

Kode Kelompok : WOI

Nama Kelompok : WOI

1. 13521063 / Salomo Reinhart Gregory Manalu
2. 13521069 / Louis Caesa Kusuma
3. 13521071 / Margaretha Olivia Haryono
4. 13521085 / Addin Munawwar Yusuf
5. 13521119 / Muhammad Rizky Sya'ban

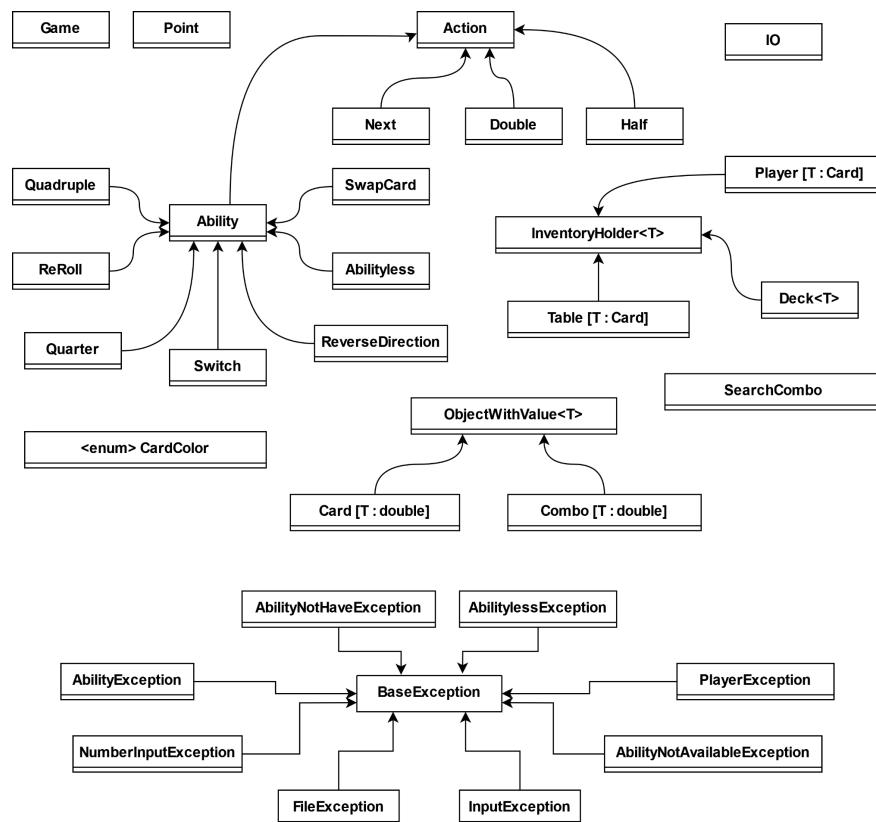
Asisten Pembimbing : Hafid Abi Daniswara

Link repo : https://github.com/margarethaolivia/Tubes-1-OOP_WOI



1. Diagram Kelas

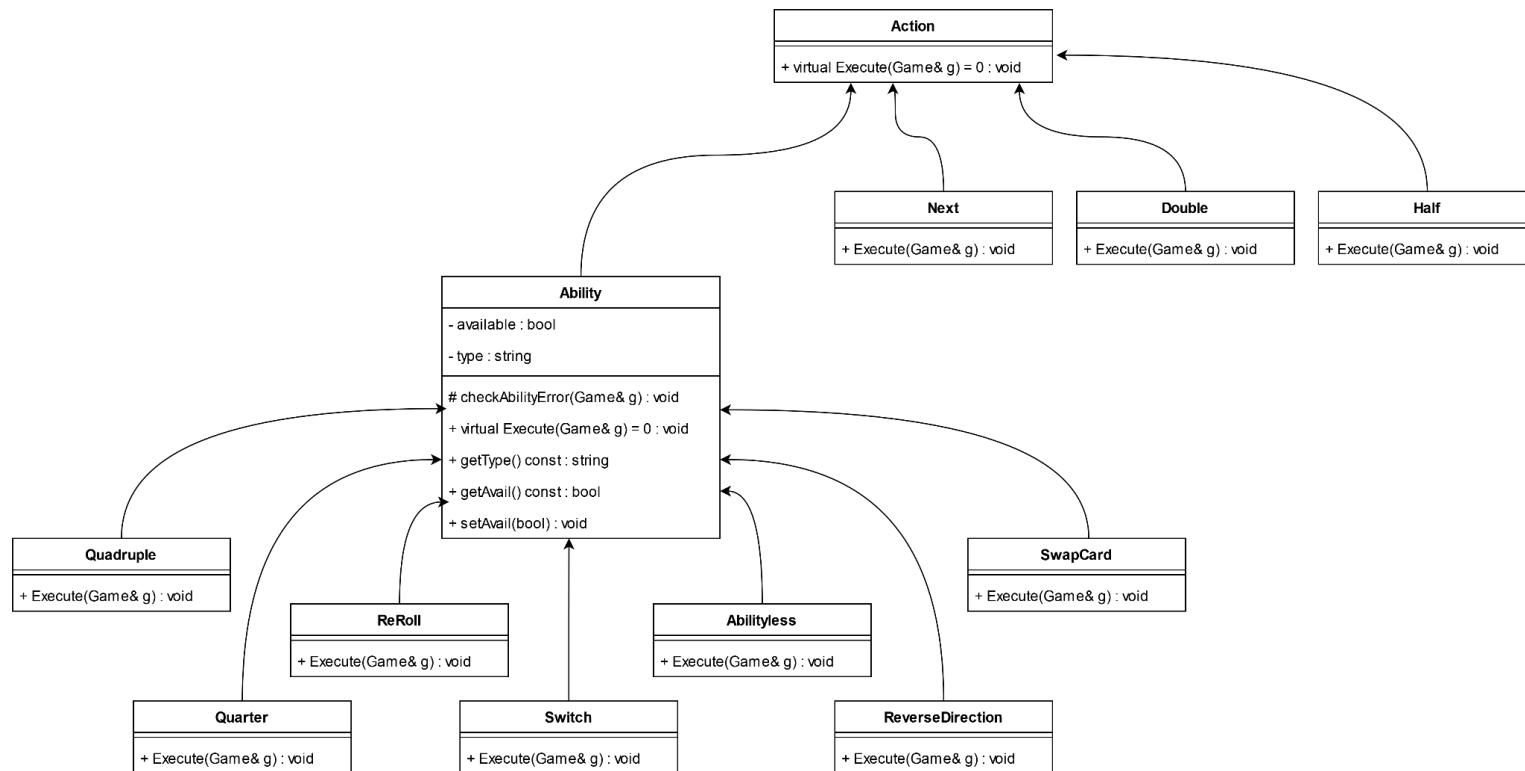
Diagram kelas atau *Class diagram* merupakan salah satu jenis diagram yang umum digunakan dalam pemodelan perangkat lunak atau aplikasi yang menggunakan pendekatan berorientasi objek. Diagram kelas berguna untuk menggambarkan struktur dari sebuah sistem ataupun program. Diagram kelas menunjukkan kelas-kelas, atributnya, serta relasinya dengan kelas-kelas lain.



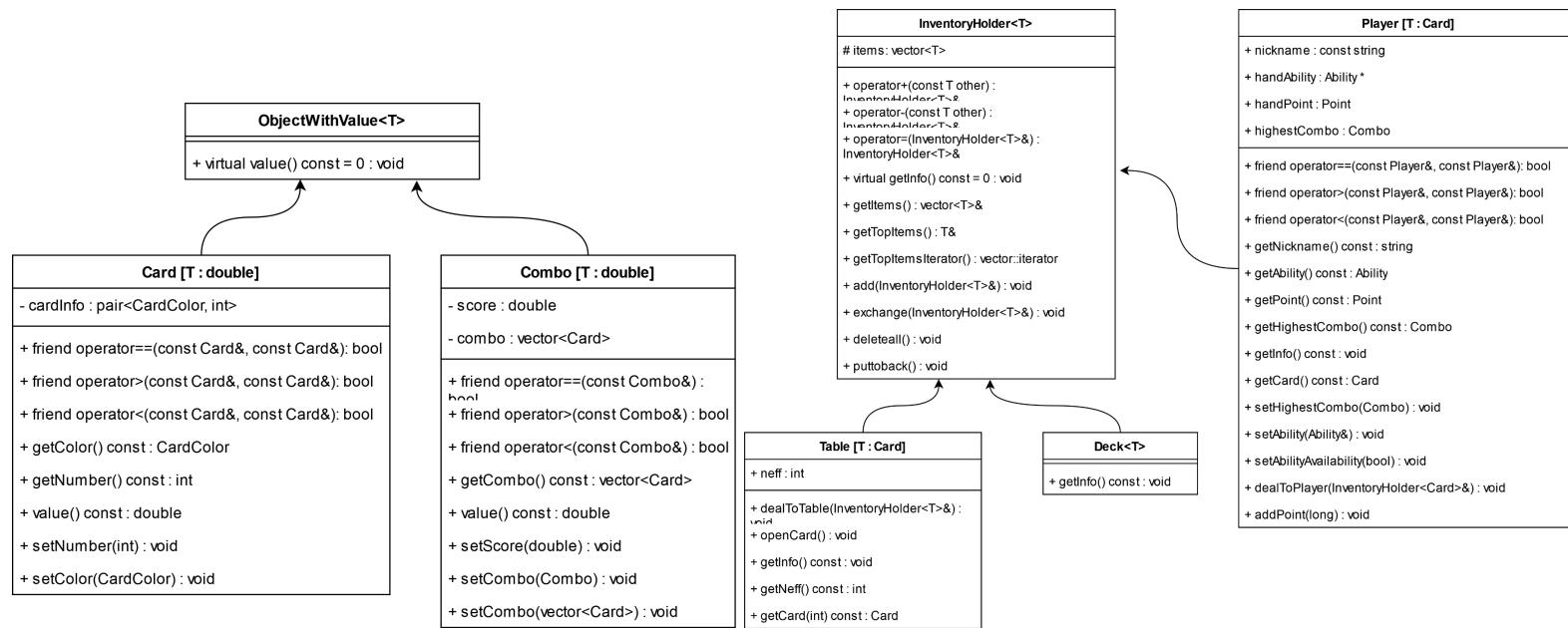
Gambar 1.1. Diagram kelas aplikasi secara umum

Gambar di atas merupakan diagram kelas secara umum permainan Candy Kingdom kelompok WOI. Hubungan antar kelas pada diagram di atas digambarkan dengan tanda panah untuk hubungan pewarisan, maupun secara hubungan secara implisit yang berupa komposit (kelas menjadi atribut pada kelas lain).

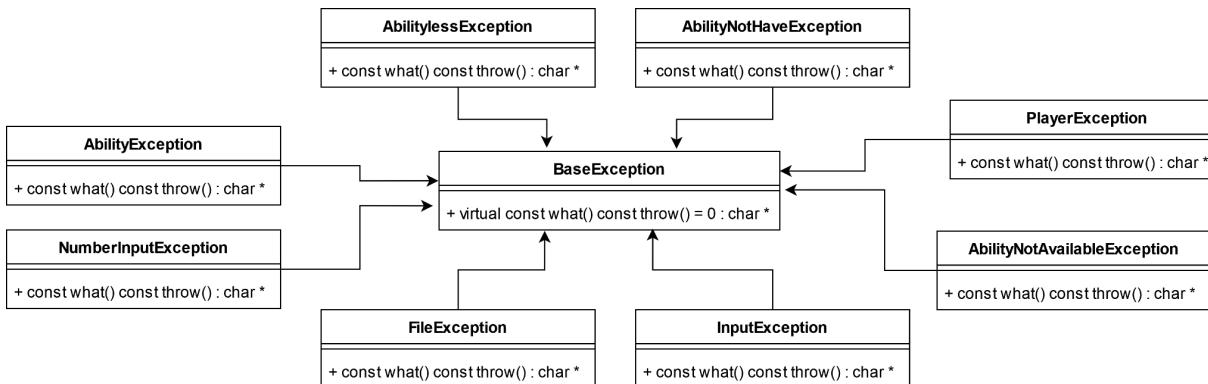
Pada diagram di atas, dapat terlihat beberapa kelas yang memiliki hubungan pewarisan, yaitu Action, Ability, InventoryHolder, dan juga BaseException.



Gambar 1.2. Diagram kelas untuk kelas Action dan anak-anaknya



Gambar 1.3. Diagram kelas untuk kelas InventoryHolder dan ObjectWithValue



Gambar 1.4. Diagram kelas untuk kelas Exception dan anak-anaknya

Desain seperti ini dipilih untuk memanfaatkan konsep pada pemrograman berorientasi objek, yaitu *Inheritance* dan *Polymorphism*. Kelas yang menjadi anak dari kelas lain (parent) mewarisi sifat yang dimiliki oleh parent-nya, termasuk atribut dan method-methodnya. Hal ini sangat bermanfaat, karena kelas dengan tingkah laku yang sama dapat digabung ke kelas yang lebih umum, dan kelas tersebut dapat di-ekstensi lebih lanjut (spesialisasi) berdasarkan tingkah laku yang diinginkan.

Selain itu, desain ini juga memungkinkan *polymorphism*, yaitu suatu kelas dapat memiliki banyak bentuk. Misalnya, kelas ability, bisa saja merupakan ability Re-Roll, bisa juga Reverse, maupun yang lainnya selama *runtime*. Meskipun ability tersebut merupakan kelas tersendiri, akan tetapi ability ini dapat memiliki tingkah laku yang berubah-ubah saat *runtime* mengikuti tingkah laku anaknya.

Selain kelas-kelas di atas, terdapat juga kelas-kelas yang tidak memiliki hubungan pewarisan.

Game	Point	SearchCombo	IO	<enum> CardColor
<pre>+ players : vector<pair<Player&, bool>> + cardDeck : Deck<Card> + abilityDeck : Deck<Ability*> + table : Table + point : Point + isReversed : bool + playerTurn : int + round : int + currentPlayer : int + getPlayers() : vector<pair<Player&, bool>& + getCardDeck() : Deck<Card>& + getAbilityDeck() : Deck<Ability*>& + getTable() : Table& + getPoint() : Point& + getReverseInfo() : bool& + getPlayerTurn() : int& + getRound() : int& + getCurrentPlayer() : int& + addPlayer() : void + incCurrentPlayer() : void + decCurrentPlayer() : void + setReverseInfo(bool) : void</pre>	<pre>- value : long long + getValue() const : long long + setValue(long long) : void + Half() : string + Double() : bool + Quadruple() : void + Quarter() : void + Add(long long) : void</pre>	<pre>- cards : vector<Card> - records : vector<Combo> + getCombo() const : Combo + getCards() const : vector<Card> + getRecords() const : vector<Combo> + getHighestCombo() : Combo + setCombo(Combo) : void + setCards(vector<Card>) : void + setRecords(vector<Combo>) : void + sortCards() : void + sortCards2() : void + highCard() : void + pair() : void + twoPair() : void + threeOfAKind() : void + straight() : void + flush() : void + fullHouse() : void + fourOfAKind() : void + straightFlush() : void + checkCombo() : void</pre>	<pre>- { colors: string } + splashScreen() : void + printThankYou() : void + printTable() : void + printEndMatch() : void + printEndGame() : void + mainMenuTitle() : void + mainMenu() : string + inputPlayerName() : vector<string> + turnInput() : string + dealMenu() : string + printAbilitySuccess(Player, vector<string>, vector<string>) : void + printAbilitySuccess(Player&, Point&) : void + printActionSuccess() : void + selectPlayer(Player&, vector<pair<Player&, bool>&, string) : vector<Player*> + selectCard(string) : int + printAbilityless(string) : void + inputFile() : pair<vector<Card>, vector<Ability*>> + stringToAbility(string) : Ability*</pre>	<pre>Green = 0 Blue = 1 Yellow = 2 Red = 3</pre>
Game (cont.)				

Gambar 1.4. Kelas-kelas yang tidak memiliki hubungan pewarisan

Meskipun kelas-kelas di atas tidak memiliki hubungan pewarisan, bukan berarti kelas tersebut tidak memiliki hubungan dengan kelas lain. Hubungan antar kelas yang ada di atas adalah hubungan komposit, dimana suatu kelas menjadi komponen/atribut dari kelas lain. Pada umumnya, kelas-kelas ini adalah kelas dasar, ataupun kelas-kelas yang sifatnya sebagai *controller* pada sistem atau program.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Inheritance adalah konsep pemrograman berorientasi objek (OOP) yang memungkinkan kelas baru dibuat dengan mewarisi properti dan metode dari kelas yang ada. Kelas yang sudah ada disebut sebagai superclass atau superklass, sedangkan kelas yang baru dibuat disebut sebagai subclass atau subclass. Melalui pewarisan, subclass dapat memiliki semua properti dan metode superclass, dan juga menambah atau mengubah properti dan metode yang ada, atau menambahkan yang baru.

Polymorphism adalah konsep OOP yang memungkinkan suatu objek memiliki banyak bentuk atau aspek. Dalam OOP, polymorphism dapat diimplementasikan dengan dua cara yaitu overriding dan overloading. Overriding terjadi ketika subclass mengubah penerapan metode yang sudah ada di superclass sehingga metode tersebut berperilaku berbeda dari subclass. Overloading terjadi ketika sebuah kelas memiliki beberapa metode dengan nama yang sama tetapi parameter atau tipe data berbeda, sehingga pemanggilan metode tergantung pada parameter atau tipe data yang digunakan. Polimorfisme memungkinkan kode menjadi lebih fleksibel dan menangani banyak kasus umum. Berikut ini adalah contoh penggunaan konsep *inheritance* dan *polymorphism*.

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... ObjectWithValue.hpp ...
src > ObjectWithValue > ObjectWithValue.hpp ...
1 /* _ should this be generic or no? */
2
3 Abstract class for Combo and Card
4 Has a pure virtual function for value
5
6 Combo : return double;
7 Card : return int;
8
9 */
10 */
11 #ifndef _OBJECTWITHVALUE_HPP_
12 #define _OBJECTWITHVALUE_HPP_
13
14 template <class T>
15 class ObjectWithValue {
16 public:
17     virtual T value() const = 0;
18 };
19
20#endif

```

Gambar 2.1.1 Kelas *ObjectWithValue*

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... ObjectWithValue.hpp Combo.hpp ...
src > ObjectWithValue > ObjectWithValue.hpp ...
1 #ifndef COMBO_HPP_
2 #define COMBO_HPP_
3
4 #include <vector>
5 #include <algorithm>
6 #include "../Card/Card.hpp"
7 // #include "../Player/Player.hpp"
8 // #include "../Utility/Utility.hpp"
9 #include "../ObjectWithValue/ObjectWithValue.hpp"
10 // #include "../Exception/Exception.hpp"
11 // #include "../Utility/ArrayUtil.hpp"
12 // #include "SearchCard.hpp"
13
14 class Combo : public ObjectWithValue<double> {
15 private:
16     double score;
17     vector<Card> combo;
18 public:
19     Combo(); // ctor
20     ~Combo(); // destructor
21     void operator=(const Combo& combo); // constructor
22     void setScore(double number); // Setter
23     double getScore() const; // Score Getter
24     void setCards(const vector<Card>& cards); // Cards Setter
25     void getCards(vector<Card> & cards); // Cards Getter
26     void setCombo(Combo& combo); // Combo Setter
27     void setCombo(vector<Card> & cards); // Combo Setter
28
29     void sortCombo(); // sorting combo by number and color
30
31     bool operator<(const Combo& other);
32     bool operator>(const Combo& other);
33     bool operator==(const Combo& other);
34
35 };
36
37

```

Gambar 2.1.2 Kelas *Combo*

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... ObjectWithValue.hpp Card.hpp ...
src > Card > Card.hpp ...
1 /* Card class header */
2
3 */
4
5 #ifndef CARD_HPP_
6 #define CARD_HPP_
7
8 #include "../ObjectWithValue/ObjectWithValue.hpp"
9 #include <iostream>
10 #include <string>
11 #include <map>
12 using namespace std;
13
14 enum CardColor {
15     /* Card Color */
16     Green = 0,
17     Blue = 1,
18     Yellow = 2,
19     Red = 3
20 };
21
22 const map<CardColor, string> cardToString = {
23     /* Mapping color to int for */
24     {Green, "Green"}, // Green
25     {Blue, "Blue"}, // Blue
26     {Yellow, "Yellow"}, // Yellow
27     {Red, "Red"} // Red
28 };
29
30 class Card : public ObjectWithValue<double> {
31 private:
32     pair<CardColor, int> cardInfo;
33     // TO BE ASKED : map for mapping card color to int, instead of implicit def. from enum (?)
34     public:
35     Card(CardColor, int);
36     Card(const Card&); //cctor
37     ~Card(); //destructor
38
39 };

```

Gambar 2.1.3 Kelas *Card*

Gambar di atas merupakan salah satu implementasi dari *inheritance* dan *polymorphism* yang ada di tugas besar ini. *ObjectWithValue* merupakan sebuah kelas abstrak. *ObjectWithValue* hanya memiliki 1 *pure virtual function* yang harus diimplementasikan di turunan-turunan kelas tersebut. Fungsi tersebut adalah *value()*. Kelas turunan dari *ObjectWithValue* adalah kelas *Card* dan kelas *Combo*. *value()* di kelas *Card* mengembalikan nilai dari sebuah kartu, sedangkan *value()* di kelas *Combo* mengembalikan nilai dari sebuah kombinasi kartu yang dimiliki oleh *player*. Konsep *inheritance* dan *polymorphism* dalam kasus ini cocok digunakan. Dalam permainan ini, perlu diketahui nilai dari kombinasi untuk memenangkan game dan untuk mengetahui nilai dari kombinasi, perlu juga diketahui nilai dari kartu.

Contoh lain penggunaan konsep *inheritance* dan *polymorphism* pada tugas besar kelompok ini adalah kelas *Ability*.

```

File Edit Selection View Go Run Terminal Help
Ability.hpp - Tubes-1-OOP_WOI - Visual Studio Code
EXPLORER OUTLINE ObjectWithValue.hpp Ability.hpp ...
src > Ability > Ability.hpp > ...
11  class Ability;
12  class Player;
13
14  class Ability : public Action {
15      private:
16          bool available;
17          string type;
18
19      protected:
20          /* Util */
21          void checkAbilityError(Game& g);
22
23      public:
24          Ability(string);
25          /* Execute, pass the game state to ability to act accordingly */
26          virtual void Execute(Game& g) = 0;
27
28          /* Getter */
29          string getType() const;
30          bool getAvail() const;
31
32          /* Setter */
33          void setAvail(bool);
34      };
35
36  class ReRoll: public Ability {
37      private:
38
39      public:
40          ReRoll();
41          void Execute(Game& g); // lihat situasi
42      };
43
44  class Quadruple: public Ability {
45      private:
46
47      public:
48          Quadruple(); // default ctor
49          void Execute(Game& g);
50      };

```

Gambar 2.1.4 Kelas Ability

Ability merupakan sebuah kelas abstrak yang memiliki atribut *available* dan *type* dan sebuah method yang pure virtua yaitu *Execute()*. *Ability* memiliki beberapa kelas turunan. Kelas turunan tersebut adalah *ReRoll*, *Quadruple*, *Quarter*, *SwapCard*, *Switch*, *Abilityless*, dan

ReverseDirection. Setiap kelas turunan akan mengimplementasikan *Execute()*. Setiap kelas akan memberikan perubahan pada game dan player yang berbeda-beda. Dengan adanya konsep *inheritance* dan *polymorphism*, kita dapat melakukan berbagai ability dengan memberikan dampak yang berbeda terhadap game. Ini merupakan keuntungan *inheritance* dan *polymorphism*.

2.2. Method/Operator Overloading

Operator overloading adalah konsep pemrograman berorientasi objek (OOP) di mana operator seperti +, -, *, / dan lainnya dapat didefinisikan ulang atau perilakunya diubah sehingga dapat diterapkan ke objek kelas. Dalam OOP, setiap operator memiliki perilaku default yang ditentukan dalam bahasa pemrograman, misalnya operator + digunakan untuk menjumlahkan angka. Namun, kelebihan operator dapat mengubah perilaku default ini sehingga operator dapat digunakan pada objek dari kelas yang berbeda.

Overloading operator dapat diterapkan pada operator matematika, operator pembanding, operator logika, dan lainnya. Misalnya, dalam kelas yang merepresentasikan bilangan kompleks, operator + dapat didefinisikan ulang untuk menjumlahkan dua bilangan kompleks. Demikian pula, untuk kelas yang merepresentasikan objek sebagai string, operator + dapat didefinisikan ulang agar dapat menggabungkan dua string.

Untuk melakukan operator overloading, pemrogram harus mendefinisikan metode di kelas dengan nama yang sama dengan operator yang perilakunya akan dimodifikasi. Berikut di bawah ini merupakan contoh penggunaan konsep operator overloading.

```

File Edit Selection View Go Run Terminal Help
InventoryHolder.hpp - Tubes-1-OOP_WOI - Visual Studio Code
EXPLORER ... < InventoryHolder.hpp > ...
src > InventoryHolder > < InventoryHolder.hpp > ...
18     InventoryHolder();
19     ~InventoryHolder()
20 {
21     while (!this->items.empty())
22     {
23         this->items.pop_back();
24     }
25 }
26
27 InventoryHolder(const InventoryHolder& other) {
28     this->items = other.items;
29 }
30
31 InventoryHolder<T>& operator+(const T other){
32     items.push_back(other);
33     return *this;
34 }
35
36 InventoryHolder<T>& operator-(const T other){
37     auto itr = this->items.begin();
38     while (!(itr == other)) {
39         itr++;
40     }
41     items.erase(itr);
42
43     return *this;
44 }
45
46 InventoryHolder<T>& operator=(InventoryHolder<T>& other){
47     this->items = other.items;
48     return *this;
49 }
50
51 vector<T>& getItems() {
52     return this->items;
53 }
54

```

Gambar 2.2.1 Kelas *InventoryHolder*

Konsep operator overloading digunakan di beberapa kelas pada tugas besar ini. Salah satu implementasinya adalah pada kelas *InventoryHolder*. *InventoryHolder* merupakan sebuah kelas abstrak yang memiliki atribut *items*. *Items* merupakan sebuah array yang terdiri beberapa tipe data. Untuk melakukan operasi pada atribut *items*, kelas *InventoryHolder* memiliki beberapa operator overloading untuk melakukan operasi-operasi tersebut. Operator overloading pada *InventoryHolder* adalah operator +, -, dan =. Operator+ memiliki fungsi untuk menambahkan suatu objek ke dalam *items* dengan tipe data yang sama. Operator- memiliki fungsi untuk menghapus suatu objek ke dari *items*.

Operator= memiliki fungsi untuk melakukan *assignment*. Misalkan *items* = *otherItems*, maka *otherItems* di assign ke *items*. Operator-operator overloading ini banyak dimanfaatkan di beberapa method yang dimiliki oleh *InventoryHolder* dan kelas turunan *InventoryHolder* seperti *add()*, *exchange()*, dan *puttoback()*. Keuntungan dari operator overloading adalah kode menjadi lebih sederhana dan mengurangi perulangan kode.

2.3. Template & Generic Classes

Template adalah cetak biru atau kerangka kerja yang digunakan untuk membuat objek baru dengan karakteristik yang sama. Dalam pemrograman, template sering digunakan untuk menghasilkan kode yang sama tetapi dengan nilai yang berbeda. Contoh umum model adalah model fungsi atau model kelas. Dalam model fungsional, blok kode dapat diimplementasikan dengan nilai yang berbeda tanpa harus menulis kode yang sama berulang kali. Dalam model kelas, sebuah kelas dapat dibuat dengan tipe data yang berbeda tanpa harus menulis kelas yang sama lebih dari satu kali.

Kelas generic adalah kelas yang dirancang untuk digunakan dengan tipe tipe data yang berbeda. Dalam bahasa pemrograman, kelas generik memungkinkan programmer untuk menentukan tipe data saat menggunakan kelas tersebut. Ini memungkinkan pengembang untuk menggunakan kelas yang sama dengan tipe data yang berbeda tanpa harus menulis kelas yang sama berulang kali.

Berikut di bawah ini merupakan contoh penerapan konsep template dan generic class.

The screenshot shows a Visual Studio Code interface with the following details:

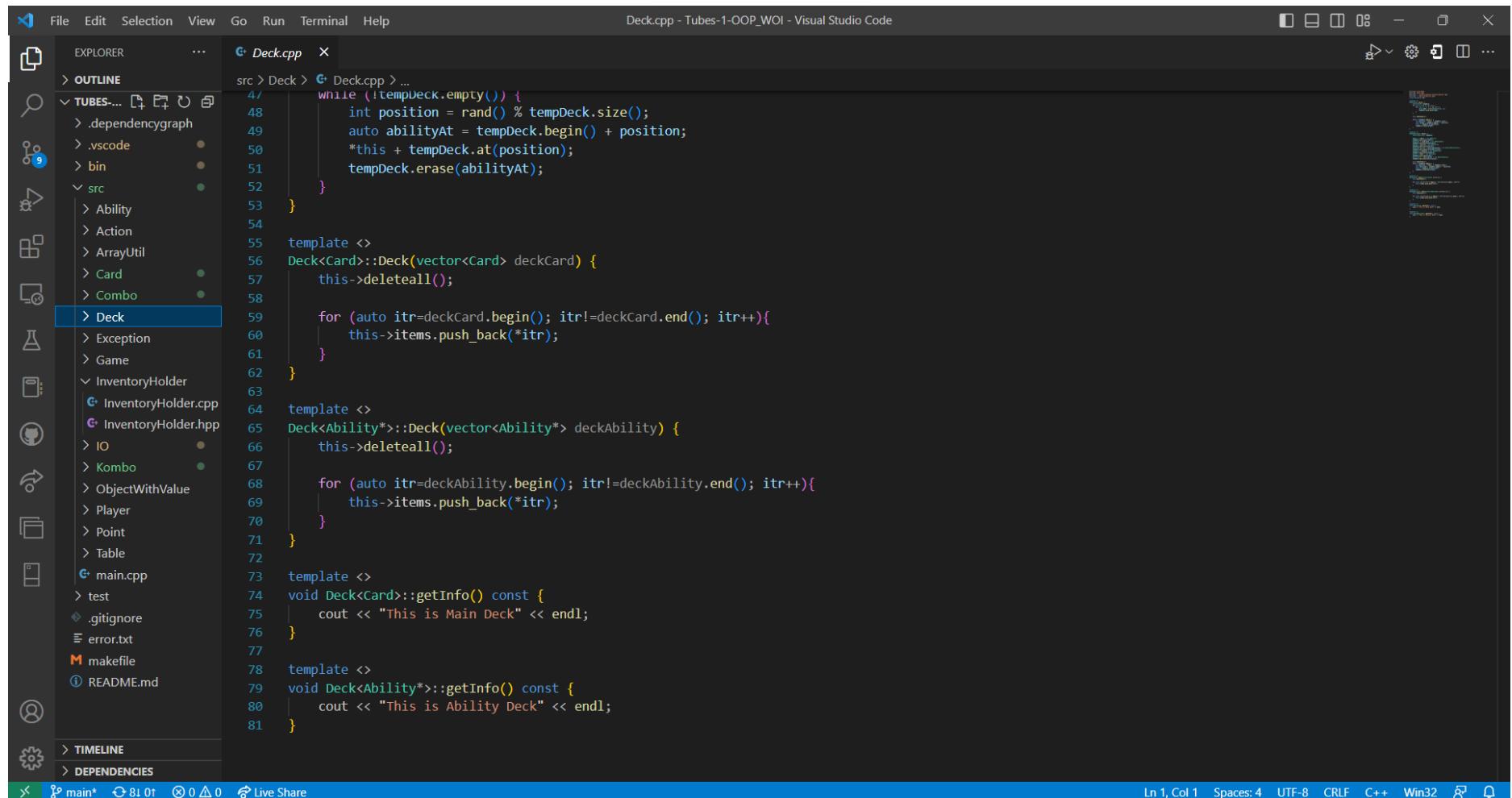
- File Explorer (Left):** Shows the project structure under "TUBES-1-OOP_WOI". The "src" folder contains subfolders like Ability, Action, ArrayUtil, Card, Combo, Deck, Exception, Game, InventoryHolder, IO, Kombo, ObjectWithValue, Player, Point, Table, and main.cpp. It also lists files like driverDeck.cpp, .gitignore, error.txt, and makefile.
- Code Editor (Center):** Displays the content of the "Deck.hpp" file.

```
#ifndef DECK_HPP
#define DECK_HPP
#include <iostream>
#include <stdlib.h>
#include <vector>
#include "../InventoryHolder/InventoryHolder.hpp"
#include "../Card/Card.hpp"
using namespace std;

template <class T>
class Deck: public InventoryHolder<T> {
protected:
public:
    Deck();
    Deck(vector<T>);
    void getInfo() const;
};

#endif
```
- Bottom Status Bar:** Shows file status (main.cpp), line count (81), character count (0), and other settings like spaces: 4, UTF-8, CRLF, C++, Win32, and a Live Share icon.

Gambar 2.3.1 Kelas *Deck* (File Header)



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Deck.cpp - Tubes-1-OOP_WOI - Visual Studio Code.
- Explorer Panel (Left):**
 - OUTLINE:** Shows a tree view of the project structure. The **Deck** class under the **src** folder is selected, highlighted with a blue border.
 - Sources:** TUBES..., .dependencygraph, .vscode, bin, src (containing Ability, Action, ArrayUtil, Card, Combo, Deck), InventoryHolder (containing InventoryHolder.cpp, InventoryHolder.hpp), IO, Kombo, ObjectWithValue, Player, Point, Table, main.cpp, test, .gitignore, error.txt, makefile, README.md.
 - Timeline and Dependencies:** Buttons for Timeline and Dependencies.
- Editor Area (Center):** Displays the content of the Deck.cpp file. The code is a generic class definition for **Deck** that inherits from **InventoryHolder**. It includes methods for adding cards and abilities to the deck and printing information about the deck.
- Bottom Status Bar:** ShowsLn 1, Col 1, Spaces: 4, UTF-8, CRLF, C++, Win32, and a few icons.

```

Deck.cpp

src > Deck > Deck.cpp > ...
4/   while (!tempDeck.empty()) {
48     int position = rand() % tempDeck.size();
49     auto abilityAt = tempDeck.begin() + position;
50     *this + tempDeck.at(position);
51     tempDeck.erase(abilityAt);
52   }
53 }
54
55 template <>
56 Deck<Card>::Deck(vector<Card> deckCard) {
57   this->deleteall();
58
59   for (auto itr=deckCard.begin(); itr!=deckCard.end(); itr++){
60     this->items.push_back(*itr);
61   }
62 }
63
64 template <>
65 Deck<Ability*>::Deck(vector<Ability*> deckAbility) {
66   this->deleteall();
67
68   for (auto itr=deckAbility.begin(); itr!=deckAbility.end(); itr++){
69     this->items.push_back(*itr);
70   }
71 }
72
73 template <>
74 void Deck<Card>::getInfo() const {
75   cout << "This is Main Deck" << endl;
76 }
77
78 template <>
79 void Deck<Ability*>::getInfo() const {
80   cout << "This is Ability Deck" << endl;
81 }

```

Gambar 2.3.2 Kelas *Deck* (File cpp)

Dalam tugas besar ini, ada beberapa class yang merupakan class generic. Salah satunya adalah kelas *Deck*. *Deck* merupakan sebuah kelas turunan dari kelas *InventoryHolder*. Kelas *Deck* memiliki atribut *items* (warisan dari parent class). *Deck* merupakan class generic yang dimana

atribut *items*-nya dengan dapat menyimpan beberapa tipe data. Pada tugas besar ini, kelas Deck menyimpan tipe data Card dan Ability. Kelas Deck menyimpan kartu dan ability sehingga nantinya kartu dan ability dapat diberikan kepada table dan player. Keuntungan dari template dan generic class adalah mengurangi kode yang berulang. Kita dapat dengan mudah melakukan operasi yang sama untuk 2 tipe yang berbeda (Card dan Ability) atau lebih dengan menggunakan template dan generic class.

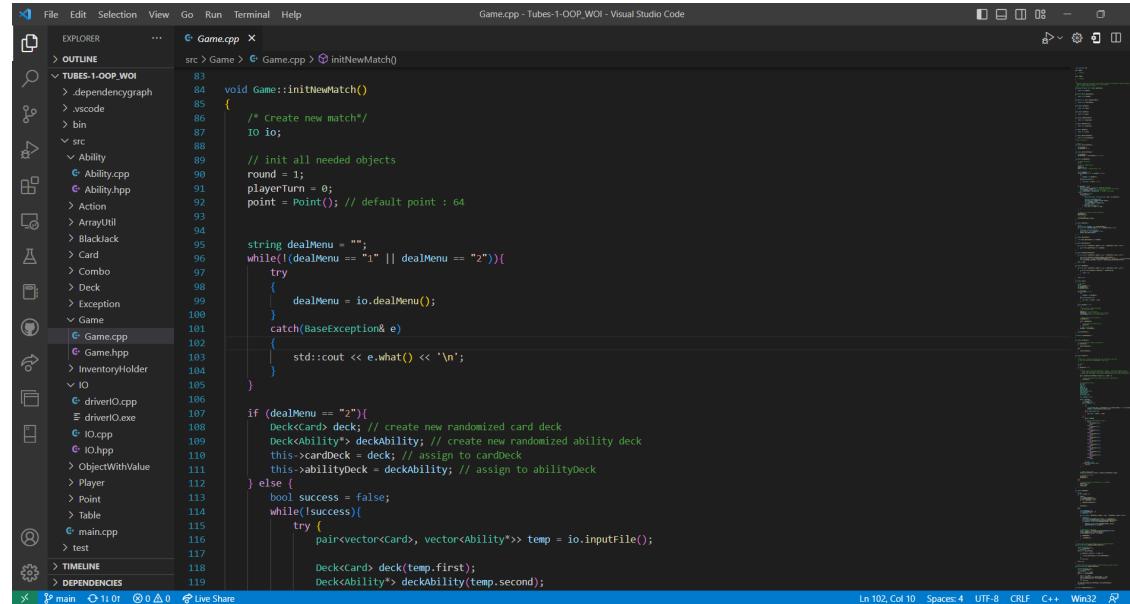
2.4. Exception

Exception adalah suatu mekanisme untuk menangani kesalahan atau kondisi abnormal dalam program. Ketika suatu kesalahan terjadi, program dapat menghasilkan sebuah exception, yang kemudian dapat ditangani oleh kode program untuk memperbaiki kesalahan tersebut. Dalam C++, ada beberapa jenis exception yang umumnya digunakan:

1. std::exception
Ini adalah kelas dasar untuk semua exception dalam C++. Kita dapat membuat subclass dari std::exception untuk membuat exception kustom dengan pesan yang lebih spesifik.
2. std::runtime_error
Ini adalah kelas yang digunakan untuk menghasilkan exception yang terkait dengan runtime. Misalnya, jika suatu file yang diperlukan tidak dapat ditemukan saat program dijalankan, exception std::runtime_error dapat dihasilkan.
3. std::logic_error
Ini adalah kelas yang digunakan untuk menghasilkan exception yang terkait dengan logika program. Misalnya, jika suatu program mencoba untuk membagi bilangan dengan nol, exception std::logic_error dapat dihasilkan.
4. std::bad_alloc
Ini adalah kelas yang digunakan untuk menghasilkan exception yang terkait dengan alokasi memori yang gagal. Misalnya, jika suatu program mencoba untuk mengalokasikan terlalu banyak memori, exception std::bad_alloc dapat dihasilkan.

Untuk menangani exception, kita dapat menggunakan blok try-catch. Blok try digunakan untuk mencoba menjalankan kode tertentu, sementara blok catch digunakan untuk menangkap dan menangani exception yang dihasilkan dari blok try. Jika exception terjadi dalam blok try, maka kode dalam blok catch yang sesuai dengan jenis exception yang dihasilkan akan dieksekusi. Berikut beberapa penerapan exception dalam tugas besar ini.

BaseException



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure under "OUTLINE". The "src" folder contains subfolders like Ability, Action, ArrayUtil, Blackjack, Card, Combo, Deck, Exception, Game, IO, InventoryHolder, ObjectWithValue, Player, Point, Table, and main.cpp. It also lists files like Game.cpp, Game.hpp, driverIO.cpp, driverIO.exe, IO.cpp, IO.hpp, and main.cpp.
- Editor:** The main editor window displays the "Game.cpp" file with the following code:

```
83     void Game::initNewMatch()
84     {
85         /* Create new match*/
86         IO io;
87
88         // init all needed objects
89         round = 1;
90         playerturn = 0;
91         point = Point(); // default point : 64
92
93         string dealMenu = "";
94         while((dealMenu == "1" || dealMenu == "2")){
95             try
96             {
97                 dealMenu = io.dealMenu();
98             }
99             catch(BaseException& e)
100            {
101                std::cout << e.what() << '\n';
102            }
103        }
104
105        if (dealMenu == "2"){
106            Deck<Card> deck; // create new randomized card deck
107            Deck<Ability*> deckAbility; // create new randomized ability deck
108            this->cardDeck = deck; // assign to cardDeck
109            this->abilityDeck = deckAbility; // assign to abilityDeck
110        } else {
111            bool success = false;
112            while(!success){
113                try {
114                    pair<vector<Card>, vector<Ability*>> temp = io.inputFile();
115
116                    Deck<Card> deck(temp.first);
117                    Deck<Ability*> deckAbility(temp.second);
118
119                }
120            }
121        }
122    }
```

The status bar at the bottom shows: Ln 102, Col 10, Spaces: 4, UTF-8, CRLF, C++, Win32.

Gambar 2.4.1 Implementasi BaseException

NumberInputException

```
116 string IO::mainMenu(){
117     cout << endl;
118     cout << enterColor << "Enter Command >>> ";
119
120     string command;
121     cout << inputColor; cin >> command; cout << resetColor;
122     if (!(command == "1" || command == "2")){
123         NumberInputException err;
124         throw err;
125     }
126     cout << resetColor << endl;
127     return command;
128 }
```

Gambar 2.4.2 Implementasi NumberInputException

FileException

```

405     int count = 0;
406     while(getline(infile, line)){
407         istringstream iss(line);
408         string code;
409         while (iss >> code) {
410             // 52 first word is for main deck
411             if (count < 52){
412                 if (code.length() == 2){
413                     Card card(CardColor(1), 0);
414                     card.setNumber(code[0] - '0');
415                     card.setColor(stringToColor.at(code[1]));
416                     mainDeck.push_back(card);
417                     count++;
418                 } else if (code.length() == 3){
419                     Card card(CardColor(1), 0);
420                     card.setNumber((code[0] - '0')*10 + (code[1] - '0'));
421                     card.setColor(stringToColor.at(code[2]));
422                     mainDeck.push_back(card);
423                     count++;
424                 } else {
425                     throw err;
426                 }
427             } // 7 other word for ability deck
428             } else if (count <=59){
429                 if (code.length() == 2){
430                     abilityDeck.push_back(stringToAbility(code));
431                     count++;
432                 } else {
433                     throw err;
434                 }
435             } else {
436                 throw err;
437             }
438         }
439     }
440 }
```

Gambar 2.4.3 Implementasi FileException

PlayerException

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "TUBES-...". The "Ability.cpp" file is selected.
- Code Editor:** Displays the code for the Ability class, specifically the implementation of the execute method. It includes exception handling for AbilityNotAvailableException and AbilityNotHaveException.
- Status Bar:** Shows the current file is "Ability.cpp", line 55, column 1, with settings for spaces: 4, UTF-8, CRLF, C++, Win32, and a live share icon.

```
/* Switch Card Ability */
Switch::Switch(): Ability("Switch") {}

void Switch::Execute(Game& g) {
    try {
        checkAbilityError(g);
    } catch (AbilityNotAvailableException& e) {
        cout << e.what() << endl;
        throw e;
        return;
    } catch (AbilityNotHaveException& e) {
        cout << e.what() << this->getType() << endl;
        throw e;
        return;
    }
    IO io;
    // Player& owner = (g.getPlayers().begin() + g.getCurrentPlayer())->first;

    Player& owner = (g.getPlayers()[g.getCurrentPlayer()].first);

    vector<Player *> target;
    // select Player
    bool playerSelected = false;
    while (!playerSelected) {
        try {
            target = io.selectPlayer(owner, g.getPlayers(), g.getPlayers()[g.getCurrentPlayer()].first.getNickname());
            playerSelected = true;
        } catch (PlayerException& e) {
            cout << e.what() << endl;
            continue;
        }
    }

    Player& other = *target[0];
    owner.exchange(other);
    owner.exchange(other);
}
```

Gambar 2.4.4 Implementasi PlayerException

AbilityNotAvailable

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Ability.cpp - Tubes-1-OOP_WOI - Visual Studio Code.
- Explorer:** Shows the project structure under 'TUBES...'. The 'Ability' folder is expanded, showing 'Ability.cpp' which is currently selected.
- Code Editor:** Displays the code for Ability.cpp. The relevant part is:

```
51 void Ability::setAvail(bool avail)
52 {
53     this->available = avail;
54 }
55
56 /* Re-roll Ability */
57 ReRoll::ReRoll() : Ability("Re-Roll") {}
58
59 void ReRoll::Execute(Game& g) {
60     try {
61         checkAbilityError(g);
62     } catch (AbilityNotAvailableException& e) {
63         cout << e.what() << endl;
64         throw e;
65     } catch (AbilityNotHaveException& e) {
66         cout << e.what() << this->getType() << endl;
67         throw e;
68     }
69 }
70
71 Player& owner = ((g.getPlayers().begin() + g.getCurrentPlayer())->first);
72 owner.deleteAll();
73 owner.add(g.getCardDeck());
74 owner.add(g.getCardDeck());
75
76 IO io;
77 io.printAbilitySuccess(owner);
78 owner.setAbilityAvailability(false);
79 }
80
81 /* Quadruple Ability */
82 Quadruple::Quadruple() : Ability("Quadruple") {}
83
84 void Quadruple::Execute(Game &g){
85     try {
86         checkAbilityError(g);
87     }
```

The code implements the AbilityNotAvailableException class, which is caught in the Execute method of the ReRoll class. The exception is thrown if the ability is not available.

Gambar 2.4.5 Implementasi AbilityNotAvailableException

AbilityNotHaveException

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Ability.cpp, Ability.hpp, main.cpp, and others.
- Code Editor:** Displays the `Ability.cpp` file content. The code implements exception handling for `AbilityNotHaveException`.
- Status Bar:** Shows file information (main.cpp), line and column numbers (Ln 55, Col 1), and encoding (UTF-8).

```
Ability.cpp - Tubes-1-OOP_WOI - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
> OUTLINE
src > Ability > Ability.cpp ...
src > Ability > Ability.hpp
src > Ability > Action
src > Ability > ArrayUtil
src > Ability > Blackjack
src > Ability > Card
src > Ability > Combo
src > Ability > Deck
src > Ability > Exception
src > Ability > Game
src > Ability > InventoryHolder
src > Ability > IO
src > Ability > ObjectWithValue
src > Ability > Player
src > Ability > Point
src > Ability > Table
src > Ability > main.cpp
src > Ability > test
src > Ability > .gitignore
src > Ability > error.txt
src > Ability > makefile
src > Ability > README.md

Ability.cpp
Ability.hpp
Action
ArrayUtil
Blackjack
Card
Combo
Deck
Exception
Game
InventoryHolder
IO
ObjectWithValue
Player
Point
Table
main.cpp
test
.gitignore
error.txt
makefile
README.md

TIMELINE
DEPENDENCIES

Ability.cpp

51 void Ability::setAvail(bool avail)
52 {
53     this->available = avail;
54 }
55
56 /* Re-roll Ability */
57 ReRoll::ReRoll() : Ability("Re-Roll") {}
58
59 void ReRoll::Execute(Game& g) {
60     try {
61         checkAbilityError(g);
62     } catch (AbilityNotAvailableException& e) {
63         cout << e.what() << endl;
64         throw e;
65     } catch (AbilityNotHaveException& e) {
66         cout << e.what() << this->getType() << endl;
67         throw e;
68     }
69 }
70
71 Player& owner = *(g.getPlayers().begin() + g.getCurrentPlayer()->first);
72 owner.deleteAll();
73 owner.add(g.getCardDeck());
74 owner.add(g.getCardDeck());
75
76 IO io;
77 io.printAbilitySuccess(owner);
78 owner.setAbilityAvailability(false);
79 }
80
81 /* Quadruple Ability */
82 Quadruple::Quadruple() : Ability("Quadruple") {}
83
84 void Quadruple::Execute(Game &g){
85     try {
86         checkAbilityError(g);
87     }
```

Gambar 2.4.6 Implementasi FileException

2.5. C++ Standard Template Library

STL (Standard Template Library) adalah sebuah kumpulan template class dan fungsi yang disediakan oleh bahasa pemrograman C++. STL menyediakan sebuah kerangka kerja yang lengkap untuk melakukan pemrograman dengan pendekatan generic programming, yang dapat mempercepat pengembangan aplikasi dan meningkatkan kualitas kode.

Beberapa kelas yang termasuk dalam STL antara lain:

1. Container : STL menyediakan beberapa kelas container seperti vector, list, deque, set, dan map. Container classes ini dapat digunakan untuk menyimpan dan mengelola sekumpulan data, baik itu berupa tipe data primitif maupun objek.
2. Iterator : STL menyediakan iterator classes yang digunakan untuk mengakses data dalam container classes. Iterator classes memungkinkan kita untuk mengakses dan memanipulasi data dalam container dengan cara yang sama, terlepas dari jenis data yang disimpan dalam container.
3. Algorithm : STL menyediakan beberapa kelas algoritma seperti sorting, searching, dan counting. Algoritma-algoritma ini dapat digunakan untuk memanipulasi data dalam container classes dengan cara yang efisien dan mudah.

Berikut di bawah ini penggunaan STL pada tugas besar ini.

Vector

```

1 #ifndef _COMBO_HPP_
2 #define _COMBO_HPP_
3
4 #include <vector>
5 #include <algorithm>
6 #include "../Card/card.hpp"
7 // include "../Player/player.hpp"
8 // include "../Table/Table.hpp"
9 #include "../ObjectWithValue/ObjectWithValue.hpp"
10 // include "../exception/exception.h"
11 // #include "../ArrayUtil/ArrayUtil.hpp"
12 // #include "SearchCombo.hpp"
13
14 class Combo : public ObjectWithValue<double> {
15     private:
16         double score;
17         vector<Card> combo;
18
19     public:
20         Combo(); // ctor
21         ~Combo(SearchCombo combo); // constructor
22         ~Combo(); // destructor
23
24         double value() const; // Score Getter
25         void setScore(double number); // Setter
26         vector<Card> getCombo() const; // Combo Getter
27         void setCombo(Combo); // Combo Setter
28         void sortCombo(); // Sorting combo by number and color
29
30         bool operator<(const Combo& other);
31         bool operator>(const Combo& other);
32         bool operator==(const Combo& other);
33
34     };
35
36 };
37

```

Gambar 2.5.1 Penggunaan vector pada Kelas Card

Vector digunakan di banyak class pada tugas besar ini. Salah satu contohnya adalah pada kelas Combo. Combo memiliki atribut score dan combo. Atribut combo merupakan sebuah vector yang berisi tipe data Card. Vector<Card> ini akan menyimpan kombinasi kartu yang dimiliki oleh seorang player. Dalam kasus ini, vector sangat mempermudah proses mengolah kombinasi, seperti mengiterasi, *push_back*, dan juga menghapus elemen di vector.

Pair

```

File Edit Selection View Go Run Terminal Help
Card.hpp - Tubes-1-OOP.WOI - Visual Studio Code
EXPLORER OUTLINE Card.hpp ...
src > Card > Card.hpp ...
30 class Card : public ObjectWithValue<double> {
31     private:
32         pair<CardColor, int> cardInfo;
33         // TO BE ASKED : map for mapping card color to int, instead of implicit def. from enum (?)
34     public:
35         Card(CardColor, int);
36         Card(const Card&); //ctor
37         ~Card(); //destructor
38         Card& operator=(const Card&); //assignment op.
39
40         /* Operator Overload */
41         friend bool operator==(const Card&, const Card&);
42         friend bool operator>(const Card&, const Card&);
43         friend bool operator<(const Card&, const Card&);
44
45         /* Accessor */
46         CardColor getColor() const;
47         int getNumber() const;
48         double value() const; // inherited from ObjectWithValue<double>
49
50         /* Setter */
51         void setNumber(int);
52         void setColor(CardColor);
53     };
54
55 #endif

```

Gambar 2.5.2 Penggunaan pair pada Kelas Card

Pair digunakan di kelas Card. Kelas Card memiliki sebuah atribut yaitu cardInfo. cardInfo merupakan sebuah tipe data pair. Dengan pair, cardInfo dapat menyimpan 2 tipe data yaitu CardColor dan integer. Sebuah kartu memiliki warna dan angkanya. Warna dan angka ini disimpan di dalam pair. Dengan mudah, program dapat menyimpan informasi mengenai warna dan angka dari kartu. Satu buah objek bisa terdiri atas 2 tipe data yang berbeda karena adanya pair.

Map

```

3  */
4
5 #ifndef _CARD_HPP_
6 #define _CARD_HPP_
7
8 #include "../ObjectWithValue/ObjectWithValue.hpp"
9 #include <utility>
10 #include <string>
11 #include <map>
12 using namespace std;
13
14 enum CardColor {
15 /* Card color */
16     Green = 0,
17     Blue = 1,
18     Yellow = 2,
19     Red = 3
20 };
21
22 const map<CardColor, string> cardToString = {
23 /* Mapping color to int for */
24     {Green, "Green"},
25     {Blue, "Blue"},
26     {Yellow, "Yellow"},
27     {Red, "Red"}
28 };
29

```

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `Card.hpp`, `Card.cpp`, `main.cpp`, and `driverCard.cpp`.
- Code Editor:** Displays the `Card.hpp` file containing C++ code that uses a map to map `CardColor` enum values to strings.
- Terminal:** Shows command-line history related to a git restore operation.
- Bottom Status Bar:** Shows file information (main.cpp), line and column (Ln 1, Col 1), and encoding (UTF-8).

Gambar 2.5.3 Penggunaan Map

Di tugas besar ini, map digunakan untuk mendefinisikan beberapa konstanta. Contoh penggunaan map adalah pada const `cardToString`. Dengan menggunakan map pada constanta `cardToString`, program dapat mencetak warna (dalam bentuk string). Map pada const `cardToString` berisi 2 tipe data, yaitu `cardColor` dan `string`. Dengan mengakses satu jenis warna, kita dapat mendapatkan warna tersebut dalam bentuk string. Contohnya, jika ingin mencetak `cardColor` `Green`, maka kita dapat mendapatkan string “`Green`”.

2.6. Konsep OOP lain

Selain 5 konsep OOP yang diwajibkan di atas, ada beberapa konsep lain yang diimplementasikan pada permainan Candy Kingdom kelompok WOI.

Abstract Base Class adalah kelas yang memiliki minimal satu method yang pure virtual. Maksud dari method pure virtual adalah method yang harus diimplementasikan oleh kelas anaknya, dan kelas parent tidak memiliki implementasi dari kelas tersebut. Kelas abstrak tidak dapat diinstansiasi secara langsung, akan tetapi harus melalui anaknya. Kelas yang merupakan kelas abstrak pada permainan ini adalah kelas *InventoryHolder*, *Ability*, dan *ObjectWithValue*.

Composition adalah konsep dimana kelas menjadi komponen atau atribut dari kelas lain. Contoh kelas yang mengimplementasikan hal ini adalah kelas *Game*, yang memiliki banyak composite class, seperti *Player*, *Deck*, *Table*, dan lain-lain.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Generic Class

Generic class diterapkan pada class *InventoryHolder*. *InventoryHolder* memiliki sebuah atribut yaitu *items* yang merupakan sebuah vector. *items* ini dapat menyimpan berbagai tipe jenis data. Hal ini dapat dilakukan dengan cara membuat class *InventoryHolder* menjadi sebuah generic class.

```

11  template <class T>
12  class InventoryHolder
13  {
14      protected:
15      vector<T> items;
16
17      public:
18      InventoryHolder(){};
19      ~InventoryHolder()
20      {
21          while (!this->items.empty())
22          {
23              this->items.pop_back();
24          }
25      }
26
27      InventoryHolder(const InventoryHolder& other) {
28          this->items = other.items;
29      }
30
31      InventoryHolder<T>& operator+(const T other){
32          items.push_back(other);
33      }
34
35      void print() const
36      {
37          cout << "InventoryHolder<" << typeid(T).name() << ">:" << endl;
38          for (const auto& item : items)
39          {
40              cout << item << endl;
41          }
42      }
43
44      void clear()
45      {
46          items.clear();
47      }
48
49      const vector<T>& getItems() const
50      {
51          return items;
52      }
53
54      void setItems(const vector<T> &items)
55      {
56          this->items = items;
57      }
58
59      void addItems(const T &item)
60      {
61          items.push_back(item);
62      }
63
64      void removeItem(const T &item)
65      {
66          items.erase(remove(items.begin(), items.end(), item), items.end());
67      }
68
69      void removeIndex(int index)
70      {
71          if (index < 0 || index >= items.size())
72          {
73              throw std::out_of_range("Index out of range");
74          }
75          items.erase(items.begin() + index);
76      }
77
78      int size() const
79      {
80          return items.size();
81      }
82
83      bool isEmpty() const
84      {
85          return items.empty();
86      }
87
88      void sort()
89      {
90          sort(items.begin(), items.end());
91      }
92
93      void reverse()
94      {
95          reverse(items.begin(), items.end());
96      }
97
98      void copy(InventoryHolder &other)
99      {
100         other.items = items;
101     }
102 }
```

Gambar 3.1.1.1 Implementasi Generic Class pada Kelas *InventoryHolder*

Dari cuplikan kode di atas, dapat dilihat bahwa vector *items* dapat menyimpan tipe data T (template<class T>), sehingga memungkinkan bagi *InventoryHolder* dan kelas turunannya untuk menyimpan berbagai tipe data. Dalam tugas besar ini, *InventoryHolder* digunakan untuk menyimpan tipe data *Card* dan *Ability*.

3.1.2. Game Kartu Lain

Game kartu lain yang dibuat dalam tugas besar ini adalah **BlackJack**. Blackjack adalah permainan kartu yang dimainkan dengan satu atau lebih deck kartu. Tujuan permainan ini adalah untuk mengalahkan dealer dengan mendapatkan nilai kartu yang lebih tinggi dari dealer tanpa melebihi 21.

3.2. Bonus Kreasi Mandiri



Gambar 3.2.1 Splash Screen

Salah satu kreasi mandiri dalam tugas besar ini adalah membuat splash screen dengan ASCII art seperti pada *splash screen* di atas. Selain itu, setiap IO (input maupun output) yang terdapat pada tugas besar ini memberi warna pada string yang di cetak ke layar maupun input dari pengguna. Dengan adanya pemberian warna ini, tampilan terminal menjadi lebih menarik. Tampilan warna kartu pada table juga disesuaikan dengan CardColor dari kartu tersebut. Apabila CardColor = Green, maka tampilan warna kartu akan hijau.

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Ability	13521063 13521069 13521071 13521085	13521063 13521069 13521071 13521085

	13521119	13521119
Action	13521069	13521063 13521069 13521071 13521085 13521119
ArrayUtil	13521085	13521063 13521069 13521071 13521085 13521119
Card	13521085	13521063 13521069 13521071 13521085 13521119
Combo	13521063	13521063 13521069 13521071 13521085 13521119
SearchCombo	13521063	13521063 13521069 13521071 13521085 13521119
Deck	13521069	13521063 13521069 13521071

		13521085 13521119
Exception	13521071	13521063 13521069 13521071 13521085 13521119
Game	13521063 13521069 13521071 13521085 13521119	13521063 13521069 13521071 13521085 13521119
GameEngine	13521069 13521119	13521063 13521069 13521071 13521085 13521119
InventoryHolder	13521071	13521063 13521069 13521071 13521085 13521119
IO	13521119	13521063 13521069 13521071 13521085 13521119
ObjectWithValue	13521085	13521063 13521069

		13521071 13521085 13521119
Player	13521119	13521063 13521069 13521071 13521085 13521119
Point	13521085	13521063 13521069 13521071 13521085 13521119
Table	13521069	13521063 13521069 13521071 13521085 13521119

Kode Kelompok : WOI

Nama Kelompok : WOI

1. 13521063 / Salomo Reinhart Gregory Manalu
2. 13521069 / Louis Caesa Kusuma
3. 13521071 / Margaretha Olivia Haryono
4. 13521085 / Addin Munawwar Yusuf
5. 13521119 / Muhammad Rizky Sya'ban

Asisten Pembimbing : Hafid Abi Daniswara

1. Konten Diskusi

- maksud dari hanya memiliki satu purpose untuk kelas:
objek kartu hanya boleh berisi kartu, begitu juga buat kelas-kelas lainnya
- parser:
tidak wajib (boleh memakai cin)
- Kondisi game berakhir
 - adalah ketika ada salah satu pemain mencapai jumlah poin (ada di spek)
 - Per ronde cuma dikasih 2 kartu, tidak ada banding-banding dulu
 - Bandingkan kartu yang kita punya sama table card di akhir game (ronde akhir)
 - Berdasarkan rumus, yang paling tinggi angkanya, yang menang
- Ada kemungkinan lebih dari satu player skornya sama?
Hampir tidak ada karena ada prioritas tiap warna kartu
- Teknis reverse
Reverse sisa pemain, lalu ronde-ronde berikutnya tetap ke-reverse juga (permanen)
- Untuk kartu meja apakah dirandom? Ada batasan library?
Pakai library bawaan (#include <cstdlib>)

- Untuk pembagian kartu ke pemain:
Tidak ada ketentuan, yang penting tidak ada pemain yang memiliki kartu yang sama
- Validasi input:
Tidak wajib, bisa diasumsikan inputnya benar

2. Tindak Lanjut

- Membuat kelas player, kartu, kombo, meja, inventory holder, ability, parent combo dan kartu, dan kelas command
- Membuat kelas game untuk mengatur jalannya permainan
- List player memakai vector of pair
 - first : player
 - second : status (sudah jalan atau belum)

Kalo reverse, urutan playernya mundur (indeks mundur)

- Membuat kelas poin untuk menyimpan poin dari game saat ini
- Kelas ability jadi abstract class, setiap ability jadi kelas anaknya

3. Dokumentasi

