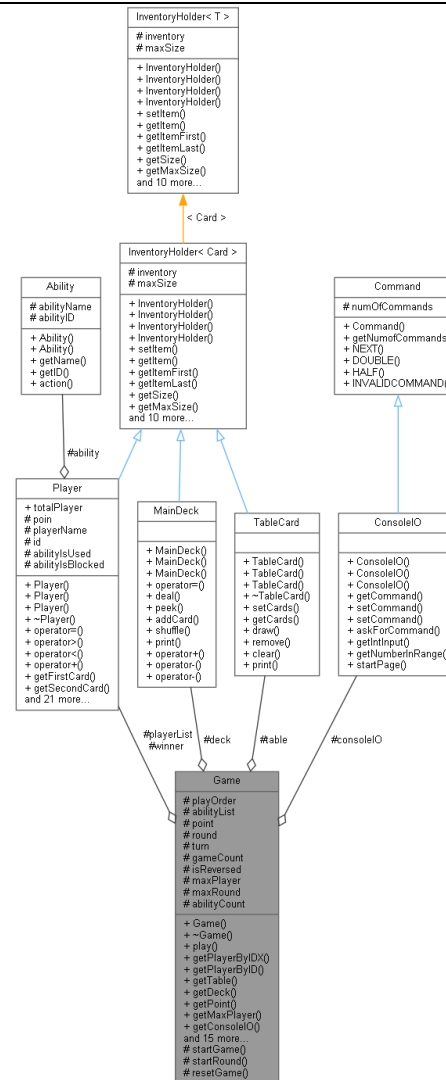




Gambar 1.2.9.2 Inheritance Diagram (UML)

x. Game





Gambar 1.2.10.2 Inheritance Diagram (UML)

c. Class Design

i. Ability

Kelas Ability berisi kemampuan-kemampuan yang ada dalam game. Kemampuan kemampuan yang ada dalam game ini cukup banyak, terdiri dari:

- SWAP, kemampuan untuk menukar 1 kartu dalam main deck pemain lain dengan 1 kartu main deck milik pemain lain.
- SWITCH, kemampuan untuk menukar 2 kartu milik sendiri dengan 2 kartu milik player lain.
- ABILITYLESS, kemampuan untuk mematikan ability lawan.
- REVERSE DIRECTION, kemampuan untuk mengubah urutan bermain menjadi kebalikannya.
- QUADRUPLE, kemampuan untuk meningkatkan poin game sebanyak 4 kali
- QUARTER, kemampuan untuk menurunkan poin game sebanyak $\frac{1}{4}$ atau 0,25 kali
- REROLL, kemampuan untuk mengembalikan kartu sendiri dan mengambil ulang 2 kartu

Ability ini juga terdapat variable id yang merepresentasikan sebuah ability. Ability ini hanya dapat digunakan sekali selama permainan. Untuk mendapatkan ability ini harus menjalankan game selama 2 ronde. Tiap Player akan mendapatkan ability hanya 1 secara acak yang dibagikan saat ronde 2.

ii. AbstractClasses

Kelas AbstractClasses menggunakan konsep template, kelas generik, serta operator overloading dalam implementasinya. Kami menggunakan konsep kelas generik untuk mengabstraksi tipe data yang diintegrasikan pada kelas InventoryHolder dalam AbstractClasses, sebagai contoh, kelas InventoryHolder diinstansiasi menjadi beberapa objek dengan tipe data template digantikan dengan Card. Selain itu, kelas InventoryHolder juga menggunakan konsep operator overloading yang digunakan sebagai metode getter dan setter elemen. Kelebihan dari implementasi dengan cara tersebut adalah objek yang terinstansiasi pada kelas lain dapat dengan mudah menggunakan kelas InventoryHolder tanpa perlu membaca bagian body kelas InventoryHolder terlebih dahulu. Namun, kekurangan dari implementasi dengan cara tersebut adalah rawan terjadinya error saat compile akibat adanya kelas template tersebut.

iii. Card

Kelas Card merupakan class yang digunakan untuk membuat objek kartu yang nanti digunakan dalam game. Kelas Card ini sangat berguna karena hampir semua kelas menggunakan class Card ini. Pada class ini terdapat constructor yang tiap objek terdiri dari variable warna dan nomor yang identik. Class Card ini nantinya dibuat dan disimpan pada player, table card, dan

maindeck. Pada game, card juga memiliki kombinasi-kombinasi berdasarkan warna dan nomor dalam game yang nantinya akan berpengaruh pada poin dalam game.

iv. **Combination**

Kelas Combination adalah class yang digunakan untuk mencari kombinasi antara kartu milik pemain dan kartu yang ada pada meja. Selain itu, kelas Combination juga menyimpan nilai absolut dari sebuah kombinasi kartu serta angka tertinggi dan warna tertinggi dari sebuah kombinasi. Kelas ini memanfaatkan konsep inheritance dengan setiap kombinasi merupakan child dari kombinasi dengan peringkat lebih rendah. Pertama kombinasi kartu akan diperiksa mulai dari kombinasi paling tinggi dan jika tidak ditemukan di suatu kelas maka akan dicari di kombinasi berikutnya yang lebih rendah.

v. **Exception**

Kelas Exception banyak menerapkan konsep inheritance dan polymorphism dalam implementasinya. Kelas Exception menggunakan konsep inheritance dan polymorphism untuk menggeneralisir tipe objek pada kelas anak yang diinstansiasi dengan cukup mendeklarasikan tipe kelas dasarnya. Polymorphism juga berfungsi pada Exception handling, dimana kita harus melakukan catch Exception berdasarkan tipenya. Akan lebih mudah dan sederhana apabila kita cukup menangkap Exception pada kelas dasarnya, karena dengan cara seperti itu, kita tidak perlu mencantumkan seluruh tipe kemungkinan tipe Exception yang dilemparkan oleh modul. Kelebihan dari implementasi dengan cara tersebut adalah mudahnya kelas Exception ini untuk di-inherit oleh kelas Exception yang lebih spesifik penggunaannya (seperti inventoryException). Selain itu, objek dengan tipe Exception tersebut mudah digunakan pada try catch block dikarenakan sifat polymorphismnya tersebut (hanya perlu menangkap tipe dasarnya saja jika terjadi error). Kekurangan dari implementasi dengan cara tersebut adalah message yang dihasilkan ketika terlemparnya Exception akan susah terlacak (susah menjadi error apa yang menghasilkan Exception tersebut)

vi. **Game**

Kelas Game merupakan kelas yang akan melakukan association terhadap kelas-kelas lain, seperti: Player, TableCard, MainDeck, serta Ability. Kelas Game juga memiliki method-method yang berfungsi untuk menjalankan permainan pada program Kami. Konsep OOP yang diterapkan dalam kelas Game antara lain adalah Inheritance dan Polymorphism. Konsep Inheritance dapat dibuktikan dengan penurunan kelas Game dari kelas ValuedObject, dimana kelas ValuedObject merupakan abstract class dan berisi method getPoint(). Sementara itu, Polymorphism diterapkan dalam beberapa method seperti getTable(), getPlayerByIdx, setTable, setDeck, dan juga setPoint()

vii. **IO**

Kelas ConsoleIO merupakan child dari kelas Command. Kelas Command itu sendiri dibuat untuk memanggil method yang telah diterapkan pada kelas lainnya untuk dapat digunakan di program utama. ConsoleIO sebagai child dari Command akan mampu mengakses method pada Command sehingga struktur dari program input command user dibuat di kelas ini. Tujuan

dibuatnya kelas ini agar pada file main.cpp tidak redundan dalam algoritmanya, dengan cukup memanggil method dari Game dan ConsoleIO, kita akan dapat menjalankan program utamanya. Kelas Command juga dibuat agar penambahan command yang dapat dipakai menjadi lebih mudah karena cukup ditambahkan di kelas tersebut saja.

viii. **MainDeck**

Kelas mainDeck merupakan bentuk penerapan konsep Inheritance, dimana kelas ini merupakan turunan dari kelas InventoryHolder < Card >. Kelas ini merupakan implementasi dari kumpulan method yang digunakan untuk melakukan operasi pada objek Card dalam deck utama permainan. Pada kelas ini, konsep OOP yang diterapkan, antara lain: konsep Overloading. Konsep Overloading terlihat dengan adanya penerapan method operator+, operator-, serta operator=.

ix. **Player**

Kelas Player merupakan child dari kelas Inventory Holder. Penggunaan Inventory Holder digunakan untuk menyimpan objek kartu pada suatu player. Kelas Player menggunakan konsep Overloading Operator untuk berbagai operasi. Operasi yang digunakan adalah +, =, <, >. Untuk + digunakan untuk menambahkan kartu ke dalam inventory player. Untuk = digunakan untuk copy constructor yang digunakan untuk menyalin suatu player dengan player lain. Dan untuk operator < dan > digunakan untuk membandingkan poin antara suatu player dengan player lain. Terdapat juga constructor dan copy constructor serta getter untuk memudahkan penggunaan class Player dalam mengambil data sekaligus membuat class Player.

x. **TableCard**

Kelas TableCard merupakan child dari kelas Inventory Holder. Penggunaan Inventory Holder digunakan untuk menyimpan objek kartu yang diletakkan di meja (kartu yang digunakan bersama oleh semua Player). Kelas TableCard menyediakan method getter getCards() yang mengembalikan kartu TableCard sehingga dapat digunakan dalam perhitungan kombo, method draw() untuk menambahkan satu kartu dari MainDeck ke TableCard, method clear() untuk membersihkan kartu TableCard pada setiap akhir permainan, dan method print() yang meng-override method pure virtual print() dari kelas Inventory Holder untuk menampilkan kartu TableCard.

xi. **TemplateFunction**

Kelas TemplateFunction merupakan kelas yang menerapkan beberapa konsep OOP, antara lain: generic function, STL(Standard Template Library), dan Overloading. Konsep generic function diterapkan pada fungsi getMaxArr(), getMaxMapKey(), ShuffleArr(), serta ShuffleVec(). Dengan demikian fungsi-fungsi ini dapat bekerja dengan berbagai jenis tipe data yang berbeda tanpa harus mengimplementasikan fungsinya secara berbeda. Kode ini menggunakan STL pada pemanfaatan kelas-kelas seperti vector dan map untuk menyimpan dan memanipulasi data. Sementara konsep overloading diterapkan pada fungsi getMaxArr() dan getMaxMapKey(), dimana digunakan untuk mendefinisikan fungsi dengan tipe argumen yang berbeda.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep Inheritance dan Polymorphism digunakan di beberapa kelas pada program. Kelas yang menggunakan Inheritance dan Polymorphism adalah Command (Parent) dan ConsoleIO(Child), Exception (Parent) dan banyak Child-nya, CardException (Parent) dengan PlayerCardException(Child) dan TableCardException(Child). Inheritance digunakan untuk mengabstraksikan kelas yang terbentuk serta pula memungkinkan kita untuk menggunakan polymorphism dalam pemrograman berorientasi objek ini. Polymorphism itu sendiri digunakan untuk mempermudah pengolahan suatu objek dengan hanya cukup menggunakan tipe kelas dasarnya saja. Kedua konsep tersebut memiliki peran penting dalam pengoperasian pada objek dari kelas Exception dan Item. Kelas Exception dan IO sama-sama memiliki tipe kelas anak yang beragam, dimana polymorphism dibutuhkan untuk mengabstraksi tipe deklarasi objek-objek menjadi tipe kelas dasarnya tersebut ketika digunakan pada suatu tipe data collection (seperti array, list, dan vector).

```
#ifndef CONSOLE_IO_INTERFACE_HPP
#define CONSOLE_IO_INTERFACE_HPP

#include "commandInterface.hpp"
#include <iostream>
#include <string>
class Player;
class Game;
using namespace std;

class ConsoleIO : public Command {
private:
    string command; // command yang akan dijalankan
public:
    // tidak memerlukan default ctor (tidak ada list of exception), assignment operator (tidak ada assignment), dan
    // dtor (perluanya perhitungan jumlah object ConsoleIO setelah dihapus)
    ConsoleIO(); // user-defined ctor
    ConsoleIO(string); // ctor
    ConsoleIO(const ConsoleIO& CI); // cctor
    string getCommand(); // command getter
    void setCommand(string); // command setter
    void setCommand(); // command setter (override function for CLI-based command)
    void askForCommand(Player& _player, Game& _game); // main logic
    int getIntInput(); // input number
    int getNumberInRange(int _lower, int _upper); // input range number
    void startPage();
};

#endif
```

Gambar 2.1.1 Interface ConsoleIO

```
#ifndef COMMAND_INTERFACE_HPP
#define COMMAND_INTERFACE_HPP

#include <iostream>
#include <string>
#include "../Exception/commandExceptionInterface.hpp"
class Game;
class Player;
// #include "../Exception/commandException.cpp"

using namespace std;

class Command {
protected:
    static int numOfCommands;

public:
    Command(); // Default Constructor
    int getNumofCommands(); // Returns the number of commands
    void NEXT(); // Next Command
    void DOUBLE(Player& _game); // Double Command
    void HALF(Player& _game); // Half Command
    void INVALIDCOMMAND(string command); // Invalid Command
};

#endif
```

Gambar 2.1.2 Interface Command

2.2. Method/Operator Overloading

Operator Overloading merupakan kasus khusus polymorphism, dimana operator yang berbeda memiliki implementasi yang berbeda tergantung classnya. Operator Overloading digunakan untuk memudahkan operasi dari berbagai type data sesuai dengan kebutuhan. Untuk

method atau operator overloading, kami memakai beberapa operator overloading, diantaranya =,>,<,+,- . Untuk operator operator tersebut digunakan untuk copy constructor(untuk =) dan mengecek kebenaran apakah lebih besar atau lebih kecil untuk sebuah nilai(untuk < dan >). Selain itu juga ada operasi matematika, yaitu + dan - digunakan untuk menambah atau mengurangi objek pada suatu list/array/vector. Berikut contoh penggunaan operator overloading pada program kami.

```

MainDeck& MainDeck::operator+(const Card& card){
    addCard(card);
    return *this;
}
MainDeck& MainDeck::operator-(Player& player){
    player.addCard(getItemLast());
    deleteLast();
    return *this;
}

MainDeck& MainDeck::operator-(TableCard& table){
    table.draw(getItemLast());
    deleteLast();
    return *this;
}

```

Gambar 2.2.1 Fungsi Overloading (MainDeck)

```

Player& Player::operator=(const Player& p)
{
    int banyak = p.getSize();
    for(int i=0;i<banyak;i++)
    {
        addCard(p.inventory[i]);
    }
    this->poin = p.poin;
    this->playerName = p.playerName;
    this->ability = p.ability;
    this->abilityIsUsed = p.abilityIsUsed;
    this->abilityIsBlocked = false;
    return *this;
}

```

Gambar 2.2.2 Fungsi Overloading (Player)

```

Player& Player::operator+(const Card& x)
{
    addCard(x);
    return *this;
}

bool Player::operator>(const Player& p)
{
    return(this->poin > p.poin);
}

bool Player::operator<(const Player& p)
{
    return(this->poin < p.poin);
}

```

Gambar 2.2.3 Fungsi Overloading (Player)

2.3. Template & Generic Classes

Untuk template dan generic class, kelompok kami menggunakannya pada Inventory Holder dan Template Function. Template dan Generic class ini digunakan untuk memakai fungsi fungsi yang dapat dipakai oleh sebuah class.



```

- □ ×

#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP
#include <iostream>
#include <vector>
#include "../Exception/inventoryExceptionInterface.hpp"

template <class T>
class InventoryHolder{
protected:
    std::vector<T> inventory;
    int maxSize;
public:
    // Default ctor
    InventoryHolder() : inventory(0), maxSize(0){}

```

Gambar 2.3.1 Generic Class (InventoryHolder< T >)

Untuk Inventory Holder, kelompok kami memakai template dan generic class karena Inventory Holder digunakan untuk menyimpan sebuah objek ke dalam vector bertipe class yang diinginkan. Inventory Holder dapat dipakai oleh berbagai class. Oleh karena itu, Inventory Holder sangat fleksibel dan dapat digunakan untuk berbagai class.

Lalu untuk Template Function, kelompok kami menerapkan template dan generic class pada Template Function karena pada Template Function berisi fungsi fungsi pengolahan array dan vector. Karena array dan vector dapat berisi objek dengan berbagai class/type yang sejenis, maka dibuat Template Function sebagai template dan generic class.

```

template <class T>
T& getMaxArr(T arr[], int n){
    T& min = arr[0];
    for (int i = 1; i < n; i++){
        if (arr[i] > min){
            min = arr[i];
        }
    }
    return min;
}

template <class T>
void ShuffleArr(T arr[], int n){
    for (int i = 0; i < n-1; i++){
        int randIdx = rand()%(n-i)+i;
        swap(arr[i], arr[randIdx]);
    }
}

template <class T>
void ShuffleVec(vector<T>& vec){
    for (int i = 0; i < vec.size()-1; i++){
        int randIdx = rand()%(vec.size()-i)+i;
        swap(vec[i], vec[randIdx]);
    }
}


```

Gambar 2.3.2 Template Function

2.4. Exception

Konsep Exception hampir digunakan di setiap kelas karena banyak kasus yang perlu ditangani sehingga perlu dilakukan Exception. Seringkali, kita membutuhkan penanganan kesalahan yang muncul saat eksekusi program. Penanganan kesalahan tersebut membutuhkan kita untuk menambahkan instruksi-instruksi tertentu yang membuat program menjadi rumit dan rawan terjadinya ketidaktelitian pada implementasi. Oleh karena itu, C++ menyediakan fitur exception untuk menangani kesalahan yang terjadi saat runtime menggunakan sintaks throw, try, dan catch. Sintaks throw memiliki kegunaan yang mirip dengan sintaks return, dimana throw sendiri dapat dikatakan sebagai return dengan exception. Tetapi jika suatu method selesai dengan throw, maka method tersebut dikatakan selesai secara abnormal. Ketika kita ingin

mengakses exception yang di-throw oleh suatu method, maka block method tersebut harus terdapat di dalam sintaks try, serta memiliki sintaks catch untuk menangkap objek exception yang di-throw.



```
#ifndef EXCEPTION_INTERFACE_HPP
#define EXCEPTION_INTERFACE_HPP

#include <string>
#include <iostream>

using namespace std;

class Exception {
protected:
    static int numOfException; // jumlah object Exception yang ada
    const int ID; // ID exception

public:
    // tidak memerlukan default ctor (tidak ada list of exception), assignment operator
    // (tidak ada assignment), dan dtor (perlunya perhitungan jumlah object Exception setelah
    // dihapus)
    Exception(int); // user-defined ctor
    Exception(const Exception&); // cctor

    static int getNumOfException(); // numOfException getter
    int getID() const; // ID getter

    virtual void displayMessage() const = 0; // message display
};
#endif
```

Gambar 2.4.1 Exception Interface


```
void setItem(int i, T a){
    if (i < 0 || i >= inventory.size())
        // exception out of bounds
    {
        throw InventoryException(1);
    }
    inventory[i] = a;
}
```

Gambar 2.4.2 Exception Out of Bounds

```
InventoryHolder(int n, int maxSize) :
    inventory(n, maxSize(maxSize)) {
    if (maxSize < n)
        // exception maxSize
    {
        throw InventoryException(0);
    }
}
```

Gambar 2.4.3 Exception MaxSize

```
T getItem(int i) const {
    if (i < 0 || i >= inventory.size())
        // exception out of bounds
    {
        throw InventoryException(1);
    }
    return inventory[i];
}
```

Gambar 2.4.4 Exception Out of Bounds

2.5. C++ Standard Template Library

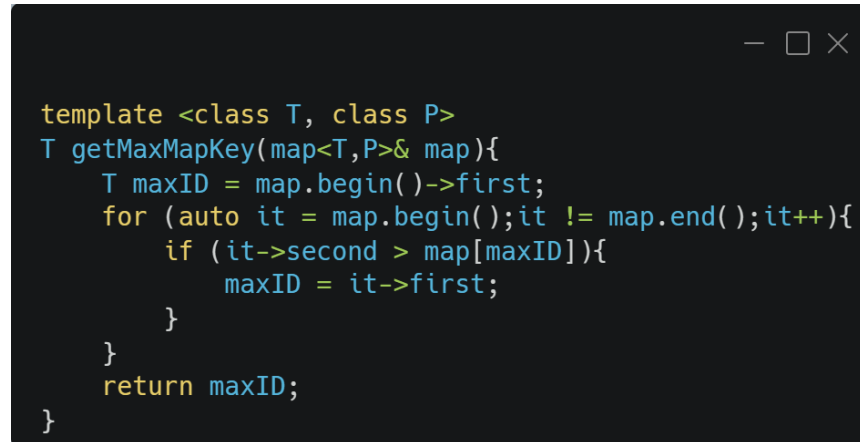
Standar Template Library merupakan class-class yang sudah disediakan oleh C++ sekaligus methodnya. Kita dapat menggunakan library ini tanpa mengimplementasikan class tersebut. Untuk penggunaan STL ini, kelompok kami menggunakan STL vector, map, pair. Vector digunakan dalam Inventory Holder untuk menyimpan sebuah objek. Vector ini bersifat dinamik sehingga lebih mudah dalam menginputkan sebuah objek daripada list yang bersifat statik. Vector ini juga hampir sama dengan list yang mana dapat diakses tiap element seperti list. Penggunaan vector ini dapat dilihat pada gambar berikut.

```
#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP
#include <iostream>
#include <vector>
#include "../Exception/inventoryExceptionInterface.hpp"

template <class T>
class InventoryHolder{
protected:
    std::vector<T> inventory;
    int maxSize;
public:
    // Default ctor
    InventoryHolder() : inventory(0), maxSize(){}
}
```

Gambar 2.5.1 Penggunaan vector pada Inventory Holder

Untuk pengaplikasian map dan pair, digunakan pada ketika mencari player dengan combo tertinggi dengan key sebagai index player dan value adalah kombinasi dari kartu player. Berikut penggunaan map pada combo pada gambar berikut.



```
template <class T, class P>
T getMaxMapKey(map<T,P>& map){
    T maxID = map.begin()->first;
    for (auto it = map.begin(); it != map.end(); it++){
        if (it->second > map[maxID]){
            maxID = it->first;
        }
    }
    return maxID;
}
```

Gambar 2.5.2 Penggunaan map pada TemplateFunction

2.6. Konsep OOP lain

Konsep OOP lain yang digunakan dalam program adalah Abstract Base Class dan Composition. Konsep Abstract Base Class diterapkan pada base class InventoryHolder dan ValuedObjects, misalnya pada InventoryHolder yang di-inherit oleh kelas MainDeck, Player, dan TableCard untuk diimplementasikan sesuai dengan kelasnya. Konsep Composition digunakan pada kelas Player yang pada atributnya menggunakan kelas Ability.

```

template <class T>
class InventoryHolder{
protected:
    std::vector<T> inventory;
    int maxSize;
public:
    // method-method lainnya
    virtual void print() = 0;
};

```

Gambar 2.6.1 Abstract Base Class pada Inventory Holder

```

class Player : public InventoryHolder<Card>{
protected:
    int poin;
    string playerName;
    const int id;
    bool abilityIsUsed; // new
    bool abilityIsBlocked;
    Ability* ability; //new
}

```

Gambar 2.6.2 Composition pada Player

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Generic Class

Untuk generic class, kami menggunakannya dalam Inventory Holder. Yang seperti sebelumnya sudah dijelaskan, digunakan untuk menyimpan sebuah objek ke dalam vector bertipe class yang diinginkan. Inventory Holder dapat dipakai oleh berbagai class. Oleh karena itu, Inventory Holder sangat fleksibel dan dapat digunakan untuk berbagai class.

A screenshot of a code editor window showing C++ code for a generic class. The code includes preprocessor directives for header files, standard library includes for iostream and vector, and an include for an exception interface. It then defines a template class InventoryHolder with a protected vector and maxSize attribute, and a public default constructor that initializes the vector and maxSize.

```
#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP
#include <iostream>
#include <vector>
#include "../Exception/inventoryExceptionInterface.hpp"

template <class T>
class InventoryHolder{
protected:
    std::vector<T> inventory;
    int maxSize;
public:
    // Default ctor
    InventoryHolder() : inventory(0), maxSize(){}
}
```

Gambar 3.1.1.1 Gambar generic class pada inventory holder

3.2. Bonus Kreasi Mandiri

3.2.1. Pewarnaan pada CLI

Kelompok kami menerapkan bonus pewarnaan pada tampilan CLI permainan untuk lebih meningkatkan daya tarik program yang kami buat. Ketika program pertama kali dijalankan maka akan muncul sebuah ASCII ART berwarna-warni (RED, GREEN, MAGENTA, CYAN, YELLOW, BLUE). Selain itu, juga masih terdapat beberapa tampilan berwarna lainnya, seperti pada Exception yang dikeluarkan, notifikasi pergantian tiap ronde dan permainan, serta beberapa tampilan lainnya. Metode pemanggilan warna yang kelompok kami lakukan adalah dengan mendeklarasikan/mendefinisikan tiap warna terlebih dahulu, agar pemanggilan selanjutnya dapat dilakukan dengan lebih mudah.

```
#define RED "\033[1m\033[31m"
#define GREEN "\033[1m\033[32m"
#define YELLOW "\033[1m\033[33m"
#define BLUE "\033[1m\033[34m"
#define MAGENTA "\033[1m\033[35m"
#define CYAN "\033[1m\033[36m"
#define RESET "\033[0m"
```

Gambar 3.2.1.1 Pendefinisian tiap warna

```

P   E   R   M   A   I   N   A   N
┌───┴───┐ ┌───┴───┐ ┌───┴───┐ ┌───┴───┐ ┌───┴───┐
│  R  │ │  A  │ │  R  │ │  I  │ │  T  │ │  U  │
│  A  │ │  L  │ │  A  │ │  K  │ │  E  │ │  R  │
│  A  │ │  A  │ │  J  │ │  A  │ │  N  │ │  P  │
│  A  │ │  A  │ │  A  │ │  A  │ │  A  │ │  E  │
└───┬───┘ └───┬───┘ └───┬───┘ └───┬───┘ └───┬───┘
    R  A  L  A  K  E  R  A  J  A  A  N  P  E  R  M  E  N

```

Gambar 3.2.1.3 Contoh output berwarna (1)

```
void ConsoleIO::startPage(){
    cout << endl;
    cout << CYAN << "===== " << RESET << endl;
    // ascii art
    cout << "ALA KERAJAAN PERMEN" << endl;
    cout << CYAN << "===== " << RESET << endl;
    cout << endl;
}
```

Gambar 3.2.1.2 Contoh pemanggilan definisi tiap warna

```

--- INPUT NAMA PEMAIN ---
Masukkan nama pemain ke-1 : 1
Masukkan nama pemain ke-2 : 2
Masukkan nama pemain ke-3 : 3
Masukkan nama pemain ke-4 : 4
Masukkan nama pemain ke-5 : 5
Masukkan nama pemain ke-6 : 6
Masukkan nama pemain ke-7 : 7

--- PERMAINAN KE 1 ---

--- RONDE KE 1 ---

```

Gambar 3.2.1.4 Contoh output berwarna (2)

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Game, InventoryHolder, MainDeck	13521169 / Muhammad Habibi Husni	Ahmad Ghulam Ilham, Sulthan Dzaky Alfaro, Muhammad Naufal Nalendra, Mohammad Rifqi Farhansyah, Muhammad Dhiwaul Akbar

Player,Ability,Combination,Laporan	13521159 / Sulthan Dzaky Alfaro	Ahmad Ghulam Ilham,Muhammad Habibi Husni
Combination, Laporan	13521152 / Muhammad Naufal Nalendra	Muhammad Habibi Husni Muhammad Naufal Nalendra
TableCard,Ability	13521118 / Ahmad Ghulam Ilham	Muhammad Naufal Nalendra, Muhammad Dhiwaul Akbar
Exception,I/O,command,Laporan	13521166 / Mohammad Rifqi Farhansyah	Muhammad Habibi Husni
Card	13521158 / Muhammad Dhiwaul Akbar	Ahmad Ghulam Ilham, Sulthan Dzaky Alfaro

LAMPIRAN FORM ASISTENSI

Kode Kelompok : HPC

Nama Kelompok : HiperC++

1. 13521118 / Ahmad Ghulam Ilham
2. 13521152 / Muhammad Naufal Nalendra
3. 13521158 / Muhammad Dhiwaul Akbar
4. 13521159 / Sulthan Dzaky Alfaro
5. 13521166 / Mohammad Rifqi Farhansyah
6. 13521169 / Muhammad Habibi Husni

Asisten Pembimbing : 13519105 / Widya Anugrah Putra

1. Konten Diskusi

- Pertanyaan Naufal: Mekanik game dasarnya dari Poker? Kalau mau tau mekaniknya bisa belajar dari Poker?
Jawaban: Gak boleh karena joody, jadi diubah pakai sistem poin. Sebenarnya ada yang diubah sedikit, jadi lebih baik ikuti spek saja.
- Pertanyaan Habibi: Ability card cuma sekali dalam satu permainan?
Jawaban: Iya, cuma dibagikan sekali di ronde 2.
- Pertanyaan Rifqi: Di awal pemain dikasih 2 kartu, kenapa tiba-tiba punya lebih dari 2 kartu?
Jawaban: Kartu yang dikombinasikan adalah kartu yang dipegang pemain dan yang berada di meja. Kartu di tangan pemain hanya 2, kartu di meja yang akan bertambah dan digunakan bersama untuk perhitungan kombo.
- Pertanyaan Habibi: Teknis minimal poin 4.a terdapat kelas abstrak dengan virtual value, kalau cuman memiliki method value sudah valid sebagai minimum atau tidak?
Jawaban: Ya, sudah valid.
- Pertanyaan Rifqi: Bagian ability ada kesalahan penulisan atau tidak?
Jawaban: Penulisan sudah benar begitu. Beda dengan Uno, pemain yang sudah melakukan aksi akan diskip.
- Pertanyaan Habibi: Terkait bonus 1, method dari kelas termasuk generik fungsi?
Jawaban: Fungsi generic pasti di luar kelas, beda dengan method.

2. Tindak Lanjut

Kelompok kami melakukan asistensi sebanyak satu kali yaitu pada tanggal 3 Maret 2023. Setelah dilaksanakannya asistensi, tahapan awal dari tindak lanjut yang kami lakukan adalah dengan menyusun kelas-kelas yang sekiranya diperlukan untuk membuat program 'Permainan Kartu ala Kerajaan Permen' dengan berbasis OOP. Kelas yang terbentuk setelah melakukan asistensi adalah

ValuedObject, Card, Input, Combo, InventoryHolder, MainDeck, TableCard, PlayerCard, serta Exception. Namun, seiring dengan perkembangan pengerjaan, beberapa kelas pun ditambahkan. Pembagian tugas untuk tiap kelas telah dicantumkan pada laporan bagian 4. Dalam rentang waktu masa pengerjaan, kami merasa bahwa komunikasi merupakan salah satu masalah yang cukup krusial dalam pengerjaan tugas ini, sehingga kelompok kami sering sekali mengerjakan tugas ini secara bersama-sama.