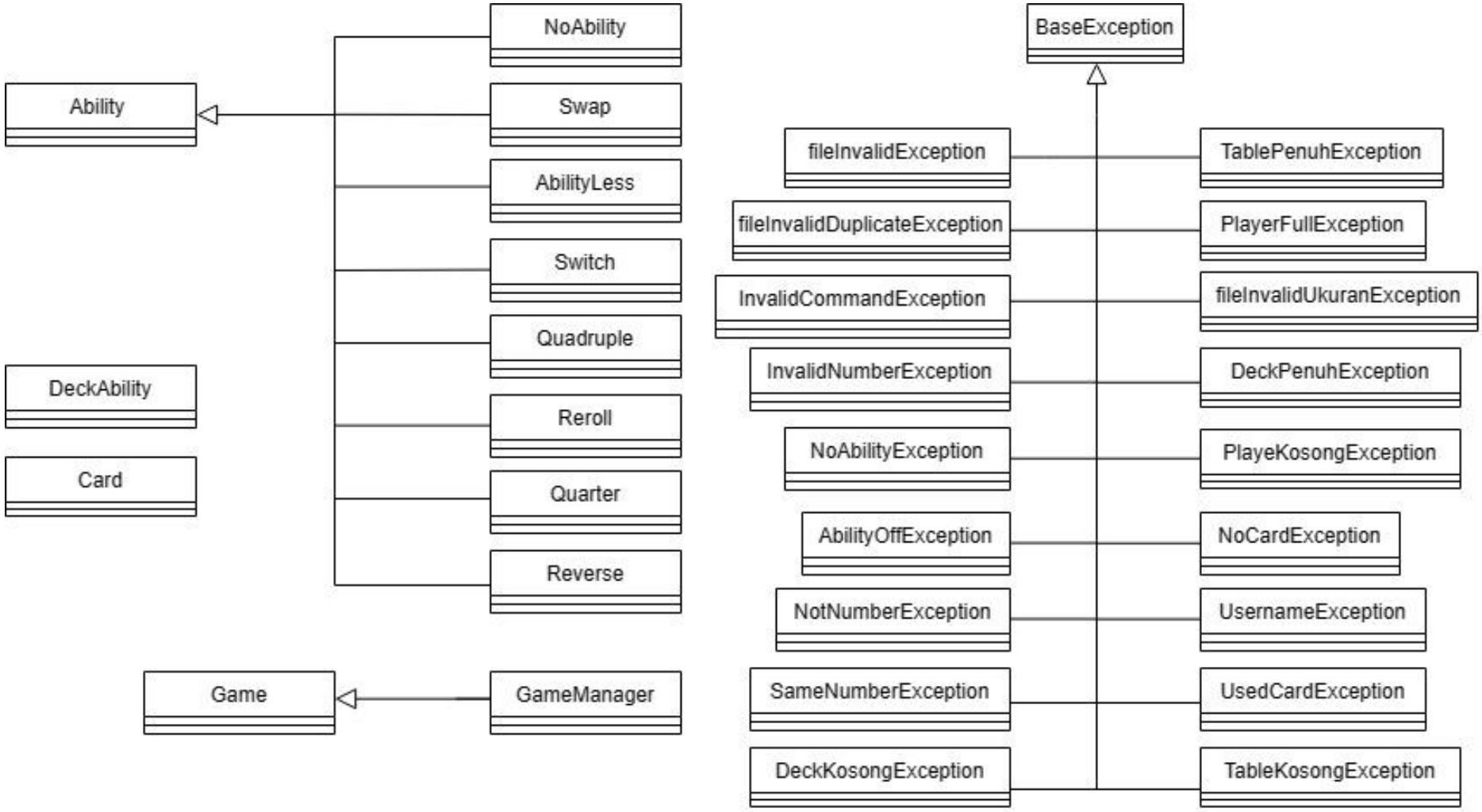
Kode Kelompok : C0K
Nama Kelompok : Anak Didikan Mama
1. 13521055 / Muhammad Bangkit Dwi Cahyono
2. 13521072 / Irsyad Nurwidianto Basuki
3. 13521081 / Bagas Aryo Seto
4. 13521103 / Aulia Mey Diva Annandya
5. 13521104 / Muhammad Zaydan Athallah
Asisten Pembimbing : Fabian Savero Diaz Pranoto
Link Repository : https://github.com/bangkitdc/IF2210_TB1_C0K

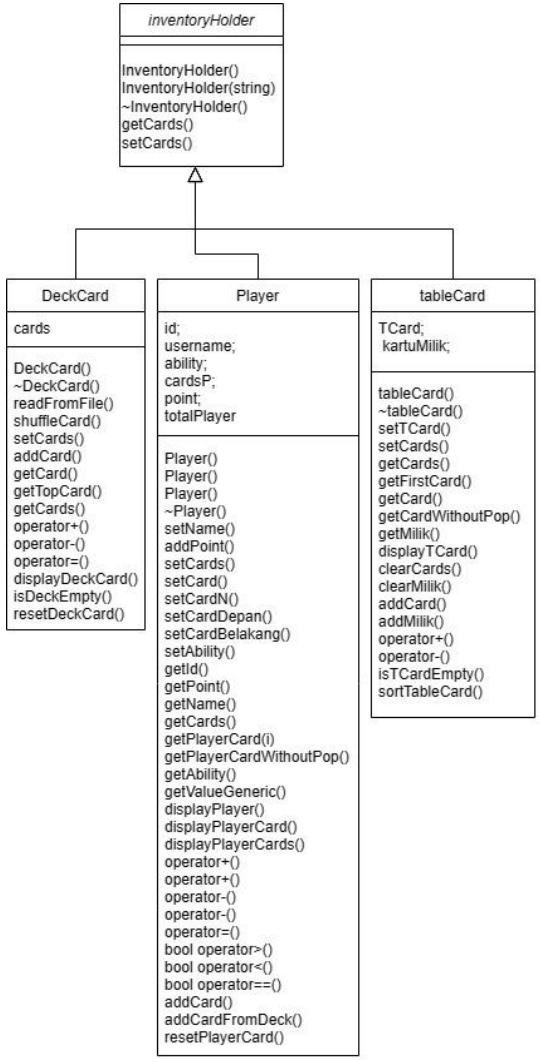
1. Diagram Kelas

a) Class Hierarchy



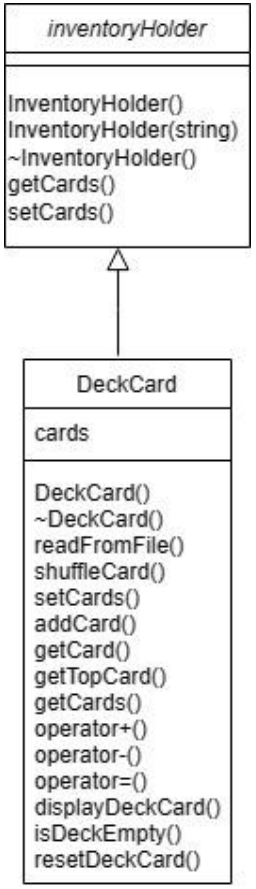
b) Class Legend

Class InventoryHolder Inheritance Diagram



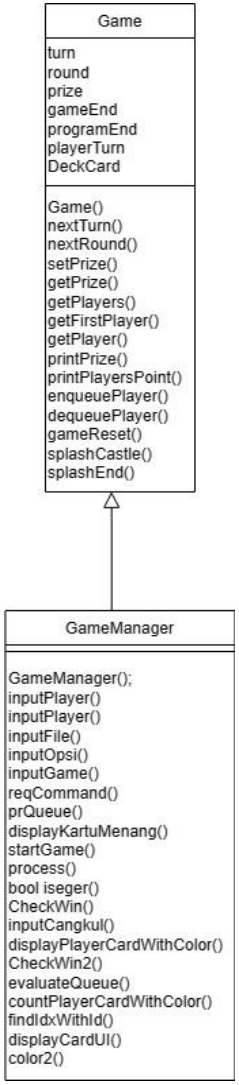
Class Player Inheritance Diagram

Class DeckCard Inheritance Diagram

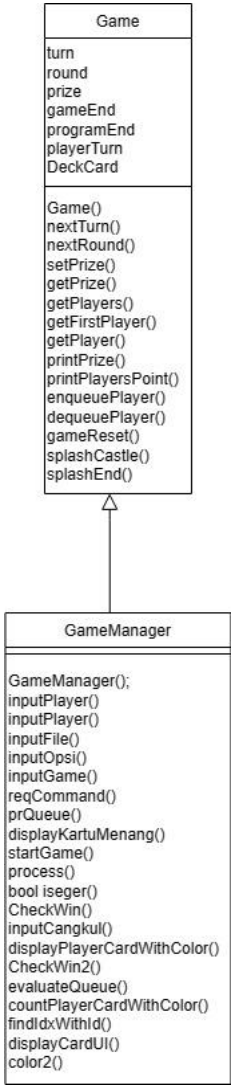


Class tableCard Inheritance Diagram

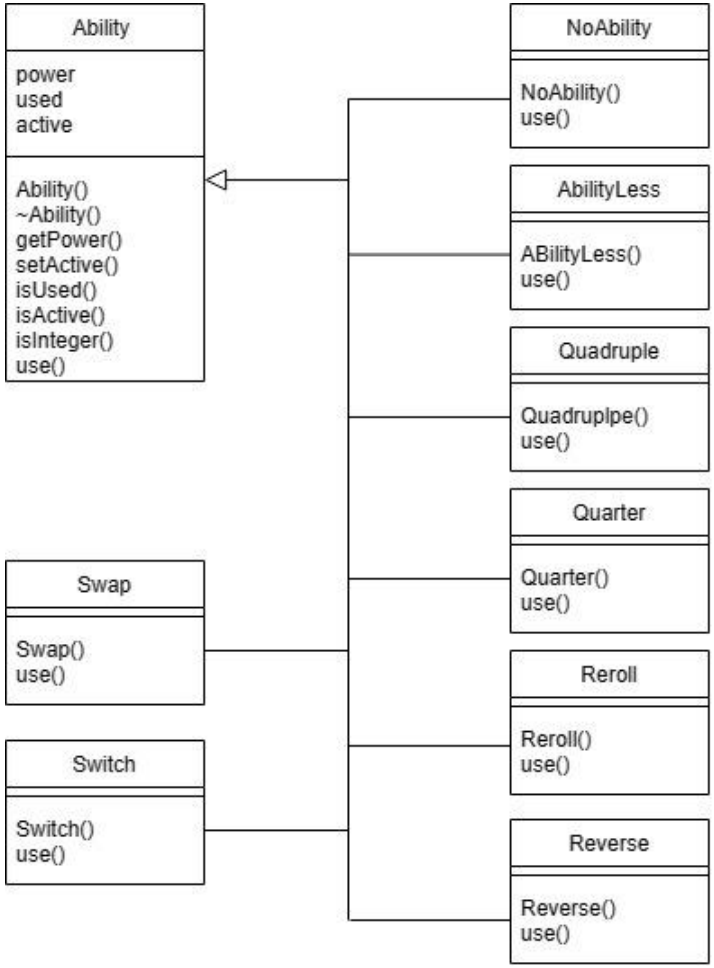
Class Game Inheritance Diagram

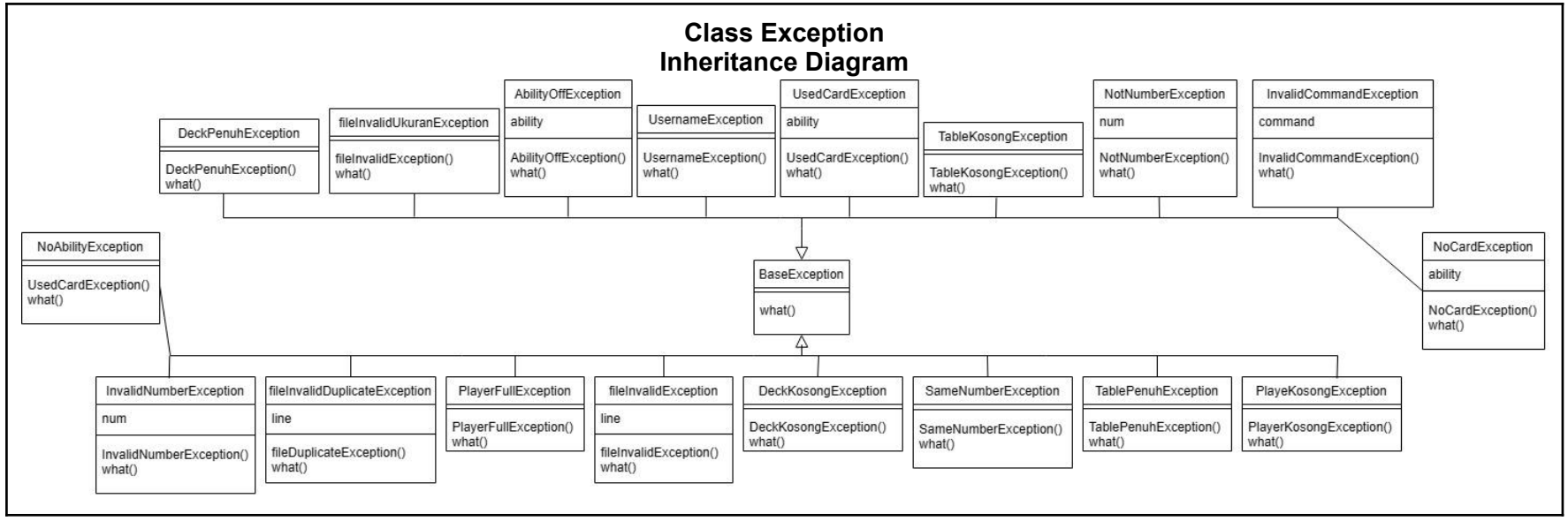


Class GameManager Inheritance Diagram



Class Abilty
Inheritance Diagram





c) Class Design

i) InventoryHolder

Kelas Item banyak menerapkan konsep inheritance, polymorphism, operator overloading, dan function overloading dalam implementasinya. Inheritance digunakan untuk menspesifikasikan suatu inventoryHolder menjadi jenis yang lebih spesifik, yaitu Player, DeckCard, dan TableCard. Kelas ini memiliki 2 pure virtual function yaitu getCards() dan setCards() untuk mengambil dan melakukan assign vector kedalam masing-masing objek. Operator overloading digunakan untuk memindahkan kartu antara objek turunan kelas inventoryHolder. Kelebihan dari kelas ini adalah kita dapat dengan mudah mengakses inventory tiap objek yang mempunyai baseclass yang sama. Kekurangan dari kelas ini adalah banyaknya getter dan setter yang diperlukan karena kelas-kelas inventoryHolder ini merupakan objek yang diakses oleh semua objek lain di dalam program ini yang memiliki kebutuhan beragam.

ii) Exception

Kelas Exception sering menggunakan konsep inheritance dan polymorphism untuk membuat tipe objek yang lebih umum di kelas anaknya. Polymorphism juga diterapkan pada Exception handling, di mana kita harus menangani Exception sesuai tipe

yang dilemparkan. Sebenarnya, lebih mudah dan sederhana jika kita menangkap Exception pada kelas dasarnya saja, sehingga kita tidak perlu menyebutkan setiap kemungkinan tipe Exception yang mungkin terjadi di modul. Cara ini memungkinkan kelas Exception diwarisi oleh kelas Exception yang lebih spesifik. Selain itu, objek dengan tipe Exception tersebut dapat digunakan dengan mudah di dalam blok try-catch karena sifat polimorfismenya (hanya perlu menangkap tipe dasar jika terjadi kesalahan). Namun, kekurangan dari cara ini adalah pesan yang dihasilkan ketika Exception terjadi sulit dilacak (sulit untuk mengetahui penyebab Exception yang dihasilkan).

iii) Ability

Kelas Ability menggunakan konsep inheritance dan polymorphism untuk membuat tipe objek umum yang dapat disimpan oleh objek Player. Ketika objek ability-ability yang lebih spesifik dibagikan kepada pemain, pemain hanya perlu menyimpan satu jenis objek yaitu Ability sehingga tidak harus menginisiasi jenis objek Ability yang spesifik. Kelebihan dari cara ini adalah generalisasi yang memudahkan dalam menyimpan objek Ability pada objek Player. Kekurangannya ialah banyak subclass yang harus didefinisikan masing-masing.

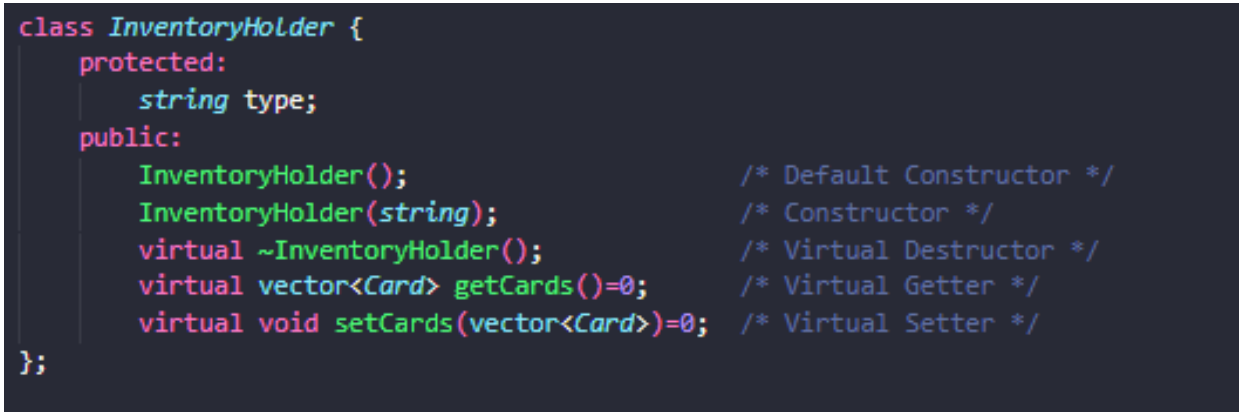
iv) cardValue

Kelas cardValue menggunakan konsep inheritance dan polymorphism untuk membuat tipe objek umum yang dapat diimplementasikan oleh objek Combination. Objek combination ini digunakan untuk meng-*evaluate* kartu dan menemukan kombo tertinggi yang dimiliki setiap player yang nantinya akan menentukan pemenang di tiap ronde. Kelebihan dari cara ini adalah pencarian kombinasi dimudahkan dengan menggunakan metode - metode yang tersedia di objek Combination.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

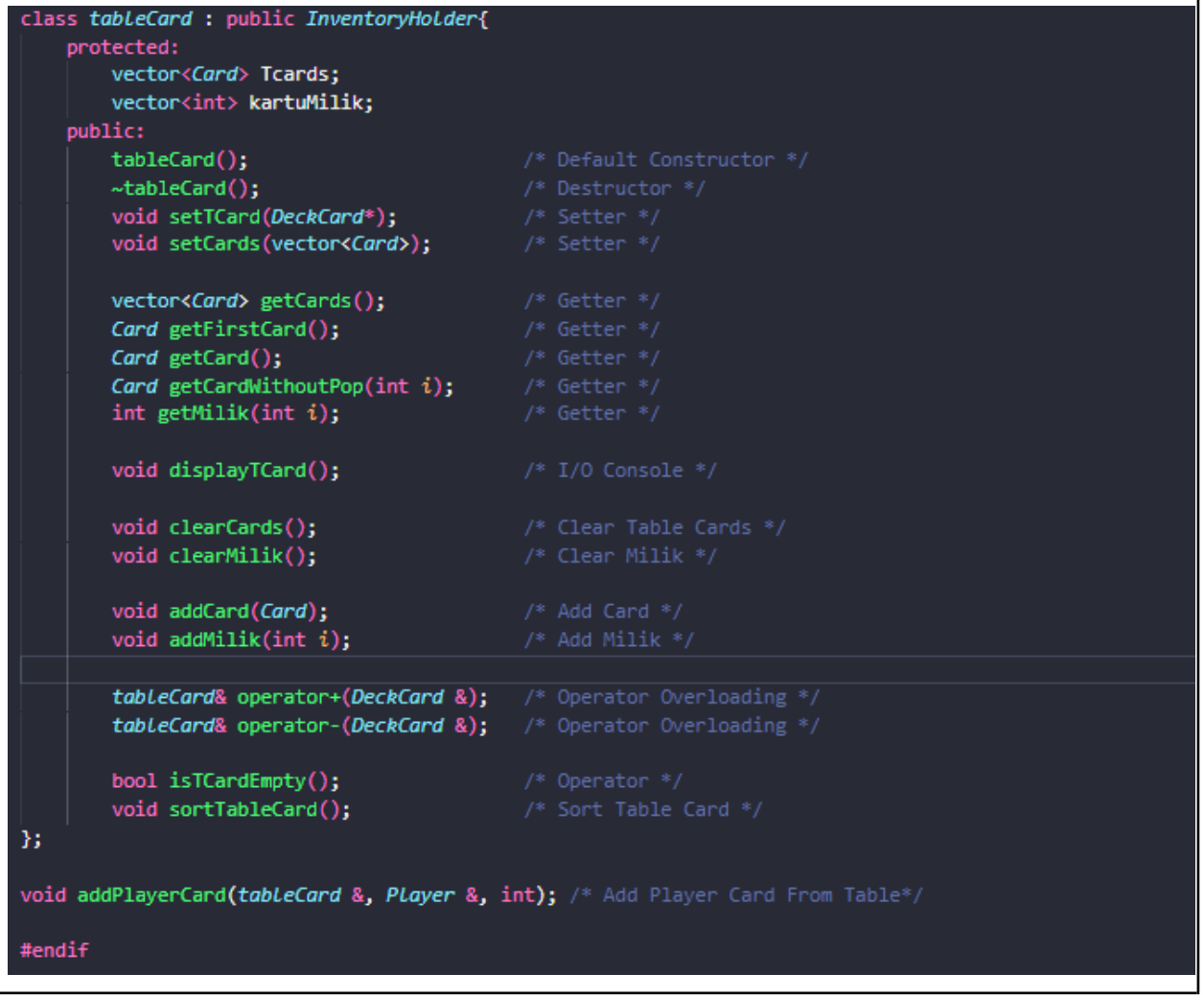
Kami mengimplementasikan *inheritance* dan *polymorphism* hampir ke seluruh kelas pada program ini. Salah satu contohnya ada pada kelas InventoryHolder dimana memiliki *base class* InventoryHolder dan subclassnya yaitu Player, DeckCard, dan tableCard.Kelas InventoryHolder memiliki dua buah *pure virtual function* yaitu getCards() dan setCards() yang mengembalikan atau menerima vector<Card>. Alasan kami menggunakan konsep ini adalah untuk memudahkan penggunaan

InventoryHolder	

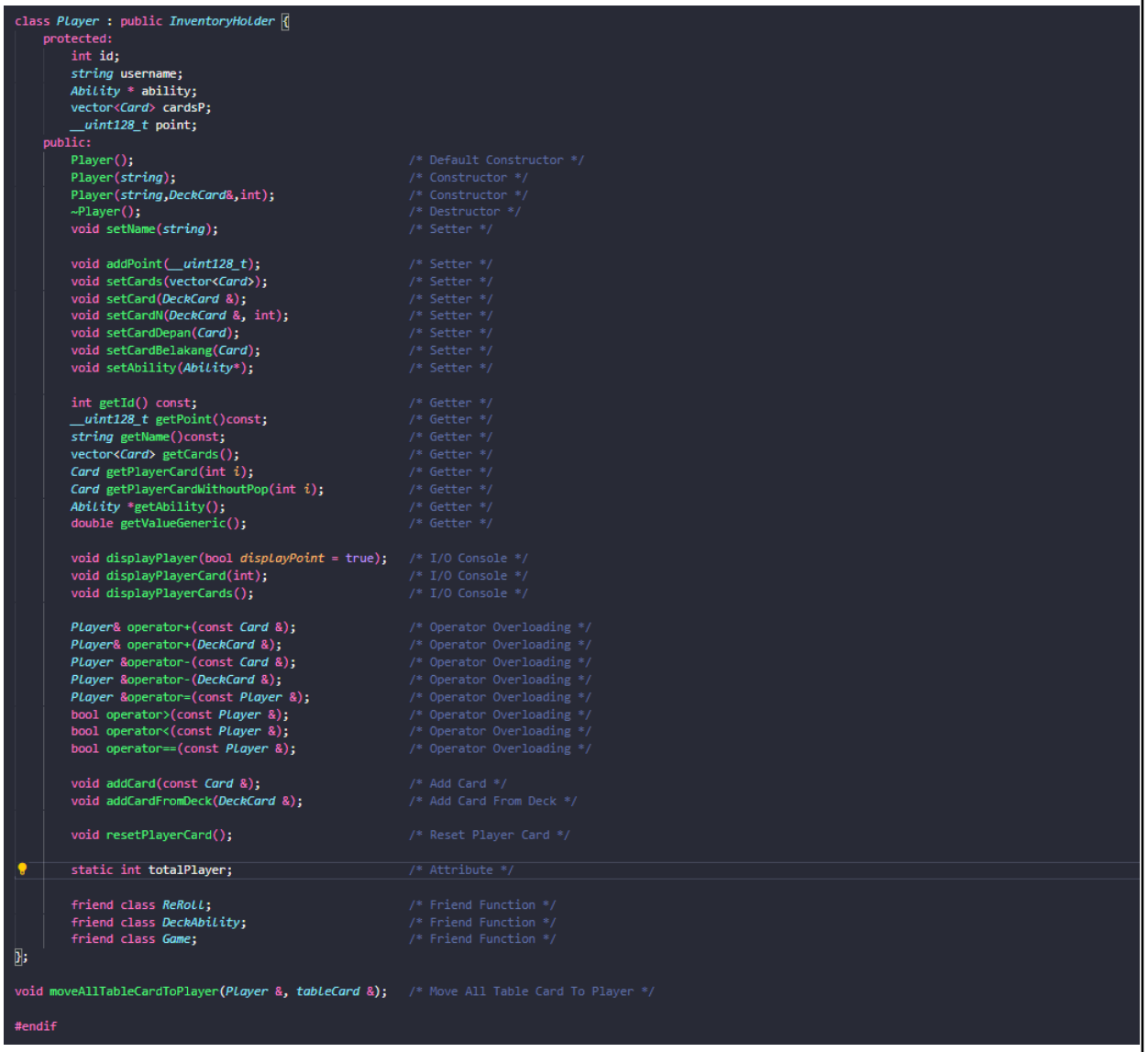
```
class InventoryHolder {
    protected:
        string type;
    public:
        InventoryHolder();                      /* Default Constructor */
        InventoryHolder(string);                /* Constructor */
        virtual ~InventoryHolder();             /* Virtual Destructor */
        virtual vector<Card> getCards()=0;      /* Virtual Getter */
        virtual void setCards(vector<Card>)=0;  /* Virtual Setter */
};
```

DeckCard	

```
class DeckCard : public InventoryHolder {
    protected:
        vector<Card> cards;
    public:
        DeckCard();                                 /* Default Constructor */
        ~DeckCard();                                /* Destructor  */
        void readFromFile(string);                  /* I/O File */
        void shuffleCard();                         /* Shuffle Card */
        void setCards(vector<Card>);                /* Setter */
        void addCard(Card a);                       /* Setter */
        Card getCard();                             /* Getter */
        Card getTopCard();                          /* Getter */
        vector<Card> getCards();                    /* Getter */

        DeckCard& operator+(const Card &);          /* Operator Overloading */
        DeckCard &operator-(const Card &);          /* Operator Overloading */
        DeckCard &operator=(const DeckCard &);      /* Operator Overloading */

        void displayDeckCard();                     /* Display Deck */
        bool isDeckEmpty();                         /* Boolean Check Empty */
        void resetDeckCard();                       /* Reset Deck Card */
};

#endif
```

tableCard	

```
class tableCard : public InventoryHolder{
    protected:
        vector<Card> Tcards;
        vector<int> kartuMilik;
    public:
        tableCard();                        /* Default Constructor */
        ~tableCard();                       /* Destructor */
        void setTCard(DeckCard*);           /* Setter */
        void setCards(vector<Card>);        /* Setter */

        vector<Card> getCards();            /* Getter */
        Card getFirstCard();                /* Getter */
        Card getCard();                     /* Getter */
        Card getCardWithoutPop(int i);      /* Getter */
        int getMilik(int i);                /* Getter */

        void displayTCard();                /* I/O Console */

        void clearCards();                  /* Clear Table Cards */
        void clearMilik();                  /* Clear Milik */

        void addCard(Card);                 /* Add Card */
        void addMilik(int i);               /* Add Milik */

        tableCard& operator+(DeckCard &);   /* Operator Overloading */
        tableCard& operator-(DeckCard &);   /* Operator Overloading */

        bool isTCardEmpty();                /* Operator */
        void sortTableCard();               /* Sort Table Card */
};

void addPlayerCard(tableCard &, Player &, int); /* Add Player Card From Table*/

#endif
```

Player	

```
class Player : public InventoryHolder {
    protected:
        int id;
        string username;
        Ability * ability;
        vector<Card> cardsP;
        __uint128_t point;
    public:
        Player();                                       /* Default Constructor */
        Player(string);                                 /* Constructor */
        Player(string,DeckCard&,int);                   /* Constructor */
        ~Player();                                      /* Destructor */
        void setName(string);                           /* Setter */

        void addPoint(__uint128_t);                     /* Setter */
        void setCards(vector<Card>);                    /* Setter */
        void setCard(DeckCard &);                       /* Setter */
        void setCardN(DeckCard &, int);                 /* Setter */
        void setCardDepan(Card);                        /* Setter */
        void setCardBelakang(Card);                     /* Setter */
        void setAbility(Ability*);                      /* Setter */

        int getId() const;                              /* Getter */
        __uint128_t getPoint()const;                    /* Getter */
        string getName()const;                          /* Getter */
        vector<Card> getCards();                        /* Getter */
        Card getPlayerCard(int i);                      /* Getter */
        Card getPlayerCardWithoutPop(int i);            /* Getter */
        Ability *getAbility();                          /* Getter */
        double getValueGeneric();                       /* Getter */

        void displayPlayer(bool displayPoint = true);   /* I/O Console */
        void displayPlayerCard(int);                    /* I/O Console */
        void displayPlayerCards();                      /* I/O Console */

        Player& operator+(const Card &);                /* Operator Overloading */
        Player& operator+(DeckCard &);                  /* Operator Overloading */
        Player &operator-(const Card &);                /* Operator Overloading */
        Player &operator-(DeckCard &);                  /* Operator Overloading */
        Player &operator=(const Player &);              /* Operator Overloading */
        bool operator>(const Player &);                 /* Operator Overloading */
        bool operator<(const Player &);                 /* Operator Overloading */
        bool operator==(const Player &);                /* Operator Overloading */

        void addCard(const Card &);                     /* Add Card */
        void addCardFromDeck(DeckCard &);               /* Add Card From Deck */

        void resetPlayerCard();                         /* Reset Player Card */

        static int totalPlayer;                         /* Attribute */

        friend class ReRoll;                            /* Friend Function */
        friend class DeckAbility;                       /* Friend Function */
        friend class Game;                              /* Friend Function */
};

void moveAllTableCardToPlayer(Player &, tableCard &);   /* Move All Table Card To Player */

#endif
```

2.2. Method/Operator Overloading

Pada program, terdapat penggunaan konsep Operator Overloading pada beberapa kelas seperti DeckCard, Player, dan tableCard. Konsep Method Overloading digunakan ketika terdapat beberapa fungsi yang berbeda namun memiliki logika penggunaan yang sama. Perbedaan antara fungsi tersebut ditentukan oleh signature-nya. Sementara itu, Operator Overloading digunakan untuk mempermudah penulisan kode. Contohnya, dalam melakukan pemindahan elemen dari tableCard ke DeckCard, dapat dituliskan dengan mudah menggunakan <tableCard>+<DeckCard> yang merupakan operator overloading.

```
tableCard& operator+(DeckCard &);//Menambanh kartu di table card dengan mengambil kartu dari DeckCard

tableCard& operator-(DeckCard &);//Mengurangin kartu di table card dan mengembalikan kartu ke DeckCard
```

```
Player& operator-(const Card &);//Menambanh kartu di Player dengan mengambil kartu dari DeckCard
Player& operator-(DeckCard &);//Mengurangi kartu di Player dan mengembalikan kartu ke DeckCard
Player& operator=(const Player &);//assignment dari Player A ke Player this
```

```
DeckCard& operator+(const Card &); //Menambahkan kartu ke DeckCard
DeckCard& operator-(const Card &); //Mengurangi kartu tertentu dari DeckCard
DeckCard& operator=(const DeckCard &); //Assignment dari DeckCard A ke DeckCard this
```

2.3. Template & Generic Classes

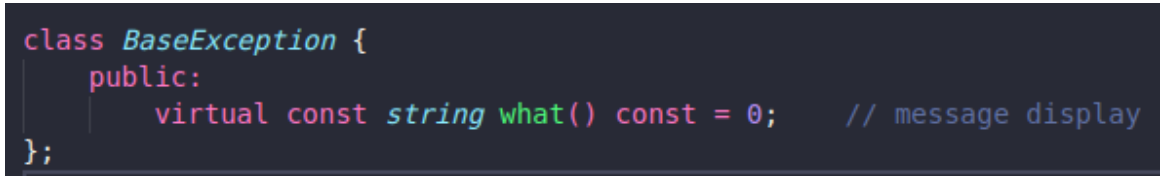
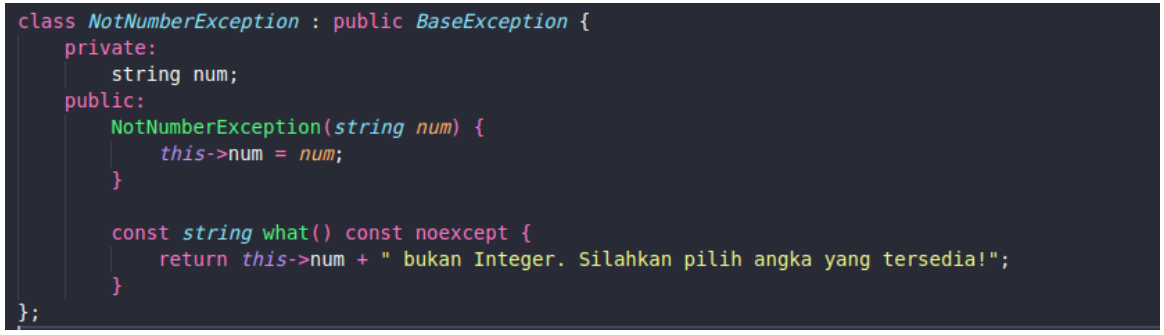
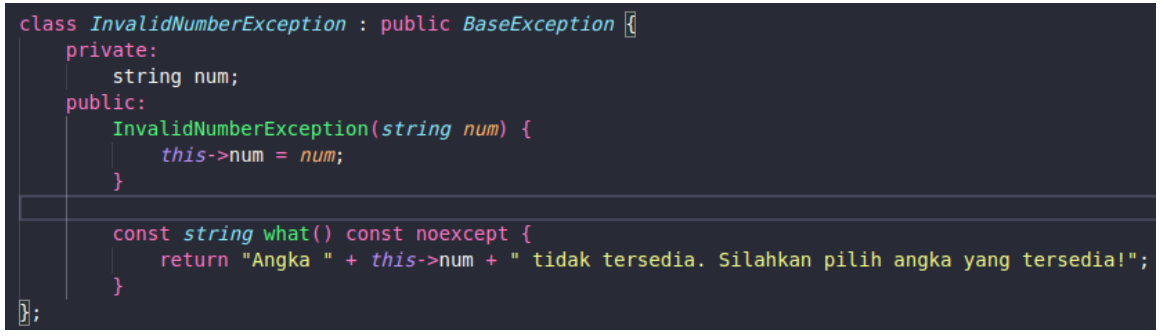
Kami mengimplementasikan Generic Function pada program ini. Generic function ini dapat menerima vector, deque, dan container lainnya serta isi container yang berbeda-beda (dapat berupa objek) yang mengembalikan element yang memiliki nilai terbesar di dalam container.Nilai terbesar ini didapatkan dengan membandingkan nilai di dalam container satu-persatu.Tiap element dibandingkan dengan tipe nilai yang berbeda tergantung tipe objeknya.

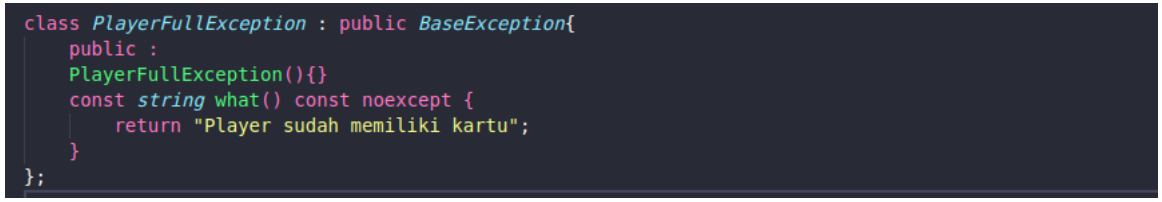
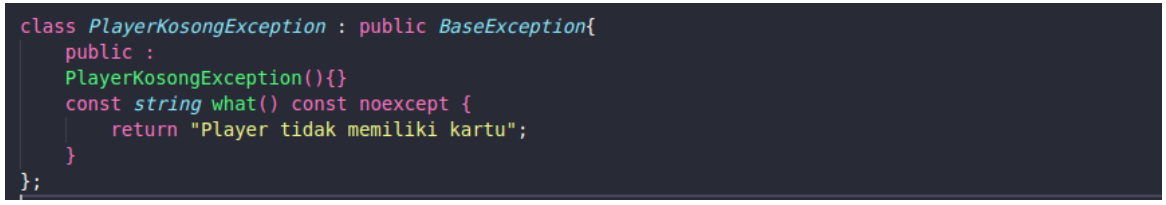
Berikut contohnya :

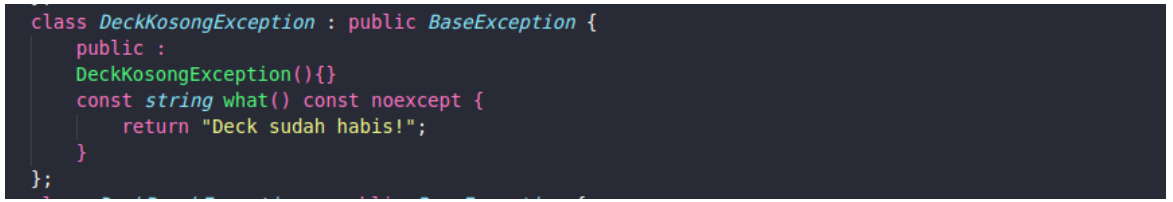
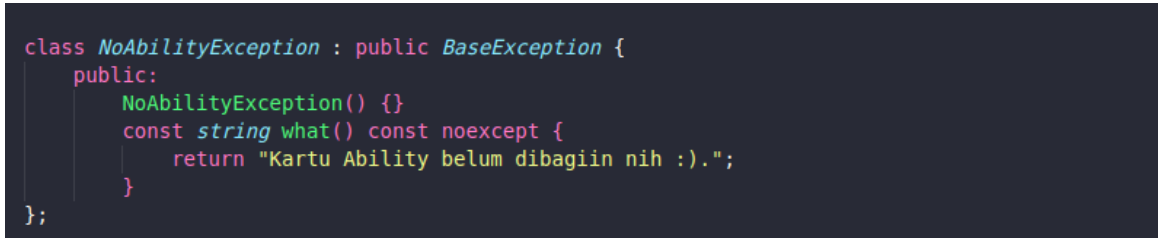
```cpp
template <typename Container>
typename Container::value_type getMaxValue(const Container &container)
{
    typename Container::value_type max_val = container.front();
    for (const auto &val : container)
    {
        if (val > max_val)
        {
            max_val = val;
        }
    }
    return max_val;
}

template<typename Container>
void sort_container(Container& container) {
    std::sort(container.begin(), container.end());
}
```

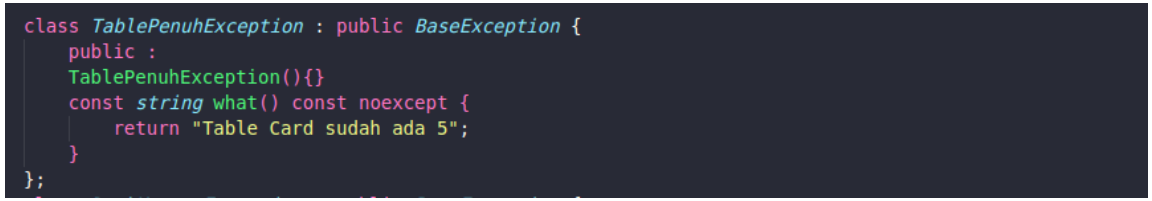
2.4. Exception

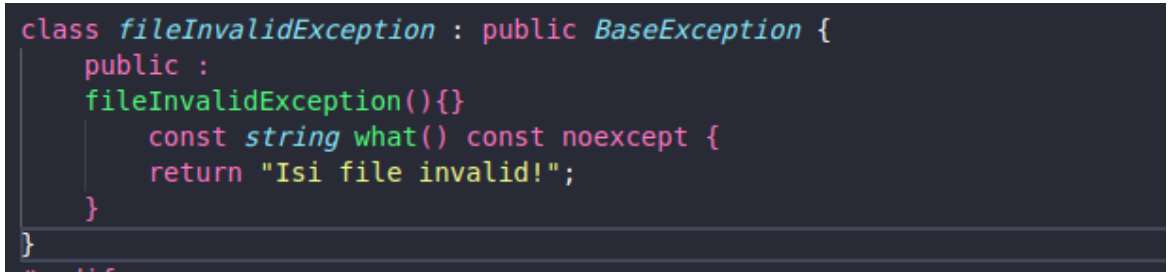
Konsep Exception hampir selalu digunakan di setiap kelas karena seringkali terdapat kasus yang memerlukan penanganan kesalahan pada saat program dieksekusi. Penanganan kesalahan tersebut dapat membuat program menjadi rumit dan rentan terhadap kesalahan pada implementasinya. Oleh karena itu, C++ menyediakan fitur exception untuk menangani kesalahan yang terjadi saat runtime menggunakan sintaksis throw, try, dan catch. Sintaksis throw berfungsi mirip dengan sintaksis return, di mana throw digunakan untuk mengembalikan nilai exception. Namun, jika suatu method selesai dengan throw, maka method tersebut dikatakan selesai secara abnormal. Untuk mengakses exception yang di-throw oleh suatu method, kita perlu menempatkan block method tersebut dalam sintaksis try dan menggunakan sintaksis catch untuk menangkap objek exception yang di-throw.

BaseException	
NotNumberException	
InvalidNumberException	

InvalidCommandEException	
PlayerFullException	
PlayerKosongException	

DeckKosongException	
DeckPenuhException	
NoAbilityException	
TableKosongException	

TablePenuhException	
NoCardException	
UsedCardException	

AbilityOffException	
fileInvalidExceeption	

2.5. C++ Standard Template Library

Pustaka STL C++ sering digunakan di dalam kelas-kelas, dengan salah satu library yang paling sering dipakai adalah <string> karena seringkali digunakan untuk menyimpan atribut bertipe string. Library lain yang juga digunakan adalah <iostream> dan <fstream> untuk melakukan input dan output ke dalam file. Penggunaan STL bertujuan untuk menggunakan fungsi-fungsi yang telah tersedia di dalam pustaka STL sehingga tidak perlu membuat implementasi sendiri. Tipe koleksi STL vector juga digunakan karena mudah digunakan dan dapat diubah ukurannya. STL <algorithm> juga digunakan dalam proses melakukan shuffle DeckCard untuk memudahkan proses acak kartu dan bisa lebih menekankan pada jalan kerja program dan kombinasi kartu.

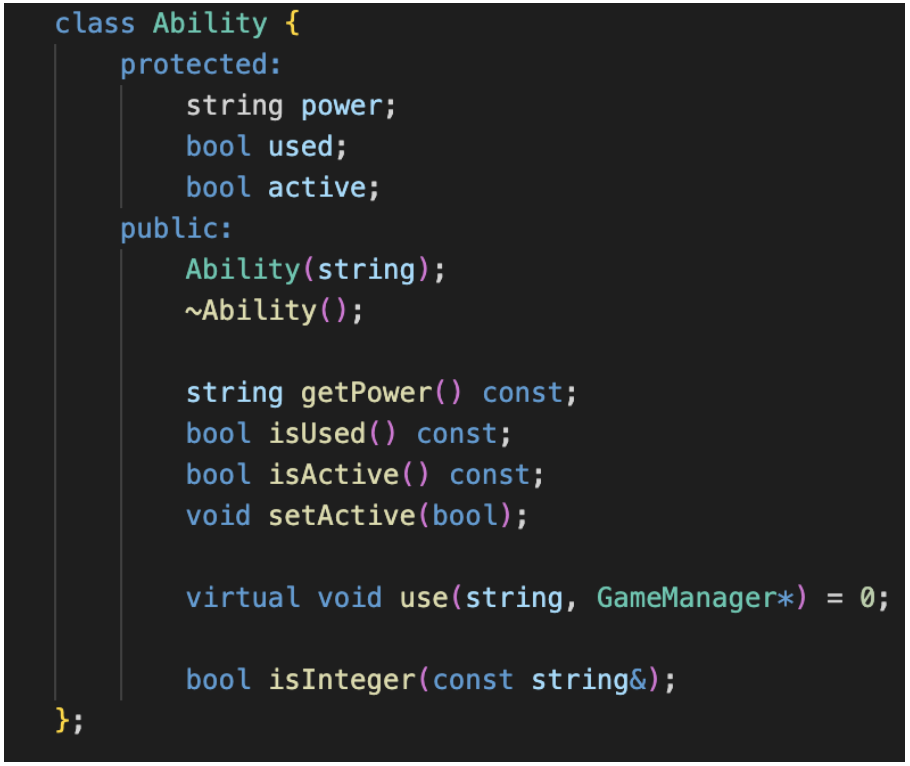
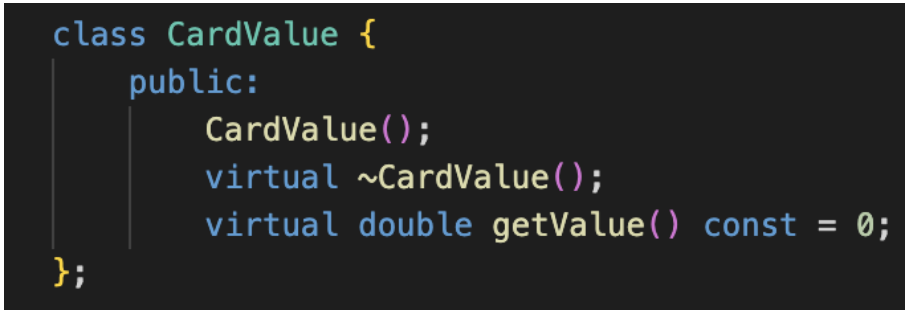
<fstream>

```
void DeckCard::readFromFile(string filename){
    fstream f;
    vector<Card> temp1234;
    set<string> temp1;
    f.open(filename,ios::in);
    int lineCount=1;
    while(!f.eof()){
        string line;
        string cardColor;
        int cardNum;

        f>>line;
        if(line.size()>3){
            throw fileInvalidException(lineCount);
        }
        switch (line[0])···

        switch (line[1])···

···

        int copyCardNum=cardNum;
        string copyCardColor=cardColor;
        temp1.insert(to_string(copyCardNum)+copyCardColor);
        temp1234.push_back(Card(cardNum,cardColor));
        if(temp1.size()!=temp1234.size()){
            throw fileInvalidDuplicateException(lineCount);
        }
        lineCount++;
    }
    f.close();
    if (temp1234.size()!= 52){
        throw fileInvalidUkuranException();
    }
    cards=temp1234;
}
```

<algorithm>	`void DeckCard::shuffleCard(){`
	`  std::random_device rd;`
	`  std::mt19937 g(rd());`
	`  std::shuffle(cards.begin(), cards.end(), g);`
	`}`

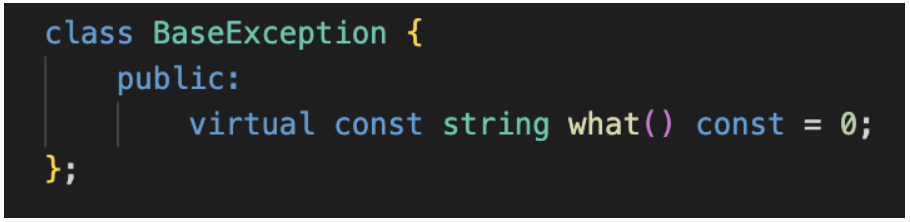
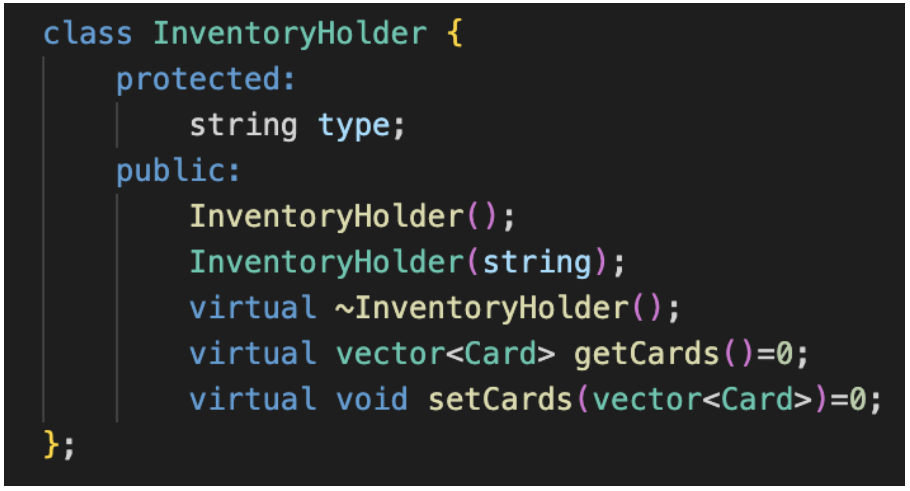
2.6. Konsep OOP lain

Kami menerapkan *Abstract Base Class* pada beberapa kelas. Kelas - kelas tersebut adalah kelas *ability*, kelas *card value*, kelas *Base Exception*, serta kelas *Inventory Holder*. Kelas - kelas abstrak tersebut berguna sebagai cetak biru dari kelas - kelas turunannya.

Abstract Base Class Ability	
Abstract Base Class cardValue	

```
class Ability {
    protected:
        string power;
        bool used;
        bool active;
    public:
        Ability(string);
        ~Ability();

        string getPower() const;
        bool isUsed() const;
        bool isActive() const;
        void setActive(bool);

        virtual void use(string, GameManager*) = 0;

        bool isInteger(const string&);
};
```

```
class CardValue {
    public:
        CardValue();
        virtual ~CardValue();
        virtual double getValue() const = 0;
};
```

Abstract Base Class BaseException	
Abstract Base Class InventoryHolder	

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Generic Class

Kami mengimplementasikan *generic function* getMaxValue yang dapat menerima input vektor deque dan kontainer lainnya. Fungsi ini juga dapat menerima isi kontainer yang berbeda - beda seperti integer, float, dan object. Fungsi ini akan mengembalikan nilai terbesar di dalam kontainer yang didapatkan dengan membandingkan nilai di dalam kontainer satu persatu. Setiap elemen dibandingkan berdasarkan jenis tipe yang ditampung kontainer.

3.1.2. Game Kartu Lain

Kami membuat game lain, yakni Cangkulan. Cara permainan Cangkulan adalah sebagai berikut.

1. Permainan terdiri dari 4 orang
2. Masing – masing pemain dibagikan tujuh kartu. Sisanya disimpan sebagai kartu ambilan (cangkulan)
3. Pemain pertama bebas membuang kartu ke meja
4. Pemain berikutnya (mengikut queue) harus membuang kartu yang sama warnanya dengan kartu pemain tadi (Saling besar-besaran)
5. Setelah semua pemain dalam satu putaran membuang satu kartu, maka pemain yang membuang kartu yang paling besar angkanya dialah yang menjadi pemain pertama putaran berikutnya, dan queue giliran berdasarkan urutan dari kartu terbesar
6. Pemain putaran berikutnya membuang kartu ke meja. Begitu terus ulang nomor 3. Pemain yang lebih dahulu habis dialah pemenangnya
7. Jika pemain tidak memiliki kartu dengan warna yang sama. Maka dia harus mengambil kartu dari tumpukan kartu ambilan satu per satu sampai menemukan kartu dengan warna yang sama. Kemudian membuang ke meja
8. Jika kartu ambilan habis, maka pemain tersebut harus mengambil kartu di atas meja yang sedang dimainkan tersebut

Dengan menggunakan konsep OOP, untuk mengerjakan bonus ini tidak dibutuhkan fungsi tambahan yang banyak karena objek/ kelas sudah ada dari permainan sebelumnya. Berikut beberapa fungsi tambahan yang dibutuhkan untuk membuat game Cangkulan.

Fungsi Tambahan di gameManager.hpp	

```
/* BONUS */

int inputCangkul(int, int);                                 /* I/O Console */
void displayPlayerCardWithColor(Player &, string);          /* I/O Console */
void CheckWin2(deque<Player> &);                            /* Validator */
void evaluateQueue(tableCard &);                            /* Validator */
vector<int> countPlayerCardWithColor(Player &, string); /* Getter */
int findIdxWithId(int);                                     /* Getter */
```

Dengan kekuatan OOP, kita dapat membuat banyak game dengan karakteristik yang mirip. Dengan menambahkan beberapa fungsi tambahan saja, kita dapat membuat kustomisasi dari objek-objek yang sudah dibuat.

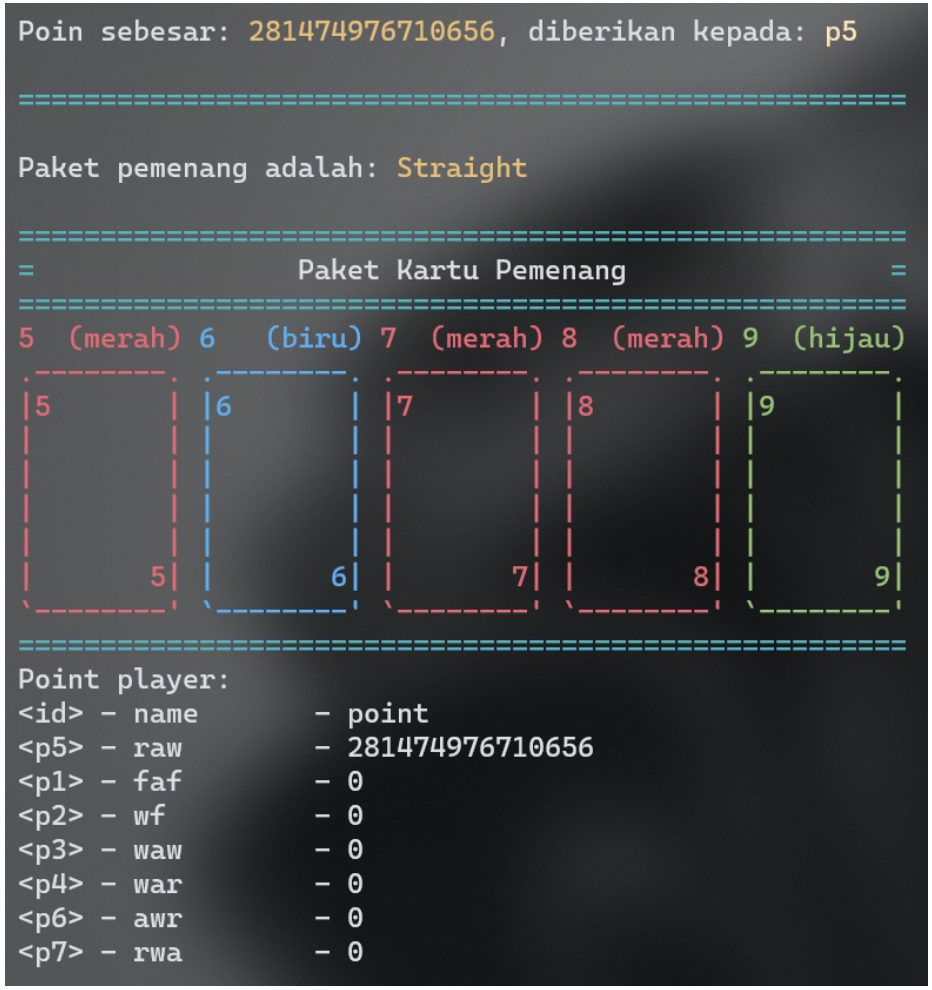
3.2. Bonus Kreasi Mandiri

3.2.1. CLI (Command Line Interface)

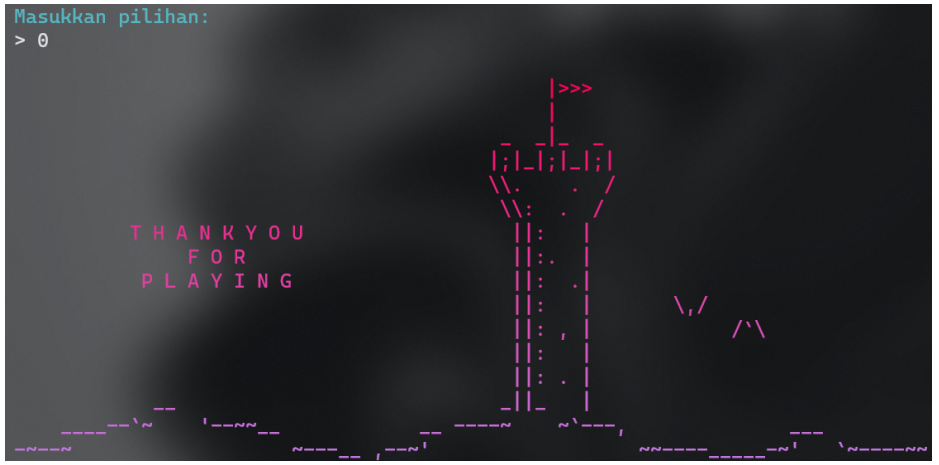
Kami mengimplementasikan *Command Line Interface* (CLI) yaitu mengontrol baris perintah serta mempercantik penampilan antarmuka dari permainan. Kami juga mengimplementasikan penggunaan RGB colour untuk mempercantik tampilan output. Selain itu, kami juga menggunakan *splash screen* pada awal penampilan program dan *interface* kartu untuk penampilan *table card* dan *player card*.

Contoh Visual	

Contoh Visual	

Contoh Visual	

Contoh Visual	

Contoh Visual	

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
InventoryHolder	13521104,13521055	13521055, 13521081, 13521072, 13521103, 13521104
DeckCard	13521104	13521055, 13521081, 13521072, 13521103, 13521104
Player	13521104, 13521055	13521055, 13521081, 13521072, 13521103, 13521104
tableCard	13521104, 13521055	13521055, 13521081, 13521072, 13521103, 13521104

Game	13521055, 13521081	13521055, 13521081, 13521072, 13521103, 13521104
GameManager	13521055, 13521081, 13521104	13521055, 13521081, 13521072, 13521103, 13521104
Generic	13521055, 13521081, 13521104	13521055, 13521081, 13521072, 13521103, 13521104
Exception	13521055, 13521081, 13521104	13521055, 13521081, 13521072, 13521103, 13521104
CardValue	13521055, 13521072, 13521103	13521055, 13521081, 13521072, 13521103, 13521104
Card	13521055, 13521072, 13521103, 13521104,13521072	13521055, 13521081, 13521072, 13521103, 13521104
Combination	13521055, 13521072,13521103	13521055, 13521081, 13521072, 13521103, 13521104
Ability	13521081	13521055, 13521081, 13521072, 13521103, 13521104
Bonus Cangkul	13521055	13521055, 13521081, 13521072, 13521103, 13521104

Form Asistensi

Kode Kelompok : C0K

Nama Kelompok : Anak Didikan Mama

1. 13521055 / Muhammad Bangkit Dwi Cahyono
2. 13521072 / Irsyad Nurwidianto Basuki
3. 13521081 / Bagas Aryo Seto
4. 13521103 / Aulia Mey Diva Anandya
5. 13521104 / Muhammad Zaydan Athallah

Asisten Pembimbing : Fabian Savero Diaz Pranoto

1. Konten Diskusi

- Pertanyaan : Kalau di meja terdapat straight flush tertinggi yaitu 13M, 12M, 11M, 10M, dan 9M yang artinya semua player memiliki kombinasi tertinggi di meja dan player A memiliki 8M, apakah player A dapat dikatakan memiliki straight flush lain?
- Jawaban : iya, player A memiliki straight flush lain 12M, 11M, 10M, 9M, dan 8M.
- Pertanyaan : Kalau di meja terdapat full house yaitu 12M, 12B, 12K, 11M, 11H dan merupakan kombinasi tertinggi yang dimiliki semua player serta player A memiliki 11B. Apakah player A dapat dikatakan memiliki Full House lain?
- Jawaban : iya, player A memiliki full house lain 11M, 11B, 11H, 12M, 12K.
- Pertanyaan : Kalau misal si player sama kartunya beda type, kek playernya pakai deque dan kartunya pakai vector, gimana caranya buat nge-generic class nya?
- Jawaban : coba bikin template dua kelas T dan U nanti implementasi di kode nya T<U>

2. Tindak Lanjut

- Untuk menindaklanjuti kemungkinan kombinasi tertinggi semua player berada pada meja, kami sedikit mengubah fungsi yang awalnya mereturn vector of vector card, menjadi pair yang memiliki vector of vector card dan vector of double yang nantinya vector sudah terurut value yang nantinya bila terdapat kombinasi tertinggi, akan di evaluate lagi dengan membandingkan value yang sudah disimpan di dalam pair dan mengambil kombinasi tertinggi pertama yang valuenya lebih kecil dari kombinasi di meja.
- Untuk generic class player dan kartu, menggunakan template 2 kelas