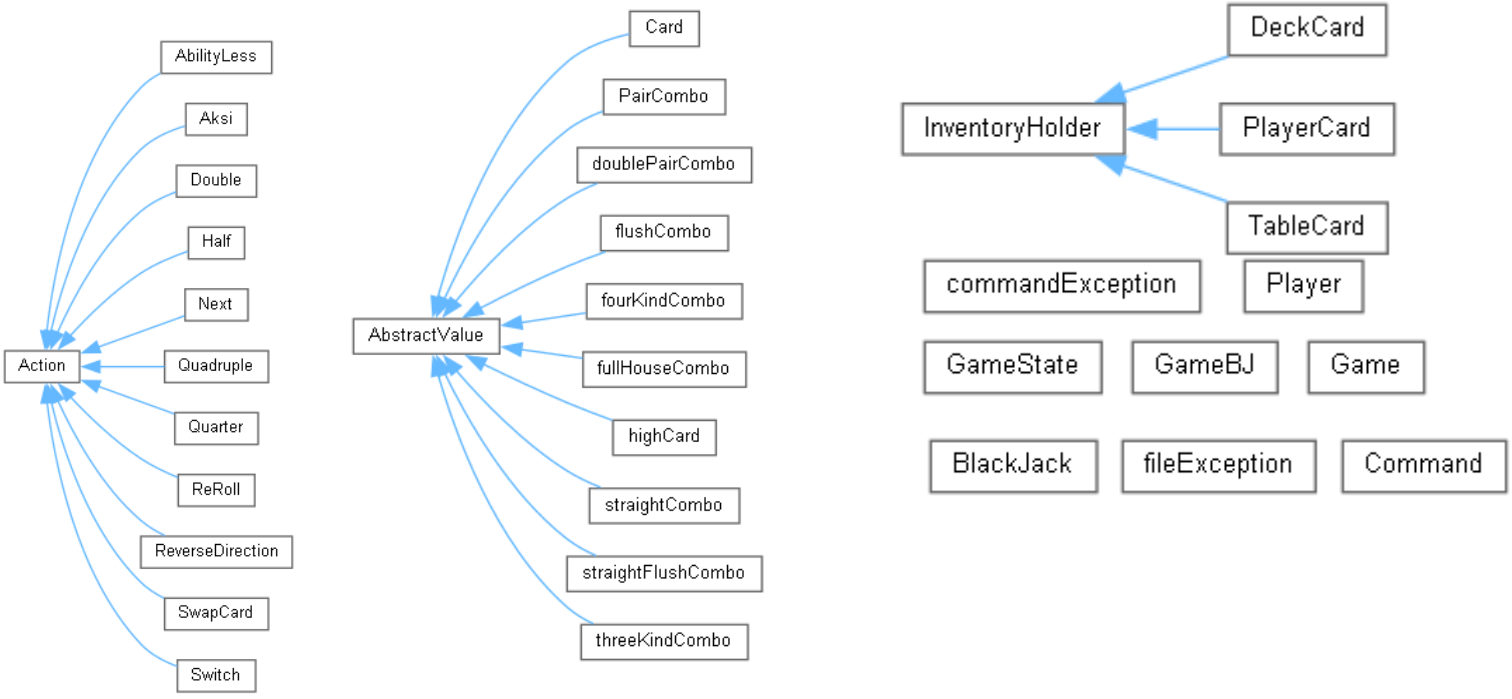
Kode Kelompok : CUM
Nama Kelompok : cumlaude
1. 13521065 / Mutawally Nawwar
2. 13521080 / Fajar Maulana Herawan
3. 13521099 / Vieri Fajar Firdaus
4. 13521122 / Ulung Adi Putra
5. 13521146 / Muhammad Zaki Amanullah
Asisten Pembimbing : Reihan Andhika Putra

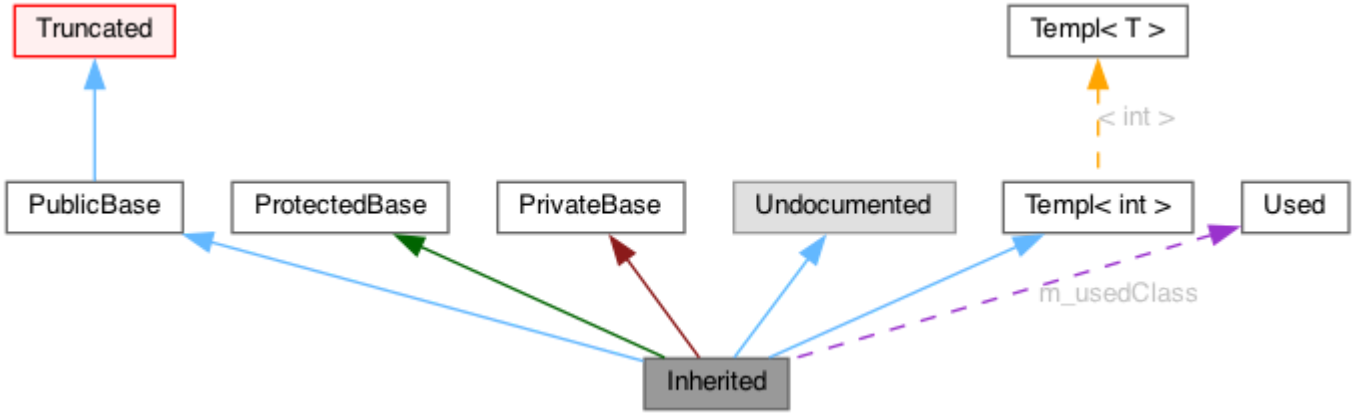
1. Diagram Kelas

a. Class Hierarchy



b. Class Diagram

Graph Legend



Setiap kotak pada graf di atas memiliki arti sebagai berikut:

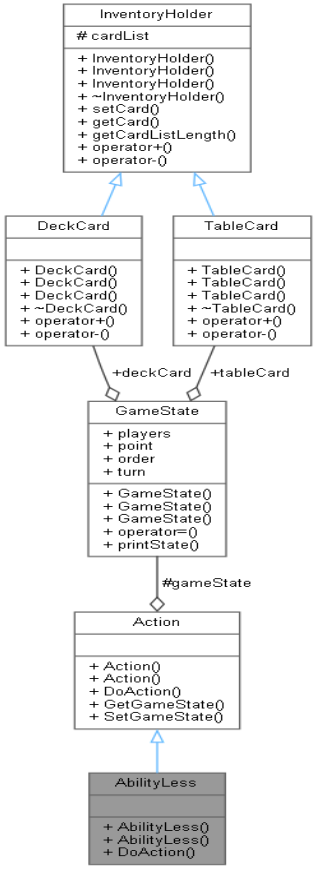
- Kotak berwarna abu-abu penuh menandakan *struct* atau *class* yang graf-nya sedang di buat
- Kotak dengan border hitam menandakan *struct* atau *class* yang terdokumentasi
- Kotak dengan border abu-abu menandakan *struct* atau *kelas* yang tidak terdokumentasi
- Kotak dengan border merah menandakan *struct* atau *kelas* yang terdokumentasi yang memiliki *inheritance* lebih dalam namun tidak ditampilkan karena melebihi boundary yang telah ditentukan.

Setiap tanda panah dalam graf memiliki arti sebagai berikut:

- Tanda panah biru menggambarkan inheritance *public* antara dua *class*
- Tanda panah hijau menandakan *protected inheritance.*
- Tanda panah merah tua menandakan *private inheritance*.
- Tanda panah berwarna ungu dengan garis putus-putus akan digunakan apabila class digunakan oleh class lain.
- Tanda panah kuning putus-putus menandakan relasi antara instance sebuah template dan template class.

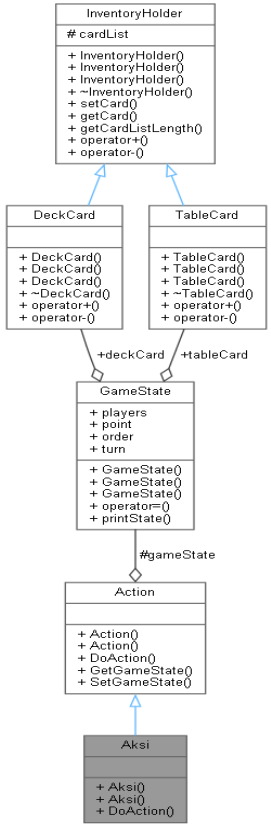
Class AbilityLess

Collaboration Graph

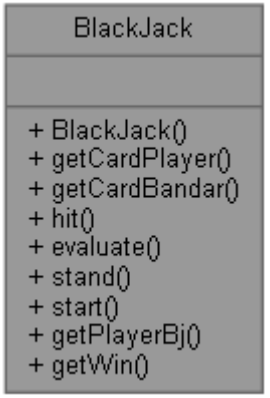


Class Aksi

Collaboration Graph

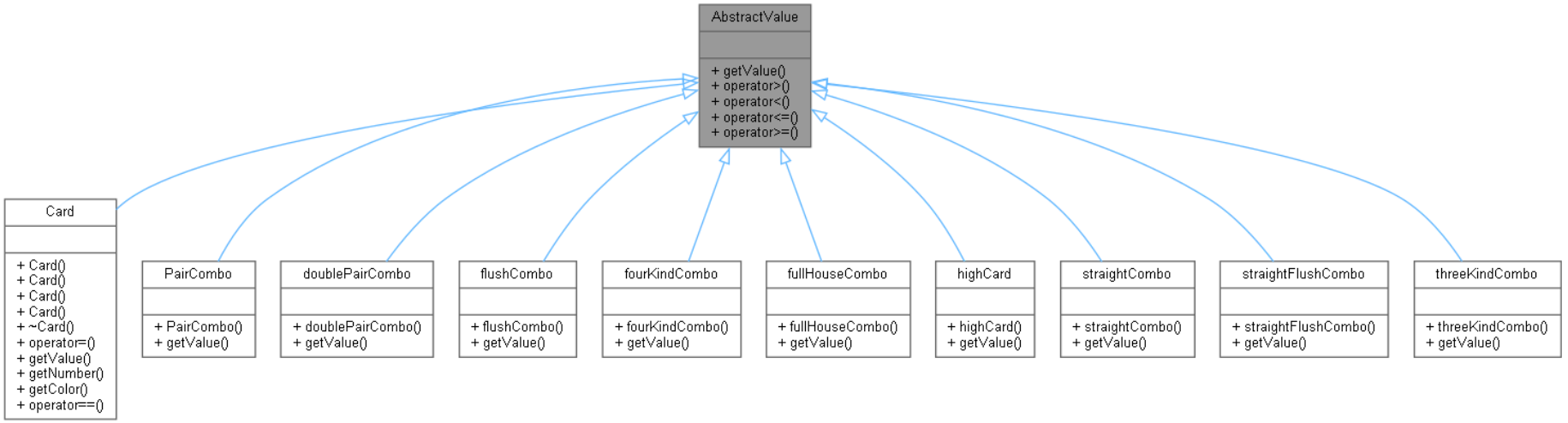


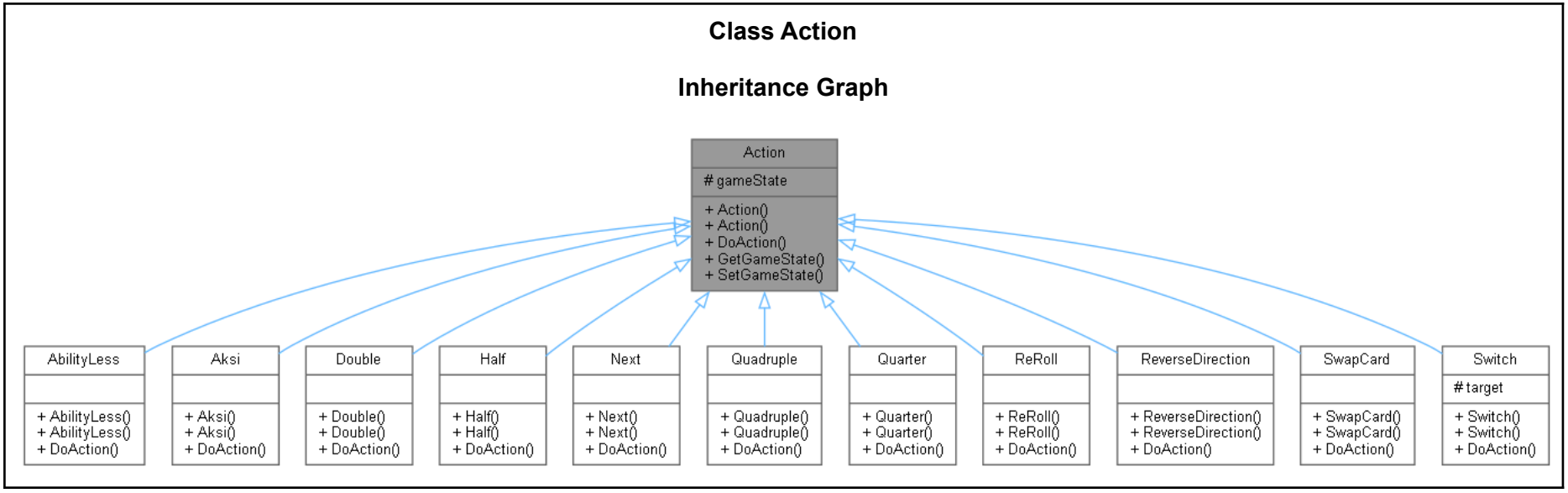
Class BlackJack



Class AbstractValue

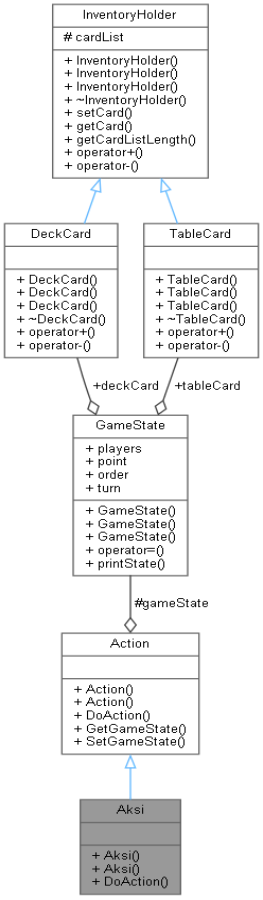
Inheritance Graph




Class Action
Inheritance Graph
Action
gameState
+ Action()
+ Action()
+ DoAction()
+ GetGameState()
+ SetGameState()
AbilityLess
+ AbilityLess()
+ AbilityLess()
+ DoAction()
Aksi
+ Aksi()
+ Aksi()
+ DoAction()
Double
+ Double()
+ Double()
+ DoAction()
Half
+ Half()
+ Half()
+ DoAction()
Next
+ Next()
+ Next()
+ DoAction()
Quadruple
+ Quadruple()
+ Quadruple()
+ DoAction()
Quarter
+ Quarter()
+ Quarter()
+ DoAction()
ReRoll
+ ReRoll()
+ ReRoll()
+ DoAction()
ReverseDirection
+ ReverseDirection()
+ ReverseDirection()
+ DoAction()
SwapCard
+ SwapCard()
+ SwapCard()
+ DoAction()
Switch
target
+ Switch()
+ Switch()
+ DoAction()

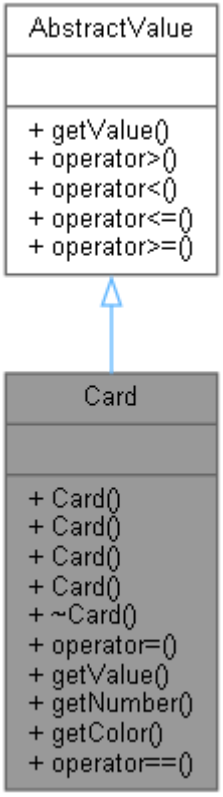
Class Aksi

Collaboration Graph

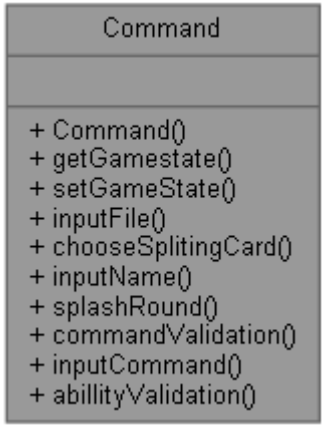


Class Card

Inheritance Graph

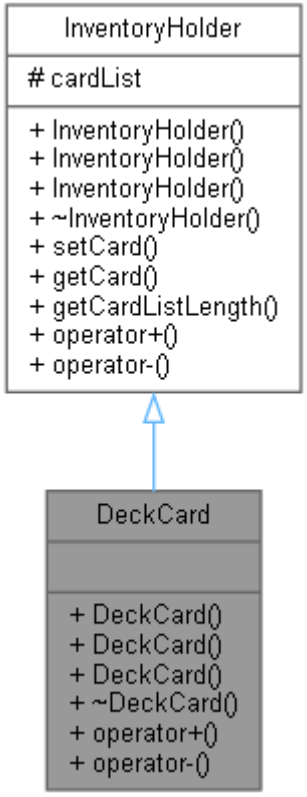


Class Command



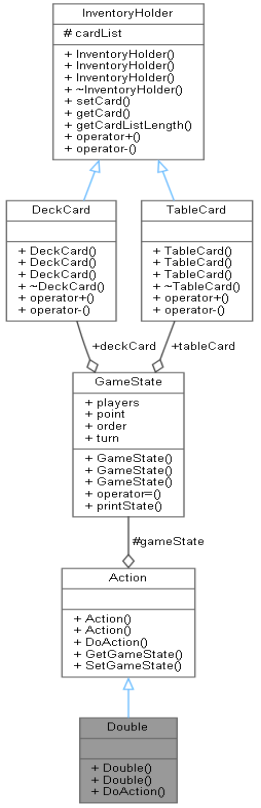
Class DeckCard

Inheritance Graph



Class Double

Collaboration Graph

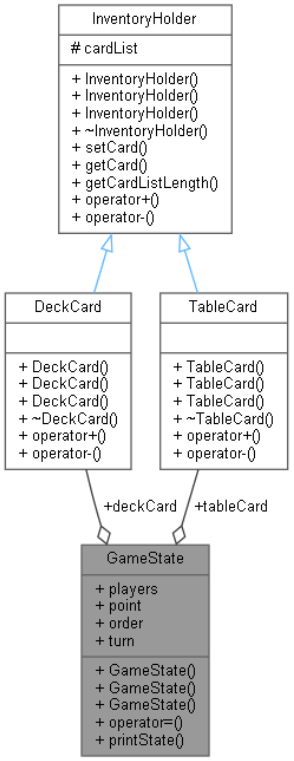


Class Game

Class GameBJ

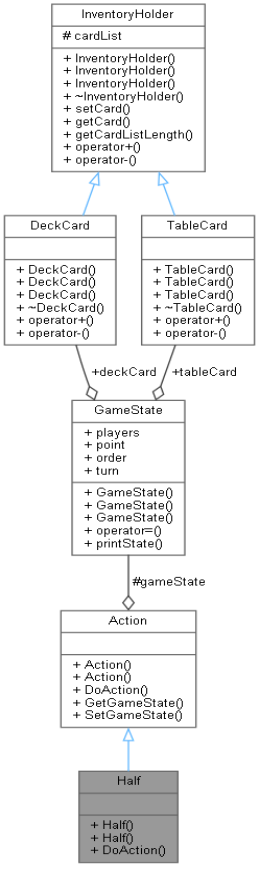
Class GameState

Collaboration Graph



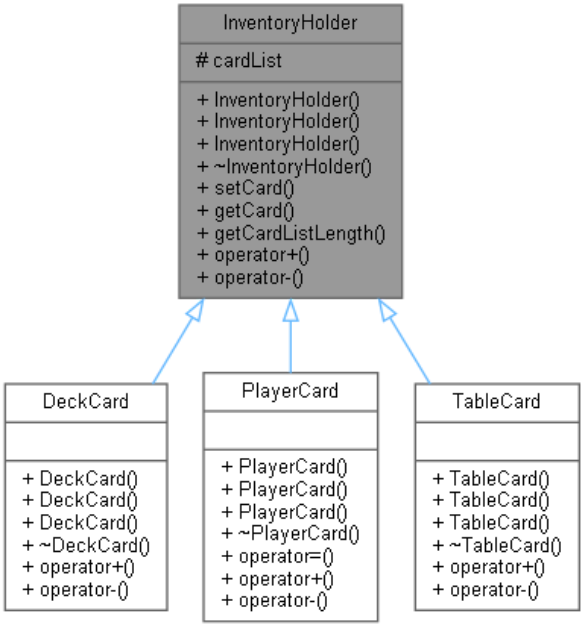
Class Half

Collaboration Graph



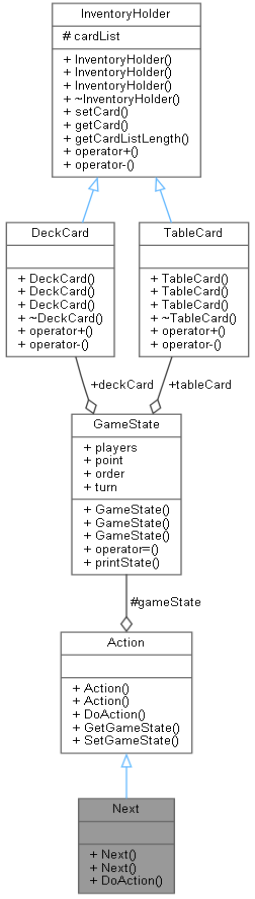
Class InventoryHolder

Inheritance Graph



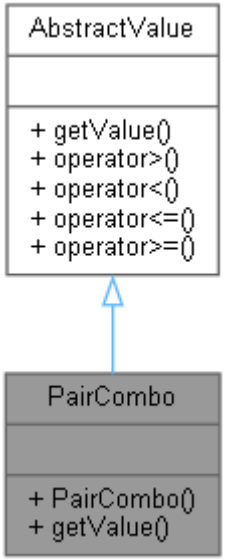
Class Next

Collaboration Graph

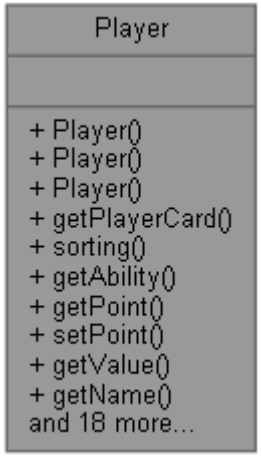


Class PairCombo

Inheritance Graph

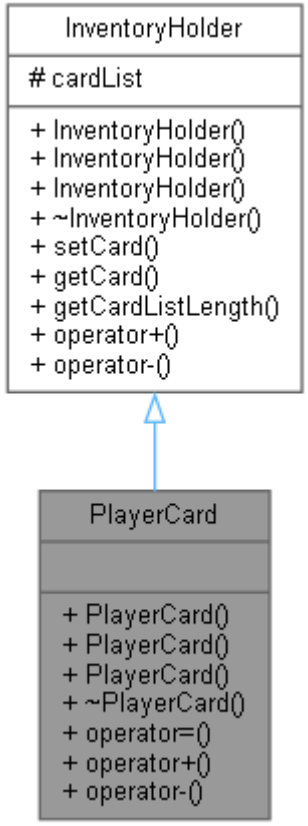


Class Player



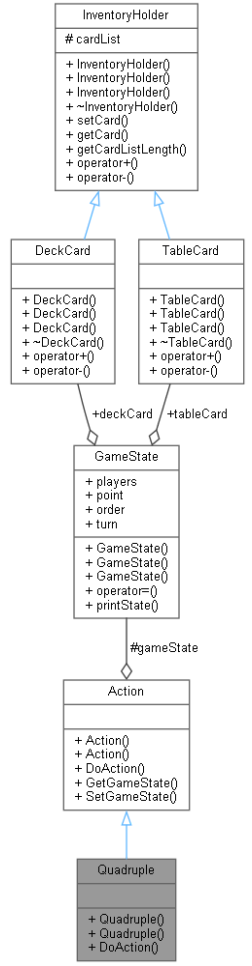
Class PlayerCard

Inheritance Graph



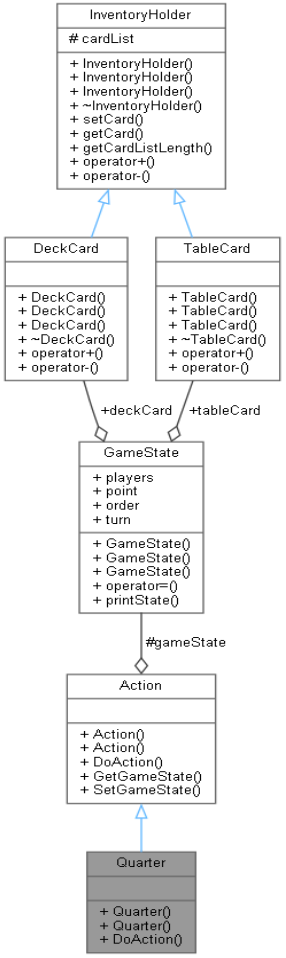
Class Quadruple

Collaboration Graph



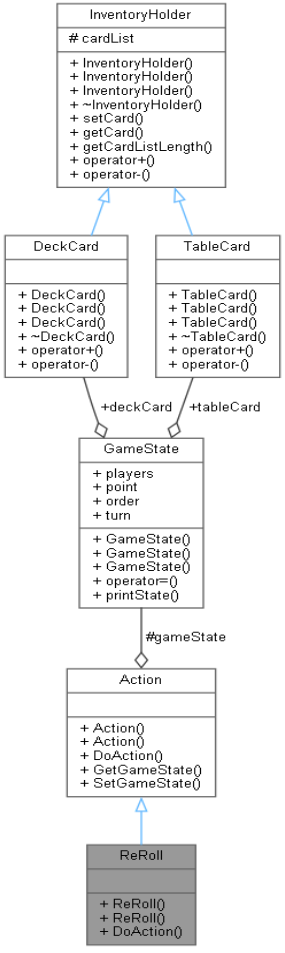
Class Quarter

Collaboration Graph



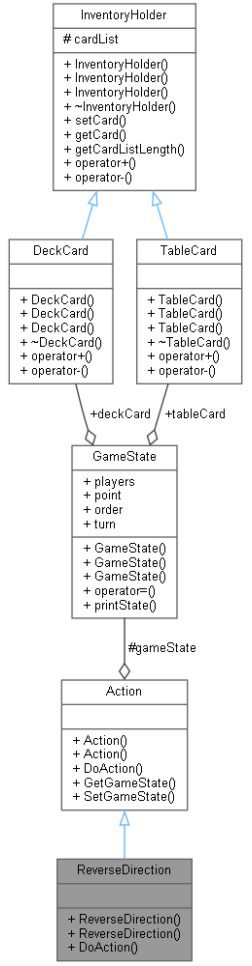
Class ReRoll

Collaboration Graph



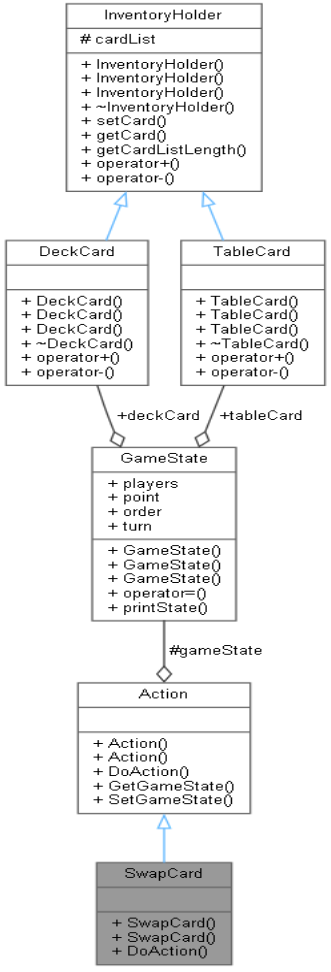
Class ReverseDirection

Collaboration Graph



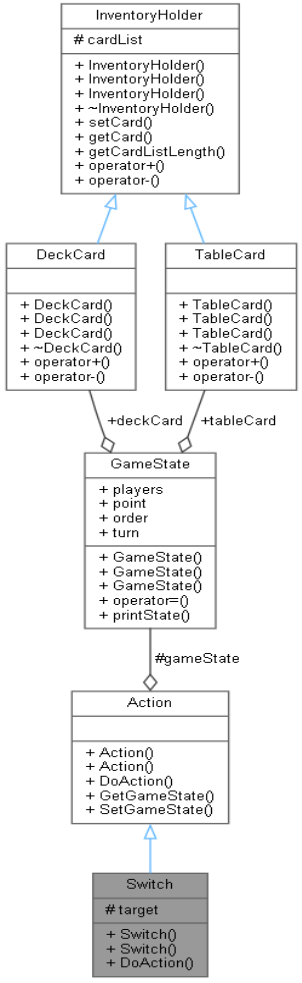
Class SwapCard

Collaboration Graph



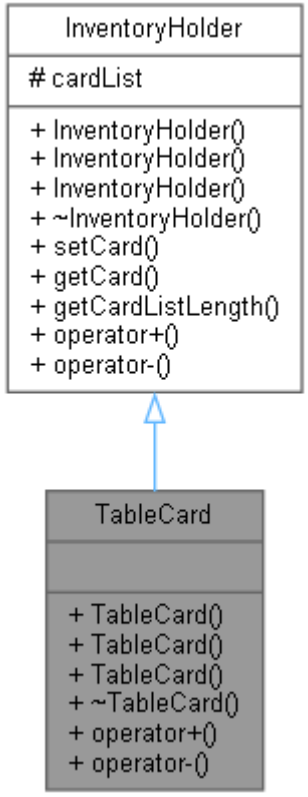
Class Switch

Collaboration Graph

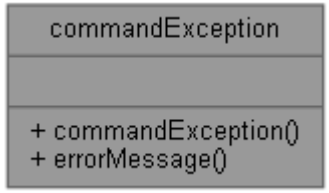


Class TableCard

Inheritance Graph

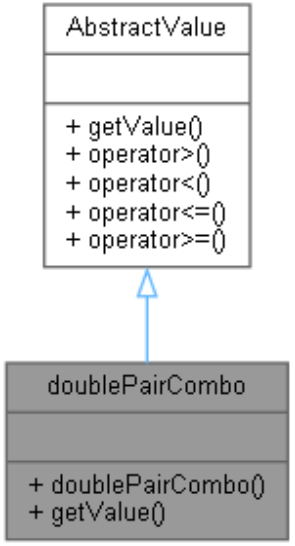


Class commandException

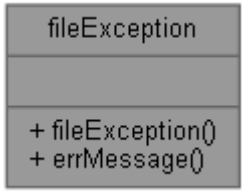


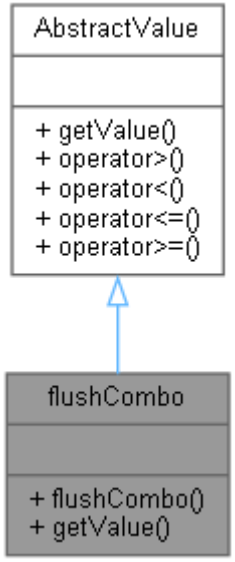
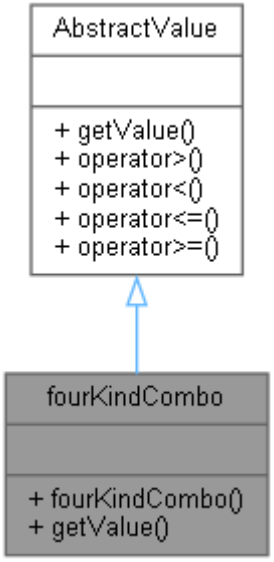
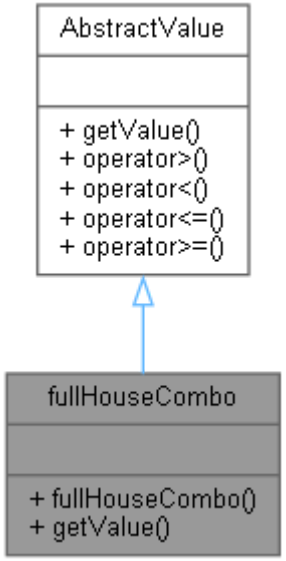
Class doublePairCombo

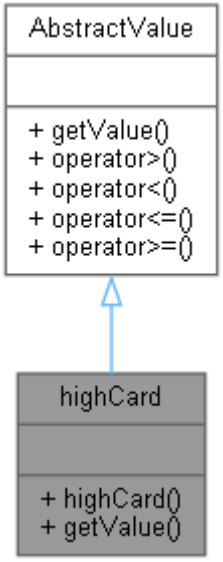
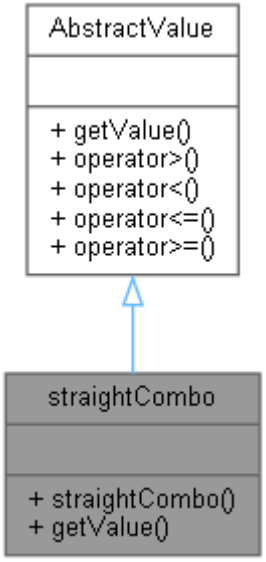
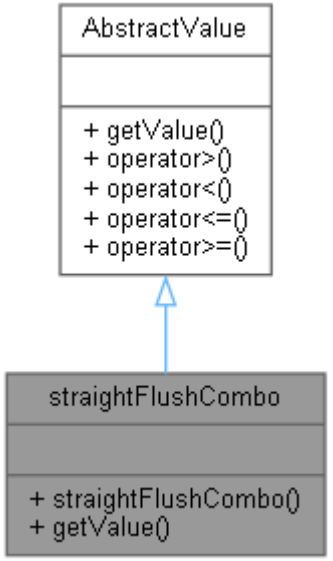
Inheritance Graph



Class fileException

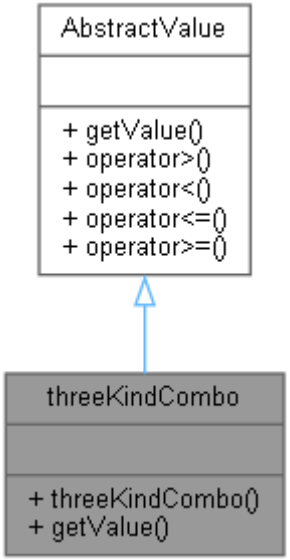


Class flushCombo	Class fourKindCombo	Class fullHouseCombo
Inheritance Graph	**Inheritance Graph**	
		

Class highCard Inheritance Graph	Class straightCombo Inheritance Graph	Class straightFlushCombo
		

Class threeKindCombo

Inheritance Graph



c. Class Design

1. Game

   Kelas Game menggunakan konsep komposisi. Kelas Game ini bertanggung jawab dalam mengatur status game dan alur permainan. Konsep komposisi pada kelas Game dipakai saat kelas game memiliki atribut gameState yang menyimpan status game ronde tersebut.Secara menyeluruh kelas game ini akan mengatur semua tata cara permainan seperti mengacak kartu,membagikan kartu, mengatur giliran pemain, dan menentukan pemenang.

   Keunggulan dari desain kelas ini adalah modularitas dan reusabilitas karena kelas-kelas yang digunakan sangat modular, maka hal tersebut akan memungkinkan untuk menggunakan kelas tersebut pada proyek lain yang memerlukan fitur yang sama atau mirip dengan permainan kartu yang dibuat. Selain itu, Kelas tersebut mampu untuk menambah atau mengubah fitur permainan dengan mudah, seperti menambahkan kemampuan baru, mengubah aturan, atau menambahkan variasi kartu sehingga membuat kelas ini flexible.

2. GameState

   Kelas Game state menggunakan konsep *composition* dan *collection*. Kelas gamestate bertanggung jawab untuk merepresentasikan status dari permainan kartu. Kelas ini menyimpan informasi seperti pemain yang berpartisipasi dalam permainan, kartu-kartu yang dimiliki oleh setiap pemain, kartu yang berada di atas meja, deck kartu yang tersisa, poin yang dimiliki, urutan giliran, dan kemampuan yang bisa digunakan oleh pemain. Konsep *composition* digunakan untuk menyimpan kumpulan objek player yang bermain pada game sedangkan kita menggunakan *composition* untuk menggunakan objek tableCard dan deckCard yang mana sebagai kartu yang ada untuk “minum”.

   Keunggulan dari desain kelas ini kita dapat mengatur setiap pemain yang bermain dengan menggunakan *collection of player*. Dengan demikian, kita dapat mengakses setiap pemain tersebut. *Collection of Player* ini disimpan di atribut *vector players*.

3. Command

   Kelas command menggunakan konsep komposisi. Kelas command bertanggung jawab untuk menangani inputan user dan mengimplementasikan *exception handling*.Selain itu, tujuan di buat nya kelas ini agar kode di main tidak *redundant* dalam algoritmanya dengan cukup memanggil *method* yang dari kelas *command* ini.

4. Exception

   Kelas *exception* ini menggunakan konsep *composition* dan *inheritance* . Konsep komposisi ini digunakan untuk mengakses gameState terkini. Kelas ini bertanggung jawab untuk mengeluarkan error saat terjadi *error* di kelas command. Keunggulan dari desain kelas ini, mudahnya kelas ini untuk digunakan oleh kelas lain. Selain itu, ojek ini mudah digunakan pada try catch block dan membuat kode untuk *exception handling* ini tidak *redundant*.

5. InventoryHolder

   InventoryHolder menggunakan polimorfisme dan pewarisan. Warisan membuat setumpuk kartu ini unik. Polimorfisme digunakan untuk menggeneralisasi tipe objek kelas anak yang dipakai di kelas lain. Kelas ini mengelola koleksi kartu. Kelas ini menambah, menghapus, dan menghitung kartu dalam koleksi. Kelas yang mewarisi dari kelas ini harus menimpa + dan - karena kelas ini membebani mereka secara murni dan virtual. Kelas Inventory Holder nantinya akan mewakili card holder (pemain) dan deck.

   Desain ini mendapat manfaat dari modularitas. Kelas abstrak dan pewarisan memungkinkan kita memberikan tugas InventoryHolder ke PlayerCard dan DeckCard. Karena setiap kelas melakukan pekerjaannya sendiri, membuat dan memelihara program menjadi lebih mudah. Fungsi virtual dalam kelas abstrak memungkinkan kelas turunan untuk mengimplementasikannya kembali. Fleksibilitas meningkat. Jadi, bergantung pada statusnya, kelas turunan dapat mengimplementasikan kemampuan kelas abstrak secara berbeda. Untuk fleksibilitas, kami menyimpan Kartu dalam vektor STL. Ini memungkinkan pengguna aplikasi menambahkan atau menghapus kartu dengan cepat dari inventaris, memberi mereka lebih banyak fleksibilitas.

6. AbstractValue

Kelas AbstractValue ini menggunakan konsep *polymorphisme* dan *Inheritance* sama seperti InventoryHolder. Konsep ini digunakan untuk membuat jenis objek yang memiliki value lebih spesifik dan menggeneralisasi tipe objek pada kelas anak yang diinstasiasi pada kelas-kelas lainnya. Kelas ini bertanggung jawab untuk menyediakan antarmuka umum untuk mengakses nilai numerik. Kelas ini memiliki satu metode murni virtual bernama getValue() yang harus diimplementasikan oleh kelas turunannya.

Polimorfisme dan operator overloading memungkinkan kelas turunan AbstractValue dengan cepat dan logis membandingkan nilai numerik dengan objek lain dari tipe yang berbeda tanpa konversi tipe. Operator < atau > akan membandingkan dua objek yang diturunkan dari AbstractValue dengan menggunakan metode getValue() mereka.

7. Action

   Kelas Action ini juga menggunakan konsep *polymorphisme* dan *inheritance*. Konsep ini digunakan untuk membuat objek seperti aksi,double,half, dan sebagainya melakukan *inheritance* dari *parent*nya yakni kelas Action. Kelas-kelas tersebut juga membuat aksi yang dilakukan di kelas action lebih spesifik.

   Keunggulan dari desain kelas ini yakni kelas turunan dari Action dapat memiliki aksi yang lebih spesifik menurut tanggung jawab masing-masing.Akan tetapi, kelas turunan tersebut tetap memiliki aksi yang umum yaitu melakukan aksi yang ada di permainan kartu.

8. Player

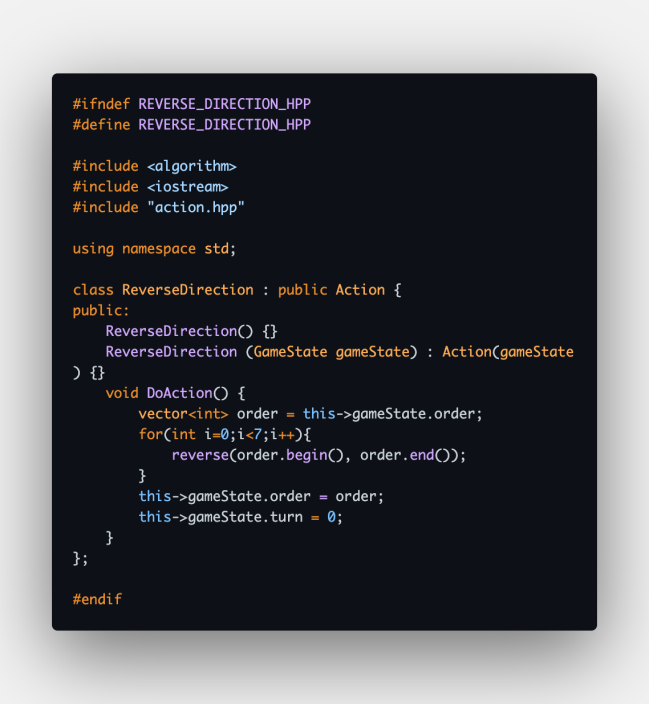
   Kelas player menggunakan konsep *composition*. Konsep ini digunakan saat kita menggunakan atribut playerCard yang berasal dari kelas playerCard. Tanggung jawab dari kelas ini yakni sebagai representasi objek pemain yang memainkan permainan. Kelas player memiliki attribute playerCard yang berarti Kartu yang dimiliki oleh player , ada juga ability untuk mendaftarkan ability yang dapat dilakukan oleh player, dan setiap player memiliki nama.

   Keunggulan dari desain kelas ini yakni adanya operator overloading >,<,== ini digunakan untuk membandingkan value dari kedua player untuk mencari pemenang dari permainan kartu. Dengan demikian, kode tidak *redundant*.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Inheritance berarti kelas turunan atau anak dapat mengambil sifat dan perilaku dari induknya. Dengan inheritance, kita dapat membuat hierarki kelas. Sedangkan dari polymorphism ini, memungkinkan suatu objek dapat memiliki beberapa tipe. Kelas yang menggunakan Inheritance dan Polymorphisme adalah pada folder ability Action (Parent), AbilityLess (Child), Double (Child), Half (Child), Next (Child), Quadruple (Child), Quarter (Child), ReRoll (Child), ReverseDirection (Child), SwapCard (Child), Switch (Child). Inheritance digunakan digunakan untuk mengabstrasikan kelas yang terbentuk serta memungkinkan untuk menggunakan polymorphism dalam konsep ini. Dengan polymorphism ini objek turunan dari parent dapat memiliki perilaku yang berbeda-beda.



```
#ifndef REVERSE_DIRECTION_HPP
#define REVERSE_DIRECTION_HPP

#include <algorithm>
#include <iostream>
#include "action.hpp"

using namespace std;

class ReverseDirection : public Action {
public:
    ReverseDirection() {}
    ReverseDirection (GameState gameState) : Action(gameState
) {}
    void DoAction() {
        vector<int> order = this->gameState.order;
        for(int i=0;i<7;i++){
            reverse(order.begin(), order.end());
        }
        this->gameState.order = order;
        this->gameState.turn = 0;
    }
};

#endif
```

```
#ifndef QUADRUPLE_HPP
#define QUADRUPLE_HPP

#include <iostream>
#include "action.hpp"
#include "next.hpp"
using namespace std;

class Quadruple : public Action {
public:
    Quadruple() {}
    Quadruple (GameState gameState) : Action(gameState) {}
    void DoAction() {
        this->gameState.point*=4;
        this->gameState.turn++;
        this->gameState.turn %= 7;
    }
};

#endif
```

```
#ifndef ACTION_HPP
#define ACTION_HPP

#include <iostream>
#include "../game/gameState.hpp"

using namespace std;

class Action {
public:
    Action() {}
    Action (GameState gameState) {
        Action::gameState = gameState;
    }
    static GameState GetGameState() {
        return gameState;
    }
    static void SetGameState(GameState other) {
        Action::gameState = other;
    }
    virtual void DoAction() = 0;

protected:
    static GameState gameState;
};

GameState Action::gameState;

#endif
```

2.2. Method/Operator Overloading

Konsep Operator Overloading digunakan di kelas AbstractValue. Konsep ini digunakan saat ingin mendefinisikan ulang perilaku operator dalam konteks objek yang sedang kita buat. sebagai contoh pada kelas AbstractValue kita ingin membandingkan *value* yang dalam tugas ini adalah nilai kartu. Oleh karena itu, kita mendefinisikan ulang perilaku untuk operator >,<,>=,<=, dan == . Selain itu, operator overloading dapat mempermudah penulisan yang dilakukan.

```
bool operator>(const AbstractValue& other) {
    return this->getValue() > other.getValue();
}
bool operator<(const AbstractValue& other) {
    return this->getValue() < other.getValue();
}
bool operator<=(const AbstractValue& other) {
    return this->getValue() <= other.getValue();
}
bool operator>=(const AbstractValue& other) {
    return this->getValue() >= other.getValue();
}
bool operator==(const AbstractValue& other) {
    return this->getValue() == other.getValue();
}
```

2.3. Template & Generic Classes

Konsep template digunakan pada kelas Game. Dengan adanya template method pada kelas Game, kita tidak perlu mendefinisikan satu per satu tipe untuk mencari player dengan poin terbesar. Pada umumnya penggunaan template ini dapat mempermudah pengguna method tersebut.

```
template <class T>
T getMax(vector<T> listData){
    int len;
    T max = listData.at(0);
    for(int i=1;i<len;i++){
        if(listData.at(i)>max){
            max = listData.at(i);
        }
    }
    return max;
}
```

2.4. Exception

Konsep Exception diperlukan saat menangani kesalahan yang muncul saat eksekusi program. Penanganan kesalahan tersebut membutuhkan kita untuk menambahkan instruksi - instruksi sehingga program menjadi rumit dan rawan adanya *bug*. Oleh karena itu, C++ menyedikan fitur *exception* untuk menangani kesalahan saat *runtime* dengan *throw,try*, dan *catch*. Sintaks *throw* memiliki kegunaan yang sama dengan *return* pada fungsi. Saat sebuah method selesai dengan throw maka dapat dikatakan method tersebut selesai dengan abnormal. Ketika kita ingin mengakses *exception* yang di-throw oleh suatu method, maka method tersebut harus terdapat di dalam block code try dan ditangkap oleh catch. Konsep ini digunakan pada kelas commnadException.

```
class commandException{
    private:
        string commandInputed;
        string abillity;
        bool abilityLess;
    public:
        commandException(string commandInputed, string
abillity, bool abilityLess){
            this->commandInputed = commandInputed;
            this->abillity = abillity;
            this->abilityLess = abilityLess;
        }
        void errorMessage(){
            if(commandInputed != " "){
                cout << commandInputed <<
" tidak ada dalam daftar perintah!!"<< endl;
                cout << "silahkan input lagi!"<<endl;
            }
            else if(abillity != " "){
                if(abilityLess == true){
                    cout << "kartu " << abillity <<
" telah dimatikan"<<endl;
                    cout <<
"silahkan masukkan perintah lainnya!"<<endl;
                }
                else{
                    cout << abillity <<
" tidak dimiliki oleh player!"<< endl;
                    cout << "silahkan input lagi!"<<endl;
                }
            }
        }

};
```

2.5. C++ Standard Template Library

C++ STL cukup banyak digunakan pada setiap kelas. Salah satunya yang paling banyak digunakan pada setiap kelas adalah <string>,<iostream>,dan <fstream>. Library <string> sering dipakai karena kebanyakan menyimpan atribut bertipe string. Sedangkan library <iostream> dan <fstream> untuk input dan output file. Kami juga menggunakan <vector> dan <map> sebagai tipe collection yang mudah digunakan dan diubah ukurannya. Selain itu, kami menggunakan algoritma sorting dari <algorithm>.

```
#include <bits/stdc++.h>

using namespace std;

class Player {
    private:
        string name;
        PlayerCard<Card> playerCard;
        string ability;
        long long int point;
        bool isAbilityLess;
        double value;
```

```
#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP

#include "card.hpp"
#include <vector>
#include <iostream>

using namespace std;


template <class T>
class InventoryHolder {
    protected:
        vector<T> cardList;

    public:
        InventoryHolder();
        InventoryHolder(vector<T>);
        InventoryHolder(const InventoryHolder<T>&);
        ~InventoryHolder();
        void setCard(vector<T>);
        vector<T>& getCard();
        size_t getCardListLength();
        friend ostream& operator<<(ostream&, const
InventoryHolder<T>&);
        virtual void operator+(const T& other) = 0;
        virtual void operator-(const T& other) = 0;
};

#endif
```

2.6. Konsep OOP lain

Konsep OOP yang lain seperti Abstract Base Class, Collection, dan Composition. Konsep Composition digunakan pada kelas Game yang pada atributnya menggunakan kelas GameState. Konsep collection digunakan pada pembuatan kelas DeckCard,PlayerCard,dan TableCard berbentuk vector of objek card. Konsep Abstract Base Class diterapkan pada base Class InventoryHolder kepada kelas PlayerCard, TableCard, dan DeckCard.

```
// class abstrak untuk diturunkan menjadi PlayerCard dan Deck
Card
#ifndef INVENTORYHOLDER_HPP
#define INVENTORYHOLDER_HPP

#include "card.hpp"
#include <vector>
#include <iostream>

using namespace std;

class InventoryHolder {
    protected:
        vector<Card> cardList;

    public:
        InventoryHolder();
        InventoryHolder(vector<Card>);
        InventoryHolder(const InventoryHolder&);
        ~InventoryHolder();
        void setCard(vector<Card>);
        vector<Card>& getCard();
        size_t getCardListLength();
        friend ostream& operator<<(ostream&, const
InventoryHolder&);
        virtual void operator+(const Card& other) = 0;
        virtual void operator-(const Card& other) = 0;
};

#endif
```

```
class Game {
    private:
        GameState gameState;
        int round;
        bool isFinish;
    public:
        Game(){
            this->gameState = GameState();
            this->round = 1;
            this->isFinish = false;
        }
```

3. Teknik Implementasi

3.1 Algoritma Skoring

Algoritma skoring digunakan untuk menentukan skor dari kombinasi kartu yang dipunya oleh player. Kombinasi yang mungkin didapatkan oleh player antara lain high card, pair, two pair, three of a kind, straight, flush, full house, four of a kind , dan straight flush. Masing masing memiliki rumus perhitungannya sendiri. Untuk kasus high card ketika player tidak memiliki kombinasi kartu. untuk memudahkan proses skoring kami menentukan nilai dari setiap warna dan angka pada setiap kombinasi yang ada, salah satu contohnya berikut

(banyak dc	Konstan	Hijau	Biru	Kuning	Merah
1	2,7	2,7	2,73	2,76	2,79
2	2,8	2,8	2,83	2,86	2,89
3	2,9	2,9	2,93	2,96	2,99
4	3	3	3,03	3,06	3,09
5	3,1	3,1	3,13	3,16	3,19
6	3,2	3,2	3,23	3,26	3,29
7	3,3	3,3	3,33	3,36	3,39
8	3,4	3,4	3,43	3,46	3,49
9	3,5	3,5	3,53	3,56	3,59
10	3,6	3,6	3,63	3,66	3,69
11	3,7	3,7	3,73	3,76	3,79
12	3,8	3,8	3,83	3,86	3,89
13	3,9	3,9	3,93	3,96	3,99

Rumus Perhitungan:

1. High Card

   Untuk high card kita akan mencari nilai kartu yang terbesar sehingga implementasi dari kelas high card ini akan mereturn perhitungan dari rumus Konstanta + Warna * 0.03.

```
class highCard :public AbstractValue {
    private:
        Card cards;
    public:
        highCard(Card cards);
        double getValue() const override;
};

#endif
```

   Pencarian nilai kartu terbesar berada pada kelas Player

```
Card getHighCard (){
    Card card = playerCard.getCard()[0];
    for(size_t i = 1; i < playerCard.getCard().size(); i++){
        if(card.getValue() < playerCard.getCard()[i].getValue()){
            card = playerCard.getCard()[i];
        }
    }
    return card;
}
```

2. Pair

   Pair terjadi saat dua kartu memiliki angka yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum high card + 0.01 + 0.1 * (nilai tertinggi pada pair - 1) + 0.03 * warna tertinggi pada pair

3. Two Pair

   Two Pair terjadi saat terdapat 2 pasang pair. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum pair + 0.01 + 0.1 * (angka tertinggi pada two pair - 1) + 0.03 * warna tertinggi pada pair dengan angka tertinggi

4. Three of a Kind

   Three of a Kind terjadi saat tiga kartu memiliki angka yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum two pair + 0.01 + 0.1 * (angka pada kombinasi three of a kind - 1) + 0.03 * warna tertinggi pada kombinasi kartu three of a kind

5. Straight

Straight terjadi saat lima kartu memiliki urutan yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum three of kind + 0.01 + 0.1 * (angka tertinggi pada kombinasi straight - 1) + 0.03 * warna tertinggi pada straight

6. Flush

   Flush terjadi saat lima kartu memiliki warna yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum straight + 0.01 + 0.1 * (nilai tertinggi pada Flush - 1) + 0.03 * warna tertinggi pada Flush

7. Full House

   Full House terjadi saat dua kartu memiliki angka yang sama dan tiga kartu memiliki angka yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum flush + 0.01 + 0.1 * (nilai tertinggi pada kombinasi tiga angka sama - 1) + 0.03 * warna tertinggi pada kombinasi tiga angka sama

8. Four of a Kind

   Four of a Kind terjadi saat empat kartu memiliki angka yang sama. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum full house + 0.01 + 0.1 * (nilai tertinggi pada pair - 1)

9. Straight Flush

   Pair terjadi saat Flush dan Straight. Rumus perhitungan dari kombinasi pair pada kelompok kami yakni nilai maksimum four of a kind + 0.1 + 0.01 * (nilai tertinggi pada pair - 1) + 0.03 * warna tertinggi pada pair

3.2 Manajemen State

Manajemen GameState yang diimplementasikan oleh kelompok kami dengan cara pembuatan class GameState. Class ini akan menjadi attribute untuk beberapa class yang akan merubah state. Ketika sebuah class mengubah state maka class game akan mengambil gameState yang telah diupdate class tersebut.

4. Library dan STL

library dan STL yang digunakan kelompok kami antara lain

1. String

   String merupakan salah satu tipe data yang banyak digunakan seperti mendefinisikan warna, nama player, dan lain-lain

2. Map

   Map ini digunakan untuk memetakan nilai dari setiap warna . Selain itu, Map ini juga digunakan di command agar dapat di switch case.

3. Vector

   Vector digunakan untuk menyimpan kumpulan objek maupun data yang ada. Kami memilih vector untuk menyimpan data karena vektor lebih fleksibel daripada array seperti memperbesar ukuran elemen secara dinamis dan memberikan kemudahan untuk memanipulasi elemen. Vektor digunakan untuk menyimpan objek kartu,player,dan lain-lain.

4. Algorithm

   Kami menggunakan library Algorithm untuk menggunakan algoritma sorting untuk menyorting kumpulan kartu yang dimiliki player.

5. Bonus Yang dikerjakan

5.1. Bonus yang diusulkan oleh spek

5.1.1. Game Kartu Lain

Bonus Game Kartu lain kami mengambil game kartu black jack. Blackjack adalah permainan kartu yang dimainkan dengan satu atau lebih dek kartu standar 52 kartu. Tujuannya adalah untuk mengalahkan dealer dengan memiliki kartu bernilai lebih tinggi daripada kartu dealer, tetapi tanpa melebihi nilai total 21.Setiap kartu dalam blackjack memiliki nilai yang berbeda. Kartu As dapat bernilai 1 atau 11, sedangkan kartu King, Queen, dan Jack bernilai 10. Sisa kartu bernilai sesuai dengan angka yang tertera pada kartu, yaitu 2-10. Pada awal permainan, dealer dan pemain akan diberi dua kartu masing-masing. Pemain kemudian memutuskan apakah akan menambah kartu lagi (hit) atau tidak (stand) dengan tujuan untuk mendapatkan total nilai kartu yang lebih tinggi. Jika nilai kartu pemain melebihi 21 maka pemain burst (kalah).

Berikut implementasi dari kelas GameStateB dan GameBj:

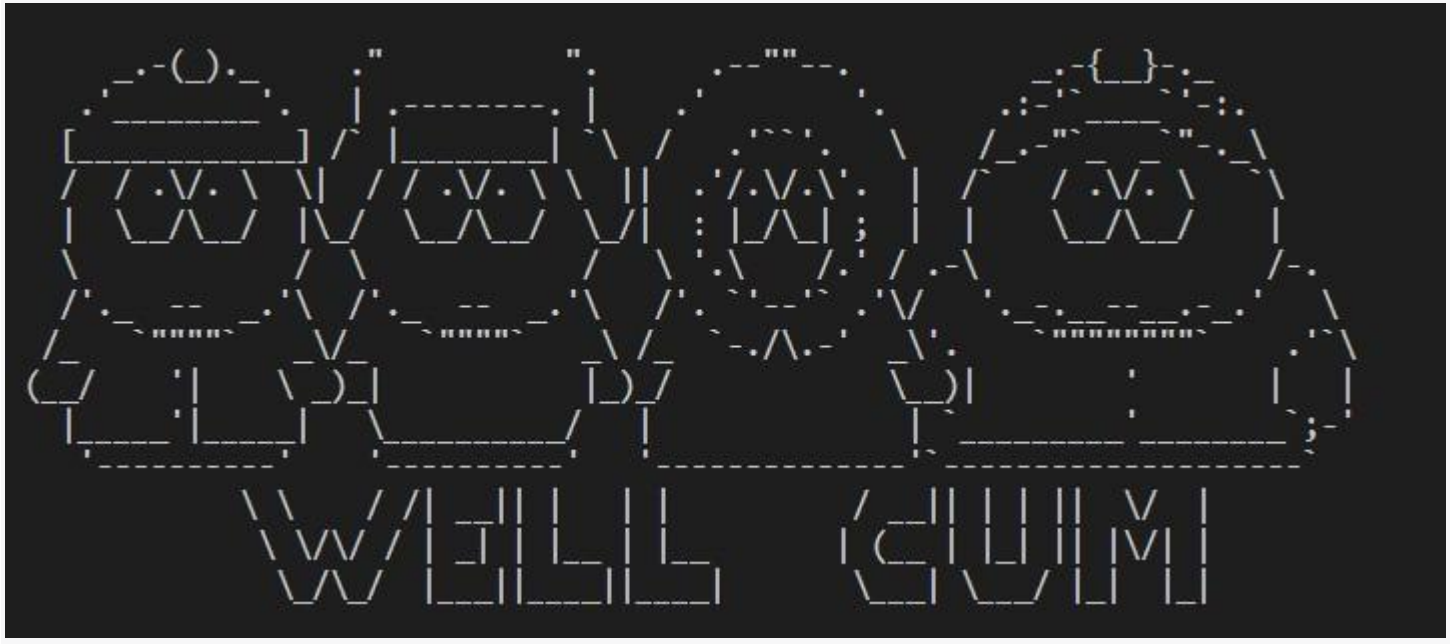
```
#include "../player/player.hpp"
#include "../card/deckCard.hpp"
#include "../card/playerCard.hpp"
#include <iostream>
#include <time.h>
using namespace std;


class BlackJack {
 public :
   BlackJack(){
       this->bandar = Player();
       this->player2 = Player();
       vector<Card> nama;
       string nama1[] = {"Wajik","Keriting","Hati","Sekop"};
       for(int i = 0;i<4;i++){
           for(int j = 1 ; j <= 13 ; j++){
               if(j == 11 || j == 12 || j == 13){
                   nama.push_back(Card(j,nama1[i],10));
               }else{
                   nama.push_back(Card(j,nama1[i],j));
               }
           }
       }
       this->deckCard = DeckCard(nama);
       bandar.setName("Bandar");
       status = true;
       win = false;
   }
```

```
#include "GameStateBj.hpp"


class GameBJ{
   private:
       BlackJack state;
       bool status;
       int point;
       string name;

   public:
       GameBJ(){
           status = true;
           point = 1000;
           cout << "Masukan Namamu : " ;
           cin >> name;
           state.getPlayerBj().setName(name);
           state.getPlayerBj().setPoint(point);
           cout << "Poin awal mu: " << point << endl;
       }
```

5.2. Bonus Kreasi Mandiri

3.2.1 Splash Screen

Splash screen adalah tampilan awal yang ditampilkan saat sebuah aplikasi atau program dimulai. Splash screen berupa gambar dan informasi singkat.



3.2.2 Ascii Art

Ascii art untuk setiap kartu yang ditampilkan.

6. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
game	13521065	13521065,13521080,13521099,13521122,13521146
ability	13521080,13521065,13521122	13521065,13521080,13521099,13521122,13521146
command	13521122	13521065,13521080,13521099,13521122,13521146
bonus	13521080	13521065,13521080,13521099,13521122,13521146
card	13521146	13521065,13521080,13521099,13521122,13521146
card (kombo)	13521099	13521065,13521080,13521099,13521122,13521146
player	13521122	13521065,13521080,13521099,13521122,13521146

7. Foto Kelompok