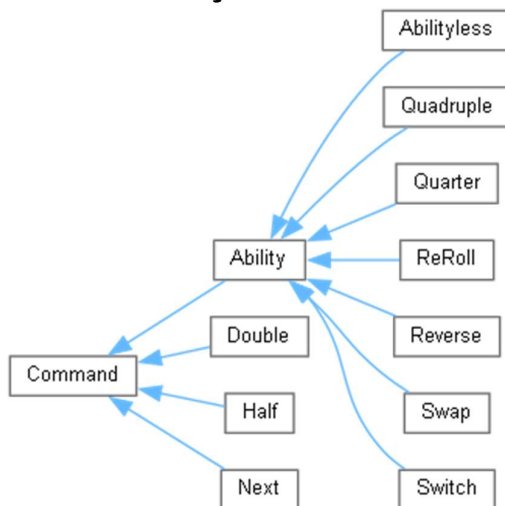


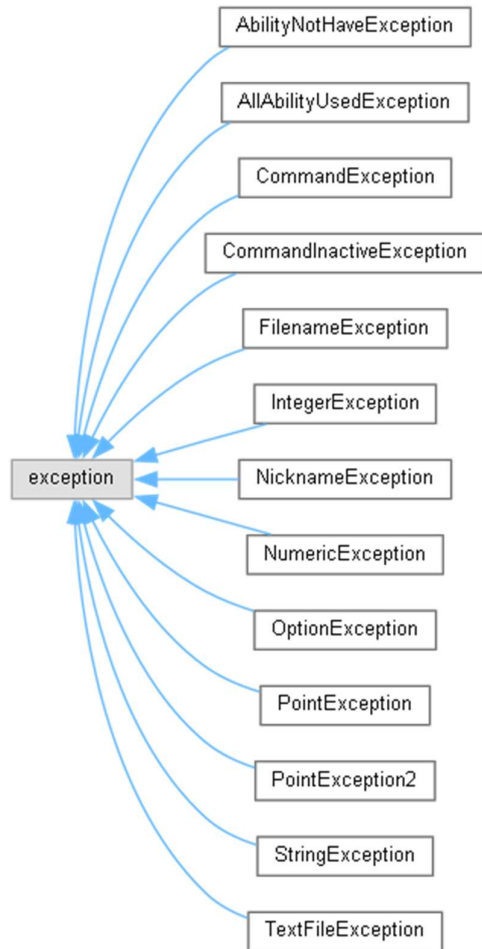
Kode Kelompok : CBR
Nama Kelompok : masbro
1. 13521044 / Rachel Gabriela Chen
2. 13521046 / Jeffrey Chow
3. 13521074 / Eugene Yap Jin Quan
4. 13521094 / Angela Livia Arumsari
5. 13521100 / Alexander Jason
Asisten Pembimbing : Steven Nataniel Kodyat

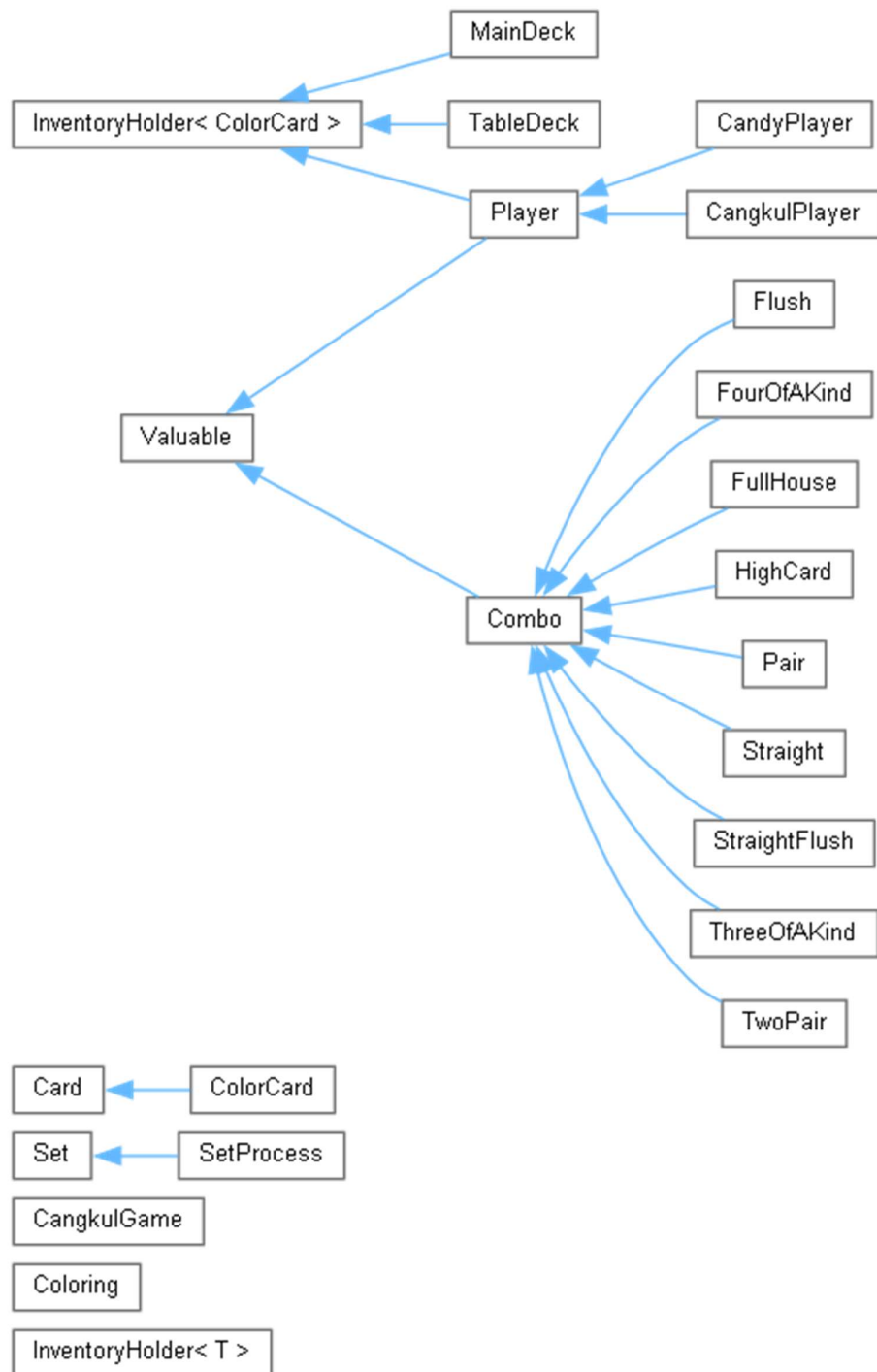


1. Diagram Kelas

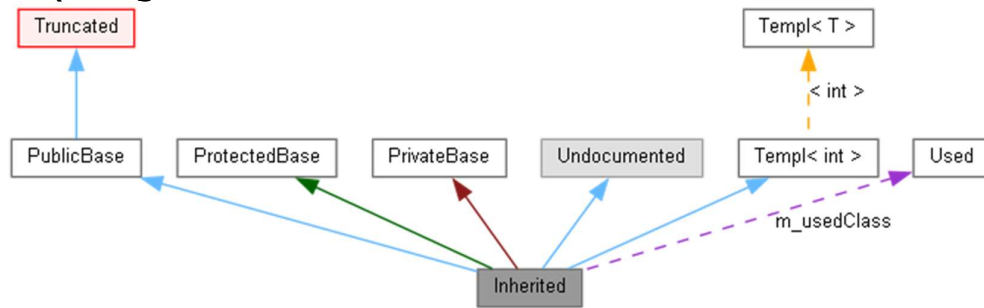
a. Class Hierarchy







b. Class Diagram Graph Legend

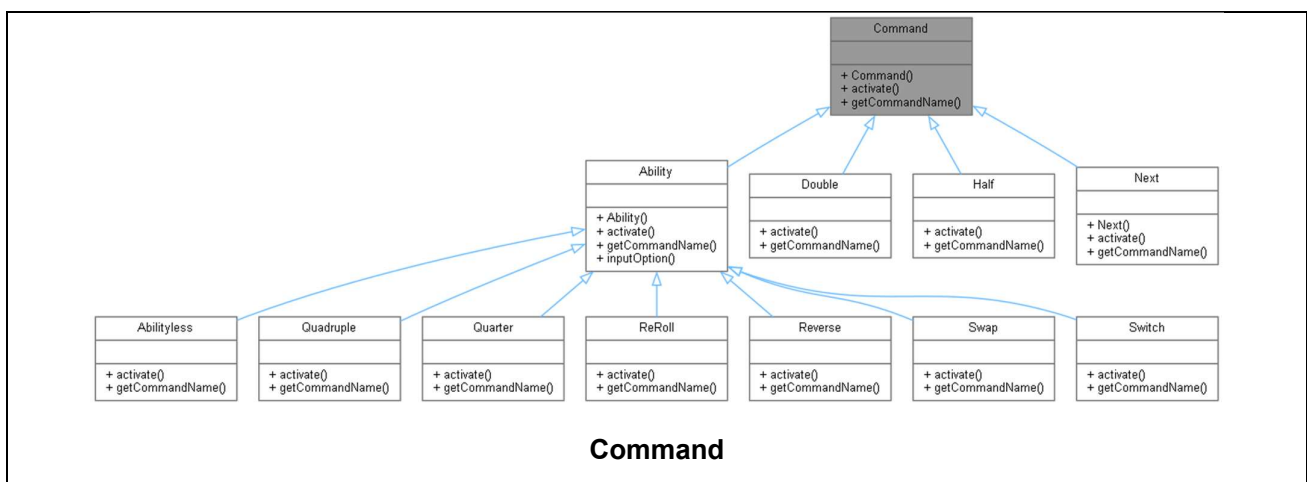


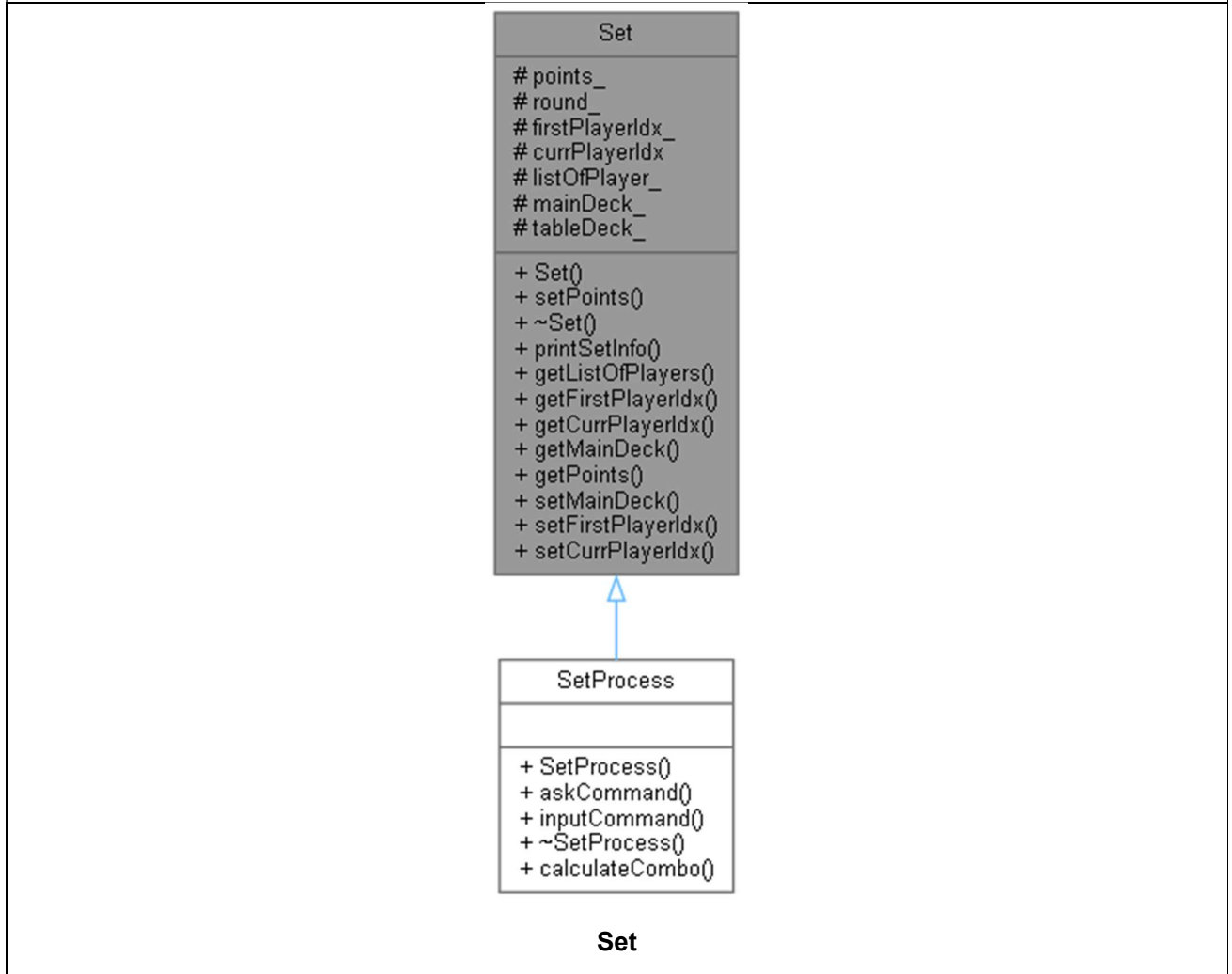
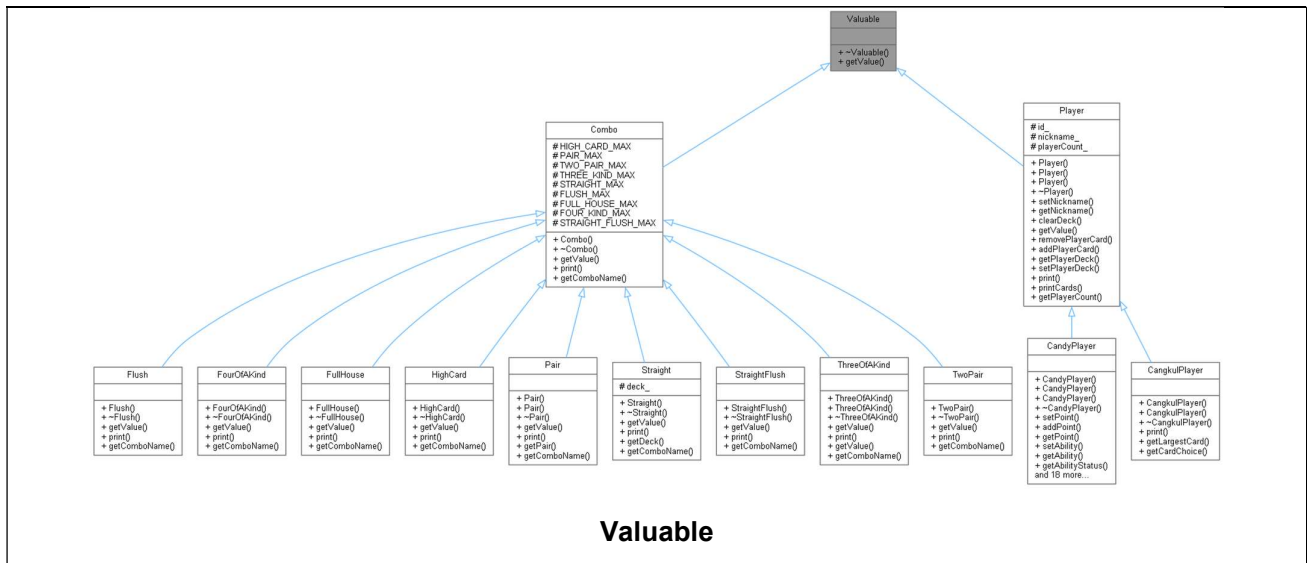
The boxes in the above graph have the following meaning:

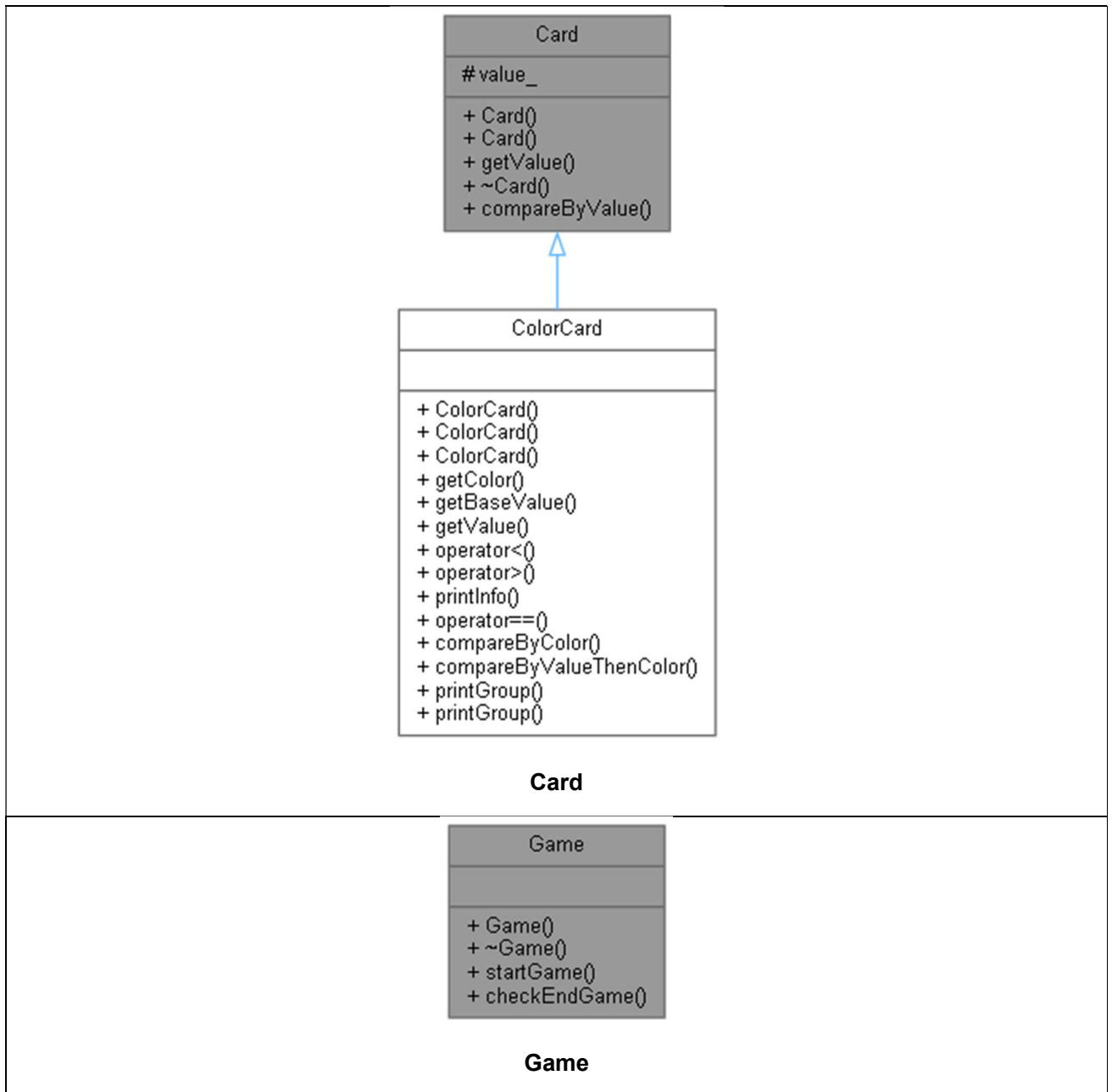
- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a gray border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class for which not all inheritance/containment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

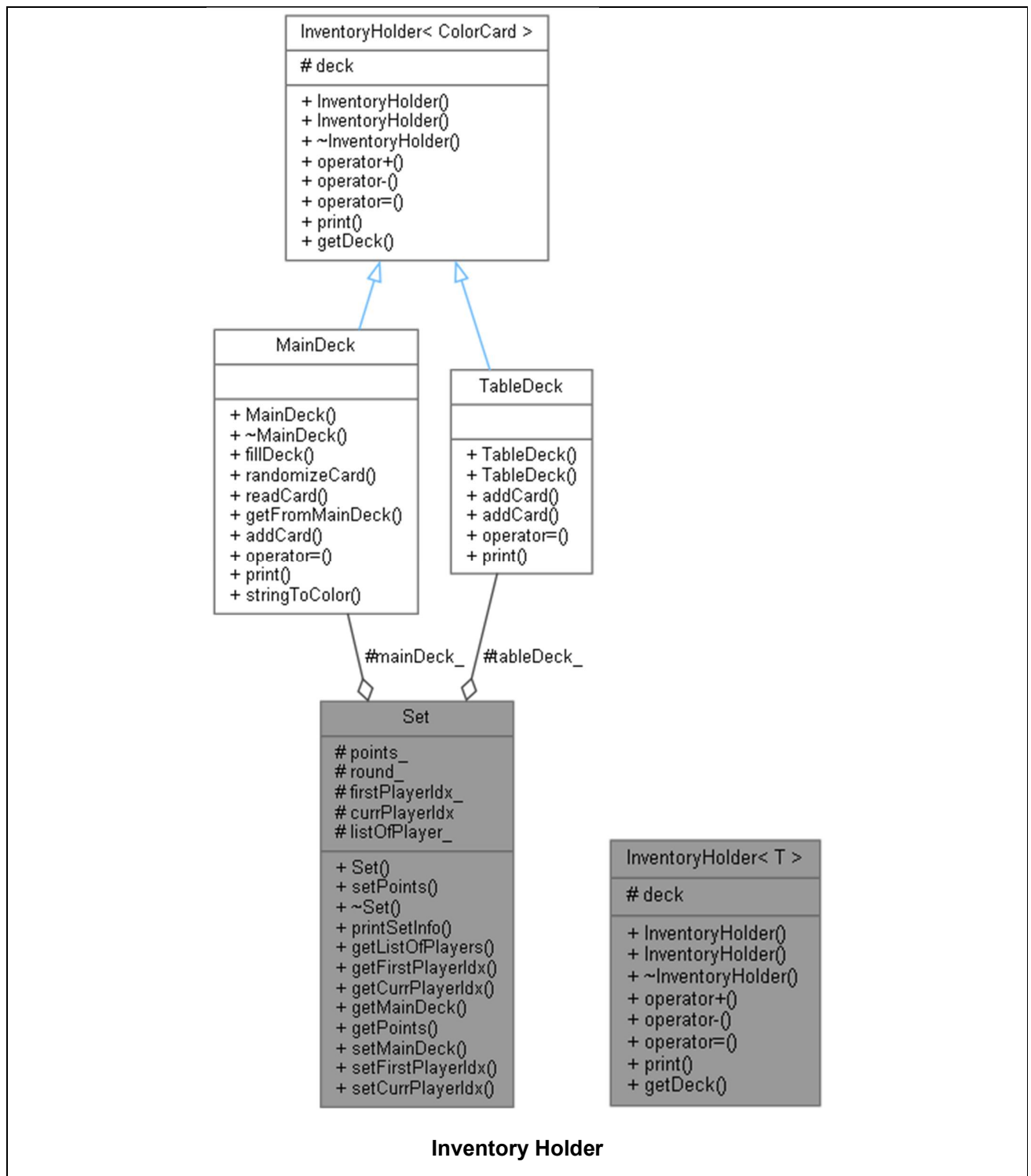
The arrows have the following meaning:

- 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.



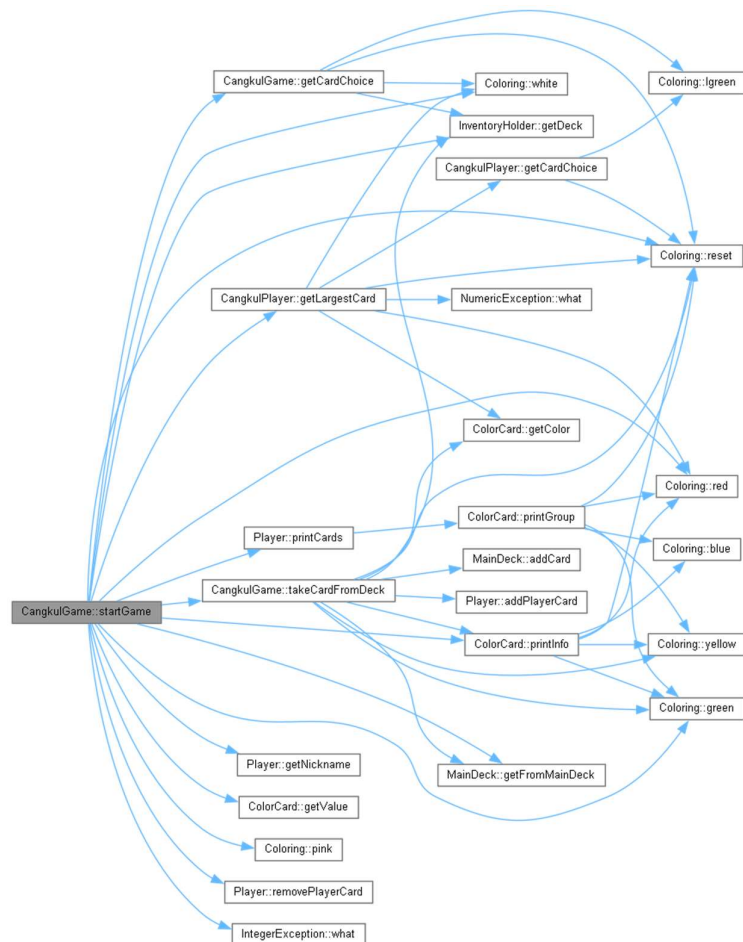








Coloring



CangkulGame (BONUS)

c. Class Design

1) **Command**

Kelas Command menerapkan konsep *inheritance* dan *polymorphism*. *Inheritance* digunakan untuk mengimplementasikan *command* yang lebih spesifik dan valid seperti *Half*, *Double*, *Next*, dan berbagai macam *Ability* lain. *Polymorphism* digunakan untuk menggeneralisasi tipe objek pada kelas anak yang diinstansiasi pada kelas lainnya. Kelebihan dari desain kelas ini adalah kita dapat dengan mudah membuat tipe *collection of command* sehingga kita tidak perlu membuat pemrograman secara procedural.

2) **Ability**

Sama seperti Command, kelas Ability juga menerapkan konsep *inheritance* dan *polymorphism*. Kelebihan dari desain kelas ini dapat membuat sebuah *collection of ability* yang valid.

3) **Exception**

Kelas Exception pada dasarnya juga menerapkan konsep yang sama dengan kelas Ability dan Command. Kelas ini menerapkan *polymorphism* pada *exception handling* dimana kita harus melakukan *catch exception* berdasarkan jenisnya. Akan lebih mudah dan sederhana apabila kita hanya perlu menangkap *exception* pada kelas dasarnya.

4) **InventoryHolder**

Kelas InventoryHolder menampung TableDeck, MainDeck, serta kelas Player sebagai objek utama di dalam permainan ini. Kelebihan dari kelas ini adalah *reusability* dimana *deck* dan *player* dapat dimanfaatkan pada permainan bonus.

5) **Player**

Kelas Player menerapkan *Inheritance* dan *Polymorphism*. Kelas ini diturunkan lagi menjadi CandyPlayer dan CangkulPlayer untuk 2 permainan yang berbeda namun sifat dari kelasnya tetap sama.

6) **Valuable**

Valuable menyimpan hasil *value* dari setiap kelas dan turunannya. Kelas ini menerapkan konsep *inheritance*, *polymorphism*, *abstraction*, dan STL.

7) **Combo**

Kelas ini menyimpan semua perhitungan dari kombinasi kartu yang ada dan memanfaatkan konsep *Abstraction* dan *Inheritance*.

8) **Card**

Kelas Card menyimpan *value* dan warna dari setiap 52 kartu yang digunakan pada permainan ini.

9) **Set**

Kelas Set berguna untuk memproses aturan, *command*, dan alur setiap set.

10) **Coloring**

Kelas ini berfungsi sebagai tampilan interface di CLI.

11) **CangkulGame**

Kelas ini merupakan kelas game bonus

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep *Inheritance* dan *Polymorphism* kami gunakan pada beberapa kelas di program kami. Kelas yang menggunakan *Inheritance* dan *Polymorphism* adalah Ability (*Parent*), yang memiliki jenis-jenis ability sebagai *child*-nya; Command (*Parent*), yang memiliki *command* yang *valid* sebagai *child*-nya; Inventory Holder (*Parent*) yang memiliki main, table, dan player

sebagai *child*-nya; Set (*Parent*) yang memiliki pemrosesan set tiap gamenya sebagai *child*; serta Valuable (*Parent*) yang memiliki card, player, dan combo sebagai *child*.

Kami menerapkan *Polymorphism* dengan tujuan agar *child* dapat diperlakukan sebagai kelas yang sama dengan *parent* nya. Kami juga menerapkan *Inheritance* dengan tujuan agar *child* dapat memiliki sifat yang sama dengan *parent* nya.

Class **Command**
sebagai Parent dan
Ability, Half, Next,
Double sebagai
child

```
1 class Command
2 {
3 public:
4     Command();
5     virtual void activate(Set&) = 0;
6
7     virtual string getCommandName() = 0;
8 };
```

```
1 class Half : public Command
2 {
3 public:
4     void activate(Set &set);
5     string getCommandName();
6 };
```

```
1 class Double : public Command
2 {
3 public:
4     void activate(Set &set);
5     string getCommandName();
6 };
```

```
1 class Next : public Command{
2 public:
3     Next();
4     void activate(Set &set);
5     string getCommandName();
6 };
```

```
1 class Ability : public Command{
2 public:
3     /**
4      * @brief Construct a new Ability object
5      *
6      */
7     Ability();
8
9     /**
10    * @brief Activate Ability
11    *
12    */
13    virtual void activate(Set &) = 0;
14
15    virtual string getCommandName() = 0;
16
17    int inputOption(int);
18 };
19
```

Polymorphism
yang memanggil
command yang
sudah diturunkan

```

1  string command = inputCommand(allowedCommands, currPlayer);
2      for (auto &c : commands_)
3      {
4          if (c->getCommandName() == command)
5          {
6              try{
7                  c->activate(*this);
8              }
9              catch(PointException &e){
10                 cout << e.what();
11             }
12             catch(PointException2 &e){
13                 cout << e.what();
14             }
15             commandSet = true;
16             break;
17         }
18     }

```

```

1  string Next::getCommandName()
2  {
3      return "NEXT";
4  }

```

```

1  string Half::getCommandName()
2  {
3      return "HALF";
4  }

```

```

1  string Double::getCommandName()
2  {
3      return "DOUBLE";
4  }

```

```

1  string Swap::getCommandName()
2  {
3      return "SWAP";
4  }

```

2.2. Method/Operator Overloading

Konsep *Method Overloading* digunakan di kelas TableDeck. Sedangkan *Operator Overloading* digunakan pada beberapa kelas yaitu InventoryHolder dan ColorCard. *Method overloading* diimplementasikan untuk fungsi yang memiliki nama yang sama namun mempunyai tipe parameter yang berbeda. Sebagai contoh, pada class TableDeck terdapat fungsi addCard yang memiliki 2 jenis parameter berbeda. Fungsi addCard dengan command MainDeck berfungsi untuk mengurangi kartu pada MainDeck dan menambahkannya pada TableDeck. Sedangkan fungsi dengan parameter ColorCard hanya menambahkan ColorCard pada parameter langsung pada TableDeck. Alur logika penggunaan pada kedua fungsi tersebut sama sehingga dapat dioverload dengan parameter yang berbeda. *Operator overloading* digunakan untuk mempermudah penulisan. Sebagai contoh, pada class ColorCard untuk membandingkan nilai 2 objek ColorCard harus mengambil atribut pada masing-masing objek. Namun dengan operator overloading, penulisan dapat menjadi sederhana dengan hanya menggunakan simbol '<'.

```

1  template <class T>
2  InventoryHolder<T> &InventoryHolder<T>::operator+(const T &toAdd)
3  {
4      this->deck.push_back(toAdd);
5      return *this;
6  }
7
8  template <class T>
9  InventoryHolder<T> &InventoryHolder<T>::operator-(const T &toRemove)
10 {
11     auto idxRemove = find(this->deck.begin(), this->deck.end(), toRe
move);
12     if (idxRemove != this->deck.end())
13     {
14         this->deck.erase(idxRemove);
15     }
16     return *this;
17 }
18
19 template <class T>
20 InventoryHolder<T> &InventoryHolder<T>::operator=(const InventoryHol
der &other)
21 {
22     if (this != &other)
23     {
24         this->deck.clear();
25         this->deck = other.deck;
26     }
27     return *this;
28 }

```

```

1  bool ColorCard::operator<(const ColorCard &other) const
2  {
3      return this->getValue() < other.getValue();
4  }
5
6  bool ColorCard::operator>(const ColorCard &other) const
7  {
8      return this->getValue() > other.getValue();
9  }
10
11 bool ColorCard::operator==(const ColorCard &other) const
12 {
13     if (this->getColor() == other.getColor() && this->getValue() == other.getValue())
14     {
15         return true;
16     }
17     return false;
18 }

```

2.3. Template & Generic Classes

Generic class digunakan pada kelas InventoryHolder. InventoryHolder memiliki satu atribut yaitu deck yang berupa vector dari sebuah class T. Hal ini dipilih untuk memudahkan penggantian tipe data pada vector deck.

```

1  template <class T>
2  class InventoryHolder
3  {
4  protected:
5      vector<T> deck;
6
7  public:
8      /**
9       * @brief Construct a new Inventory Holder object
10      */
11      InventoryHolder();
12
13      /**
14       * @brief Copy Construct a new Inventory Holder object
15       * @param other
16       */
17      InventoryHolder(const InventoryHolder &other);
18
19      /**
20       * @brief Destroy the Inventory Holder object
21       */
22      virtual ~InventoryHolder();
23
24
25
26

```

```

27
28      /**
29       * @brief Add card to InventoryHolder
30       * @param toAdd
31       * @return InventoryHolder&
32       */
33      InventoryHolder &operator+(const T &toAdd);
34
35      /**
36       * @brief Remove card to InventoryHolder
37       * @param toRemove
38       * @return InventoryHolder&
39       */
40      InventoryHolder &operator-(const T &toRemove);
41
42      InventoryHolder &operator=(const InventoryHolder &other);
43
44      /**
45       * @brief print Inventory
46       */
47      virtual void print() = 0;
48
49      vector<T> &getDeck();
50
51
52  };

```

2.4. Exception

Pada program ini, kami mengimplementasikan Exception tambahan. Exception tambahan ini diturunkan dari kelas exception yang tersedia di C++. Berikut adalah cuplikan kode header Exception yang kami terapkan.

```

1  #ifndef _EXCEPTION_HPP_
2  #define _EXCEPTION_HPP_
3
4  #include <bits/stdc++.h>
5  using namespace std;
6
7  struct NumericException : public exception
8  {
9      const char *what() const throw()
10     {
11         return "Numeric input not valid.\nPlease input only integers.";
12     }
13 };
14
15 struct TextFileException : public exception
16 {
17     const char *what() const throw()
18     {
19         return "File configuration is invalid. Make sure the configuration format is [Color] [Number] and use the valid Color or Number\n";
20     }
21 };
22
23 struct CommandException : public exception
24 {
25     const char *what() const throw()
26     {
27         return "Command input not valid.\nPlease input the right command";
28     }
29 };
30

```

```

31 struct CommandInactiveException : public exception
32 {
33     const char *what() const throw()
34     {
35         return "Ability was deactivated by another player.\n";
36     }
37 };
38
39 struct StringException : public exception
40 {
41     const char *what() const throw()
42     {
43         return "Input is invalid. Try input a valid string\n";
44     }
45 };
46
47 struct IntegerException : public exception
48 {
49     const char *what() const throw()
50     {
51         return "Input is invalid. Try input a valid integer\n";
52     }
53 };
54
55 struct NicknameException : public exception
56 {
57     const char *what() const throw()
58     {
59         return "No player with that nickname is found.\n";
60     }
61 };

```

```

62
63 struct PointException : public exception
64 {
65     const char *what() const throw()
66     {
67         return "The set point is already 1 so it cant be changed.\n";
68     }
69 };
70
71 struct FilenameException : public exception{
72     const char *what() const throw(){
73         return "File not found. Please input the correct file and follow this format [filename].txt\n";
74     }
75 };
76
77 struct OptionException : public exception{
78     const char *what() const throw(){
79         return "Option not valid. Please input valid option.\n";
80     }
81 };
82
83 #endif

```

Penjelasan Exception tambahan yang kami diimplementasikan dirinci pada tabel di bawah.

No.	Exception	Keterangan
1.	<i>NumericException</i>	Exception digunakan untuk validasi input integer.

2.	<i>TextFileException</i>	Exception dilemparkan jika format file konfigurasi permainan salah.
3.	<i>CommandException</i>	Exception dilemparkan jika perintah dalam permainan salah.
4.	<i>CommandInactiveException</i>	Exception dilempar jika pemain memanggil perintah yang sudah dinonaktifkan oleh pemain lain.
5.	<i>StringException</i>	Exception digunakan untuk validasi input string.
6.	<i>IntegerException</i>	Exception digunakan untuk validasi input integer.
7.	<i>NicknameException</i>	Exception digunakan jika dilakukan pencarian terhadap nama yang tidak ada.
8.	<i>PointException</i>	Exception digunakan jika set point sudah 1 dan masih ingin diubah.
9.	<i>PointException2</i>	Exception ini digunakan jika set point bernilai 2 dan dilakukan QUARTER.
10.	<i>FilenameException</i>	Exception digunakan ketika file yang dibaca tidak ditemukan.
11.	<i>OptionException</i>	Exception digunakan jika pengguna memilih opsi yang salah.

2.5. C++ Standard Template Library

Implementasi program kami menggunakan beberapa C++ Standard Template Library (STL), seperti vector, pair, dan map. Berikut adalah tabel yang menjelaskan penggunaan STL pada program kami.

Penggunaan STL vector	
Cuplikan Kode	Keterangan
	Contoh penggunaannya adalah pada kelompok kelas Combo. Pada beberapa kelas turunan dari Combo, vector digunakan untuk mengirimkan argumen objek-objek ColorCard. Dengan menggunakan vector, argumen tersebut dapat dengan mudah diurutkan atau dikelompokkan.

```
1 #include "FullHouse.hpp"
2 using namespace std;
3
4 FullHouse::FullHouse(vector<ColorCard> deck){
5     vector<ColorCard> temp = deck;
6     sort(temp.begin(), temp.end(), ColorCard::compareByValue);
7     /*
8     1 1 2 2 2
9     1 1 1 2 2
10    */
11    if(temp[1].getValue() == temp[2].getValue()){
12        Pair pair(make_pair(temp[3], temp[4]));
13        this->pair_ = pair;
14        vector<ColorCard> newTris;
15        newTris.push_back(temp[0]);
16        newTris.push_back(temp[1]);
17        newTris.push_back(temp[2]);
18        ThreeOfAKind tris(newTris);
19        this->tris_ = tris;
20    }else{
21        Pair pair(make_pair(temp[0], temp[1]));
22        this->pair_ = pair;
23        vector<ColorCard> newTris;
24        newTris.push_back(temp[2]);
25        newTris.push_back(temp[3]);
26        newTris.push_back(temp[4]);
27        ThreeOfAKind tris(newTris);
28        this->tris_ = tris;
29    }
30 }
31
32 FullHouse::~FullHouse(){
33 }
34
35
36 float FullHouse::getValue() const {
37     /* rumus : concat(encode(three), encode(pair)) * maxflush */
38     float res = FLUSH_MAX; /* maximum flush value */
39
40     res += tris_.getValue() + 0.00001 * pair_.getValue();
41
42     return res;
43 }
44
45 void FullHouse::print()
46 {
47     pair_.print();
48     tris_.print();
49 }
50
51 string FullHouse::getComboName() const{
52     return "FULL HOUSE";
53 }
```

Gambar: Implementasi Combo-FullHouse


```

1  #ifndef _INVENTORY_HOLDER_HPP_
2  #define _INVENTORY_HOLDER_HPP_
3
4  #include <vector>
5  #include <iterator>
6  #include <algorithm>
7  #include "../Valuable/Card/Card.hpp"
8  #include "../Valuable/Card/ColorCard/ColorCard.hpp"
9
10 using namespace std;
11
12 template <class T>
13 class InventoryHolder
14 {
15 protected:
16     vector<T> deck;
17
18 public:
19     /**
20      * @brief Construct a new Inventory Holder object
21      *
22      */
23     InventoryHolder();
24
25     /**
26      * @brief Copy Construct a new Inventory Holder object
27      *
28      * @param other
29      */
30     InventoryHolder(const InventoryHolder &other);
31
32     /**
33      * @brief Destroy the Inventory Holder object
34      *
35      */
36     virtual ~InventoryHolder();
37
38     /**
39      * @brief Add card to InventoryHolder
40      *
41      * @param toAdd
42      * @return InventoryHolder&
43      */
44     InventoryHolder &operator+(const T &toAdd);
45
46     /**
47      * @brief Remove card to InventoryHolder
48      *
49      * @param toRemove
50      * @return InventoryHolder&
51      */
52     InventoryHolder &operator-(const T &toRemove);
53
54     InventoryHolder &operator=(const InventoryHolder &other);
55
56     /**
57      * @brief print Inventory
58      *
59      */
60     virtual void print() = 0;
61
62     vector<T>& getDeck();
63 };
64
65 #endif

```

Gambar: Definisi Kelas InventoryHolder

STL vector juga digunakan dalam implementasi kelas InventoryHolder. Kelas ini merupakan spesialisasi penggunaan vector untuk menyimpan objek. Kelas ini diturunkan menjadi MainDeck dan PlayerDeck yang digunakan untuk menyimpan kartu. Penggunaan STL vector pada kasus ini mempermudah kami dalam manajemen memori.

```

1  #ifndef _MAIN_DECK_HPP_
2  #define _MAIN_DECK_HPP_
3
4  #include "../InventoryHolder.hpp"
5  #include "../Exception/Exception.hpp"
6  #include "../Valuable/Card/ColorCard/ColorCard.hpp"
7  #include "../Util/Coloring.hpp"
8
9  #include <limits>
10 #include <random>
11 #include <fstream>
12 #include <sstream>
13 #include <ctime>
14
15 using namespace std;
16
17 class MainDeck : public InventoryHolder<ColorCard>
18 {
19 private:
20 public:
21     /**
22      * @brief Construct a new Main Deck object
23      *
24      */
25     MainDeck();
26     ~MainDeck();
27     void fillDeck();
28     void randomizeCard();
29     void readCard();
30     ColorCard getFromMainDeck();
31     void addCard(ColorCard card);
32     MainDeck &
33     operator=(const MainDeck &other);
34     void print();
35     Color stringToColor(const string &color);
36 };
37
38 #endif

```

Gambar: Definisi Kelas MainDeck

```

1  #ifndef _CANDY_GAME_HPP_
2  #define _CANDY_GAME_HPP_
3
4  #include "../Valuable/Player/CandyPlayer.hpp"
5  #include "../Exception/Exception.hpp"
6  #include "../Util/Coloring.hpp"
7  #include <unistd.h>
8  #include <iomanip>
9
10 using namespace std;
11
12 class CandyGame
13 {
14 private:
15     vector<CandyPlayer> listOfPlayer;
16
17 public:
18     CandyGame();
19     ~CandyGame();
20     void startGame();
21     bool checkEndGame();
22 };
23
24 #endif

```

Gambar: Definisi Kelas CandyGame

STL vector digunakan juga dalam implementasi Game untuk menyimpan daftar pemain. Vector juga digunakan untuk menyimpan daftar kartu yang telah dibuang pada permainan cangkul.

```

1 #ifndef _CANGKULGAME_HPP
2 #define _CANGKULGAME_HPP_
3
4 #include <iostream>
5 #include <unistd.h>
6 #include "../Valuable/Player/CangkulPlayer.hpp"
7 #include "../InventoryHolder/MainDeck/MainDeck.hpp"
8 #include "../Util/Coloring.hpp"
9
10 using namespace std;
11
12 class CangkulGame
13 {
14 private:
15     vector<CangkulPlayer> listOfPlayers_;
16     MainDeck mainDeck_;
17     vector<ColorCard> thrownCards_;
18
19 public:
20     CangkulGame();
21     ~CangkulGame();
22     void startGame();
23     int getCardChoice(CangkulPlayer &roundWinner);
24     ColorCard *takeCardFromDeck(CangkulPlayer &p, ColorCard tableCard);
25 };
26
27 #endif

```

Gambar: Definisi Kelas CangkulGame

```

1 void ColorCard::printGroup(vector<ColorCard> cc)
2 {
3     Coloring clr;
4     for (int i = 0; i < 6; i++)
5     {
6         for (auto &c : cc)
7         {
8             // apply a color
9             switch (c.color_)
10             {
11                 case Color::Red:
12                     clr.red(true);
13                     break;
14                 case Color::Green:
15                     clr.green(true);
16                     break;
17                 case Color::Blue:
18                     clr.blue(true);
19                     break;
20                 case Color::Yellow:
21                     clr.yellow(true);
22                     break;
23             }
24
25             // which line
26             if (i == 0)
27             {
28                 cout << " _____" << " ";
29             }
30             else if (i == 1)
31             {
32                 cout << "|" << to_string(c.value_) << (c.value_ < 10 ? " " : "") << " |";
33             }
34             else if (i == 5)
35             {
36                 cout << " _____" << (c.value_ < 10 ? " " : "") << to_string(c.value_) << " |";
37             }
38             else
39             {
40                 cout << "|" << " |";
41             }
42
43             // padding right
44             cout << " ";
45             clr.reset();
46         }
47         cout << endl;
48     }
49 }
50

```

Gambar: Implementasi Method printGroup pada kelas ColorCard

STL vector memudahkan kami dalam implementasi pencetakan sekelompok kartu dalam satu baris. Pada kasus ini, STL vector digunakan untuk pengiriman koleksi kartu kepada method printGroup. Karena koleksi menggunakan STL vector, pencetakan jumlah kartu dalam satu baris dapat dilakukan dalam jumlah banyak.

```

1  #include "Reverse.hpp"
2
3  void Reverse::activate(Set &set)
4  {
5      vector<CandyPlayer> &listOfPlayers = set.getListOfPlayers();
6      vector<CandyPlayer> temp;
7      int currPlayerIdx = set.getCurrPlayerIdx();
8      for (int i = currPlayerIdx+1; i<listOfPlayers.size(); i++) {
9          temp.push_back(listOfPlayers[i]);
10     }
11     for(int i = 0; i < currPlayerIdx+1; i++) {
12         temp.push_back(listOfPlayers[i]);
13     }
14     listOfPlayers = temp;
15     reverse(listOfPlayers.begin(), listOfPlayers.end());
16     set.setCurrPlayerIdx(0);
17     set.setFirstPlayerIdx(0);
18     cout << "Successfully reversed the players turns." << endl;
19 }
20
21 string Reverse::getCommandName()
22 {
23     return "REVERSE";
24 }

```

Gambar: Implementasi Kelas Reverse

```

1 //Source "Swag.hgs"
2
3 void Swag::activate(Get asset)
4 {
5     Calculating slot;
6     deckIndex = PlayerId + 1; firstPlayer = getSlot(deckPlayerId);
7     lastCardPlayerId = getSlot(deckPlayerId+1);
8
9     ptr.units(0x0e);
10    cout << ListOfPlayer[currLayerId].getColumnName();
11    ptr.reset();
12
13    const w <- 1; //using NBP2000 let's
14    cout << "Please choose a player to swap their card: \n";
15    int n = 1;
16    for (int i = 0; i < ListOfPlayer.size(); i++)
17    {
18        if (ListOfPlayer[i])
19            continue;
20        else
21        {
22            cout << "[ " << n << " - " << i << " ] ";
23            ListOfPlayer[i].print();
24            n++;
25        }
26    }
27
28    bool success = false;
29    while (!success) {
30        try
31        {
32            int opt = InputGetInt(0);
33            int defFirstPlayer = opt < currLayerId ? opt - 1 : opt;
34            cout << "Please choose a player to swap their card: \n";
35            n = 1;
36            for (int i = 0; i < ListOfPlayer.size(); i++)
37            {
38                if (i == currLayerId || i == (defFirstPlayer))
39                    continue;
40                else
41                {
42                    cout << "[ " << n << " - " << i << " ] ";
43                    ListOfPlayer[i].print();
44                    n++;
45                }
46            }
47
48            opt = InputGetInt(0);
49            int defMax = max(InputGetValue, currLayerId);
50            int defMin = min(InputGetValue, currLayerId);
51            int defSwapPlayer = opt < defMin ? defMin : opt > defMax ? defMax : opt - 1 : opt;
52
53            cout << "Please choose right/left for " << ListOfPlayer[(defFirstPlayer).getColumnName()] << "'s cards: \n";
54            cout << "[ 1 ] Right [ 2 ] Left \n";
55
56            opt = InputGetInt(0);
57
58            int defSecondPlayer = 2 : opt;
59            cout << "Please choose a left/right for " << ListOfPlayer[(defSecondPlayer).getColumnName()] << "'s cards: \n";
60            cout << "[ 1 ] Right [ 2 ] Left \n";
61
62            opt = InputGetInt(0);
63
64            ListOfPlayers[currLayer - 2] opt;
65
66            CardDeck tempDeck = ListOfPlayer[(defFirstPlayer).getPlayerStock()][defCurrFirstPlayer];
67            CardCard tempCard = ListOfPlayer[(defSecondPlayer).getPlayerStock()][defCardDeckPlayer];
68            map<slot, ListOfPlayer[(defFirstPlayer).removePlayerCard(tempFirst)];
69            set<slot, ListOfPlayer[(defFirstPlayer).swapPlayerCard(tempSecond)];
70            set<slot, ListOfPlayer[(defSecondPlayer).removePlayerCard(tempSecond)];
71            set<slot, ListOfPlayer[(defSecondPlayer).addPlayerCard(tempFirst)];
72
73            success = true;
74        }
75        catch (ChangeException ex)
76        {
77            clrared();
78            cout << ex.what();
79            ptr.reset();
80            cin.clear();
81            cin.ignore(numeric_limits<streamsize_t>().max(), '\n');
82        }
83        cout << ChangeException ex;
84        {
85            clrared();
86            cout << ex.what();
87            ptr.reset();
88        }
89    }
90 }
91
92 //using Range Get Column Name()
93 return "CDSP";

```

Gambar: Implementasi Kelas Swap

Kami menggunakan STL vector dalam implementasi Ability permainan. Pada Ability permainan, STL vector digunakan untuk memanipulasi daftar urutan pemain dan kartu pemain. Contoh, pada Ability Reverse, STL vector memudahkan kami untuk mengeluarkan dan memasukkan objek dari sebuah koleksi. Pada kasus ini, koleksi objek yang diubah adalah koleksi giliran pemain.

Pada contoh lain, STL vector digunakan untuk memanipulasi kartu pemain-pemain ketika pemanggilan Swap.

Penggunaan STL map



```

1
2  string ColorCard::getColor() const
3  {
4      map<Color, string> colorMap = {
5          {Color::Red, "Red"},
6          {Color::Yellow, "Yellow"},
7          {Color::Green, "Green"},
8          {Color::Blue, "Blue"}};
9      return colorMap[color_];
10 }
11
12

```

Gambar: Implementasi Method GetColor (ColorCard)

Pada kasus ini, STL map digunakan untuk menyimpan nama enum Color secara terstruktur. Implementasi dengan cara ini juga merapikan kode implementasi method.



```

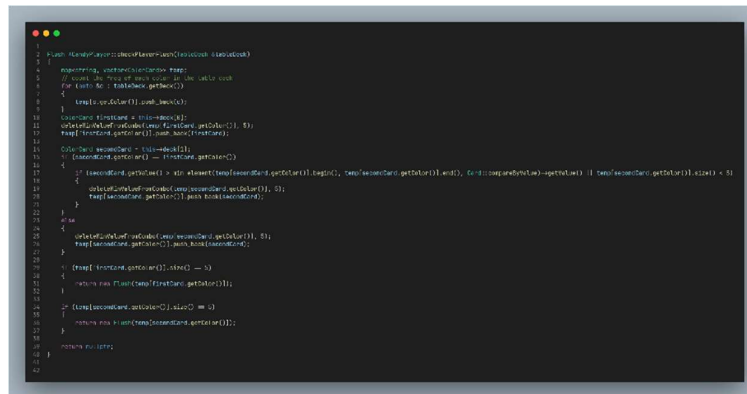
1
2  FourOfAKind *CandyPlayer::checkPlayerFourOfAKind(TableDeck &tableDeck)
3  {
4      map<int, vector<ColorCard>> allFreq;
5      map<int, vector<ColorCard>> playerFreq;
6
7      for (auto &card : this->deck)
8      {
9          playerFreq[card.getValue()].push_back(card);
10         allFreq[card.getValue()].push_back(card);
11     }
12     for (auto &card : tableDeck.getTable())
13     {
14         allFreq[card.getValue()].push_back(card);
15     }
16     for (auto it = allFreq.rbegin(); it != allFreq.rend(); it++)
17     {
18         if (it->second.size() == 4 && playerFreq[it->first].size() > 0)
19         { // jika ukurannya tepat 4 dan ada di deck player.
20             return new FourOfAKind(it->second);
21         }
22     }
23     return nullptr;
24 }
25
26
27
28

```

Gambar: Implementasi Method checkPlayerFourOfAKind (CandyPlayer)

STL map kami gunakan dalam implementasi pengecekan dan pemilihan kombo.

Pada kasus checkPlayerFourOfAKind, STL map memudahkan kami dalam membuat “tabel” frekuensi dari kemunculan angka dari setiap kartu pada Player Deck dan Table Deck. Pada kasus ini, key dari tabel adalah angka kartu, sedangkan value adalah vector kartu.



```

1
2  CandyPlayer::checkPlayerFlush(TableDeck &tableDeck)
3  {
4      map<int, vector<ColorCard>> allFreq;
5      map<int, vector<ColorCard>> playerFreq;
6
7      for (auto &card : this->deck)
8      {
9          playerFreq[card.getColor()].push_back(card);
10         allFreq[card.getColor()].push_back(card);
11     }
12     for (auto &card : tableDeck.getTable())
13     {
14         allFreq[card.getColor()].push_back(card);
15     }
16     for (auto it = allFreq.rbegin(); it != allFreq.rend(); it++)
17     {
18         if (it->second.size() == 5)
19         { // jika ukurannya tepat 5 dan ada di deck player.
20             return new Flush(it->second);
21         }
22     }
23     return nullptr;
24 }
25
26
27
28

```

Gambar: Implementasi Method checkPlayerFlush (CandyPlayer)

Pada kasus checkPlayerFlush, pemetaan menggunakan map dilakukan dengan menghitung kemunculan warna. Pembentukan “tabel” frekuensi dilakukan dengan menggunakan nama warna sebagai key.

Penggunaan STL pair

```

1  #ifndef _PAIR_HPP_
2  #define _PAIR_HPP_
3
4  #include "../Combo.hpp"
5  #include <utility>
6  #include <algorithm>
7  using namespace std;
8
9  class Pair : public Combo
10 {
11 private:
12     pair<ColorCard, ColorCard> cardPair_;
13
14 public:
15     Pair();
16     Pair(pair<ColorCard, ColorCard> cardPair);
17     ~Pair();
18     float getValue() const;
19     void print();
20     pair<ColorCard, ColorCard> getPair();
21     string getComboName() const;
22 };
23
24 #endif
25

```

Gambar: Definisi Kelas Pair

```

1  #ifndef _TWO_PAIR_HPP_
2  #define _TWO_PAIR_HPP_
3
4  #include "../Combo.hpp"
5  #include "../Pair/Pair.hpp"
6  #include <algorithm>
7  using namespace std;
8
9  class TwoPair : public Combo
10 {
11 private:
12     Pair firstPair_;
13     Pair secondPair_;
14
15 public:
16     TwoPair(Pair firstPair, Pair secondPair);
17     ~TwoPair();
18     float getValue() const;
19     void print();
20     string getComboName() const;
21 };
22
23 #endif

```

Gambar: Definisi Kelas TwoPair

Kami menggunakan STL pair pada implementasi kombo Pair. STL pair pada kelas ini digunakan untuk menyimpan dua buah objek kartu. Alasan penggunaan adalah efisiensi dibandingkan penggunaan vector untuk dua objek.

Kelas Pair ini kemudian digunakan pada implementasi kombo TwoPair sebagai atribut.

2.6. Konsep OOP lain

2.6.1. Abstract Base Class

Kami menerapkan *Abstract Base Class* dalam implementasi Valuable dan Combo yang hanya berisi *method pure virtual*. Konsep ini berguna sebagai blueprint dari kelas turunannya agar berbentuk suatu template

```

1 class Valuable {
2     public:
3         virtual ~Valuable();
4         virtual float getValue() const = 0;
5 };

```

```

1 class Combo : public Valuable
2 {
3     protected:
4         static const float HIGH_CARD_MAX;
5         static const float PAIR_MAX;
6         static const float TWO_PAIR_MAX;
7         static const float THREE_KIND_MAX;
8         static const float STRAIGHT_MAX;
9         static const float FLUSH_MAX;
10
11        static const float FULL_HOUSE_MAX;
12        static const float FOUR_KIND_MAX;
13        static const float STRAIGHT_FLUSH_MAX;
14
15    public:
16        Combo();
17        virtual ~Combo();
18        virtual float getValue() const = 0;
19        virtual void print() = 0;
20        virtual string getComboName() const = 0;
21 };

```

2.6.2. Composition

Composition digunakan dalam beberapa kelas yang anggotanya merupakan jenis kelas lain. Misalnya kelas Set memiliki atribut `listOfPlayer` yang merupakan vector dari kelas `Player`, `mainDeck_` yang merupakan kelas `MainDeck`, dan atribut `tableDeck_` yang merupakan kelas `TableDeck`.

```

1 class TableDeck : public InventoryHolder<ColorCard>
2 {
3     private:
4     public:
5         /**
6          * @brief Construct a new Table Deck object
7          *
8          */
9         TableDeck();
10        TableDeck(vector<ColorCard> cards);
11        void addCard(MainDeck &);
12        void addCard(ColorCard);
13        virtual TableDeck &operator=(const TableDeck &other);
14        void print();
15 };
16

```

```

1 class MainDeck : public InventoryHolder<ColorCard>
2 {
3     private:
4     public:
5         /**
6          * @brief Construct a new Main Deck object
7          *
8          */
9         MainDeck();
10        ~MainDeck();
11        void fillDeck();
12        void randomizeCard();
13        void readCard();
14        ColorCard getFromMainDeck();
15        void addCard(ColorCard card);
16        MainDeck &
17        operator=(const MainDeck &other);
18        void print();
19        Color stringToColor(const string &color);
20 };

```

3. Bonus Yang dikerjakan

3.1. Bonus Game Cangkul

Kami mengimplementasikan permainan kartu lain yaitu “Cangkul” yang kami beri nama “MasBro Cangkul”. Permainan “MasBro Cangkul” memiliki aturan sebagai berikut :

1. Pada awal permainan, akan dipasang satu *table card* yang akan menjadi penentu warna dari ronde pertama
2. Pada setiap ronde, semua pemain membuang kartu yang warnanya sama dengan *table card*. Jika pemain tidak memiliki kartu yang berwarna sama, pemain akan mengambil kartu dari deck sampai mendapatkan kartu yang berwarna sama dengan *table card*.
3. Pemain yang membuang kartu dengan angka terbesar menjadi pemenang ronde.
4. Pemenang ronde berhak menentukan *table card* untuk ronde berikutnya

5. Permainan dilanjutkan sampai ditemukan pemenang, yaitu pemain yang tidak memiliki kartu pada tangannya.

Implementasi permainan tambahan ini dilakukan dengan membuat dua buah class tambahan yaitu CangkulGame dan CangkulPlayer yang diturunkan dari class Player. Berikut adalah header dari CangkulGame dan CangkulPlayer :

CangkulGame.hpp	CangkulPlayer.hpp
<pre> 1 #ifndef _CANGKULGAME_HPP 2 #define _CANGKULGAME_HPP_ 3 4 #include <iostream> 5 #include <unistd.h> 6 #include "../Valuable/Player/CangkulPlayer.hpp" 7 #include "../InventoryHolder/MainDeck/MainDeck.hpp" 8 #include "../Util/Coloring.hpp" 9 10 using namespace std; 11 12 class CangkulGame 13 { 14 private: 15 vector<CangkulPlayer> listOfPlayers_; 16 MainDeck mainDeck_; 17 vector<ColorCard> thrownCards_; 18 19 public: 20 CangkulGame(); 21 ~CangkulGame(); 22 void startGame(); 23 int getCardChoice(CangkulPlayer &roundWinner); 24 ColorCard *takeCardFromDeck(CangkulPlayer &p, ColorCard &tableCard); 25 }; 26 27 #endif </pre>	<pre> 1 #ifndef _CANGKULPLAYER_HPP_ 2 #define _CANGKULPLAYER_HPP_ 3 4 #include "Player.hpp" 5 6 #include <string> 7 #include <vector> 8 #include <map> 9 #include <iostream> 10 #include <algorithm> 11 12 using namespace std; 13 14 class CangkulPlayer : public Player 15 { 16 private: 17 public: 18 CangkulPlayer(); 19 20 /** 21 * @brief Construct a new Player object 22 * 23 * @param nickname 24 */ 25 CangkulPlayer(string nickname); 26 27 /** 28 * @brief Destroy the Player object 29 * 30 */ 31 ~CangkulPlayer(); 32 33 /** 34 * @brief Print player's info including their nickname and points 35 * 36 */ 37 void print(); 38 39 ColorCard *getLargestCard(ColorCard &tableCard); 40 41 int getCardChoice(int n); 42 }; 43 44 #endif </pre>

Berikut adalah cuplikan tampilan dari permainan “MasBro Cangkul” :

```

=====
Welcome to MasbroParty Card Game
=====

Are you ready to start the game? [Y/N] : Y

Which game do you want to play?
[1] MasBro Candy
[2] MasBro Cangkul
Input option : 2

MASBRO CANGKUL

Enter the nickname of Player 1: Viel
Enter the nickname of Player 2: Rachel
Enter the nickname of Player 3: Eugene
Enter the nickname of Player 4: Jason

```



```

Table card :
[7]
[7]

It's Viel's turn!

Your cards :
[5] [1] [10] [11] [1] [13] [5]
[5] [1] [10] [11] [1] [13] [5]

Options :
[1]
[5]
[5]

[2]
[13]
[13]

[3] Pass

Choose card to open : 

```

```

Choose card to open : 3
Oops! too bad.. you can't open any card

Press enter key to take a card from deck...

You got a nonsuitable card :(

[13]
[13]

Press enter key to take a card from deck...

You got a nonsuitable card :(

[9]
[9]

Press enter key to take a card from deck...

You found a suitable card!

[12]
[12]

Viel's opened card

[12]
[12]

End of Viel's turn

```

3.2. Bonus Kreasi Mandiri

Pada implementasi program kami, kami menambahkan fitur tambahan berupa pewarnaan tampilan CLI permainan. Pewarnaan ini diterapkan untuk menunjukkan warna dari kartu, serta mewarnai pesan-pesan khusus dari permainan.

Berikut adalah header yang digunakan untuk implementasi fitur pewarnaan:

```
Coloring.hpp
```

```

1  #ifndef _COLORING_HPP_
2  #define _COLORING_HPP_
3
4  #include <iostream>
5  class Coloring{
6      public:
7          void red(bool bold = false);
8          void green(bool bold = false);
9          void blue(bool bold = false);
10         void yellow(bool bold = false);
11         void lgreen(bool bold = false);
12         void pink(bool bold = false);
13         void cyan(bool bold = false);
14         void white(bool bold = false);
15         void reset();
16     };
17
18 #endif

```

Berikut adalah cuplikan tampilan dari CLI setelah dilakukan pewarnaan :

```

=====
Welcome to MasbroParty Card Game
=====

Are you ready to start the game? [Y/N] : y

MASBROPARTY

Enter the nickname of Player 1: Vie1
Enter the nickname of Player 2: Rach
Enter the nickname of Player 3: Liv
Enter the nickname of Player 4: Chow
Enter the nickname of Player 5: Yuj
Enter the nickname of Player 6: Jas
Enter the nickname of Player 7: Catur

How do you want to fill the Main Deck?
[1] Auto Randomized
[2] Input from file .txt
Option : █

```


4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Command (Double, Half, Next)	13521044	13521044 13521046 13521074 13521094 13521100
Ability (Abilityless, Quadruple, Quarter, Re-Roll, Reverse, Swap, Switch)	13521094	13521044 13521046 13521074 13521094 13521100
Exception	13521046 13521094 13521100	13521044 13521046 13521074 13521094 13521100
Game (CandyGame, CangkulGame)	13521044 13521094	13521044 13521046 13521074 13521094 13521100
InventoryHolder (MainDeck, TableDeck)	13521094	13521044 13521046 13521074 13521094 13521100
Combo	13521074 13521100	13521044 13521046 13521074 13521094 13521100
Player (CandyPlayer, CangkulPlayer)	13521044 13521046	13521044 13521046 13521074 13521094 13521100
Set, SetProcess	13521044 13521094 13521100	13521044 13521046 13521074 13521094 13521100
Card (ColorCard)	13521044 13521046	13521044 13521046 13521074 13521094 13521100
UI (Coloring)	13521046 13521074	13521044 13521046 13521074 13521094 13521100
Laporan	13521046 13521074 13521094 13521100	

LAMPIRAN: <https://github.com/chaerla/MasbroParty>

Kode Kelompok : CBR

Nama Kelompok : masbro

1. 13521044 / Rachel Gabriela Chen

2. 13521046 / Jeffrey Chow

3. 13521074 / Eugene Yap Jin Quan

4. 13521094 / Angela Livia Arumsari

5. 13521100 / Alexander Jason

Asisten Pembimbing : Steven Nathaniel Kodyat

1. Konten Diskusi

- Class-class yang perlu dibuat secara umum.

- Q : Bagaimana implementasi virtual function value?

A : Buat Class Valuable, turunkan menjadi Player, ColorCard, dan Combo

- Q : Bagaimana implementasi abstract class InventoryHolder?

A : Sebenarnya implementasinya dibebaskan. Boleh dibuat menjadi generic class kemudian diturunkan menjadi PlayerDeck, PlayerList, MainDeck, etc.

- Q : Apakah sudah benar jika setiap ability diturunkan menjadi subclass dari class Ability?

A : Sudah benar. Method activate di class Ability sudah benar, tetapi jangan lupa bahwa method tersebut harus menerima parameter objek-objek yang akan dimanipulasi karena ability tersebut.

- Q : Apakah sudah benar jika setiap combo yang ada diturunkan menjadi subclass dari class Combo?

A : Sudah benar.

- Q : Kalau misalnya sudah ada combo straight flush artinya tidak perlu mengecek combo yang lain?

A : Tidak. Karena bisa saja combo straight flush itu didapatkan dari table card yang ada sehingga value dari setiap player sama.

2. Tindak Lanjut

Dari hasil konten diskusi, kalian selanjutnya mau melakukan apa? Dapat ditulis dalam format poin-per-poin atau paragraf.

- Mulai mengimplementasikan kelas-kelas yang perlu dibuat.
- Memperbaiki struktur kelas dan membuat kelas Valuable.