

Kode Kelompok : OEB

Nama Kelompok : OOPeZ

1. 13521059 / Arleen Chrysantha Gunardi
2. 13521124 / Michael Jonathan Halim
3. 13521127 / Marcel Ryan Antony
4. 13521143 / Raynard Tanadi
5. 13521145 / Kenneth Dave Bahana

Asisten Pembimbing : Aditya Bimawan (NIM. 13519064)

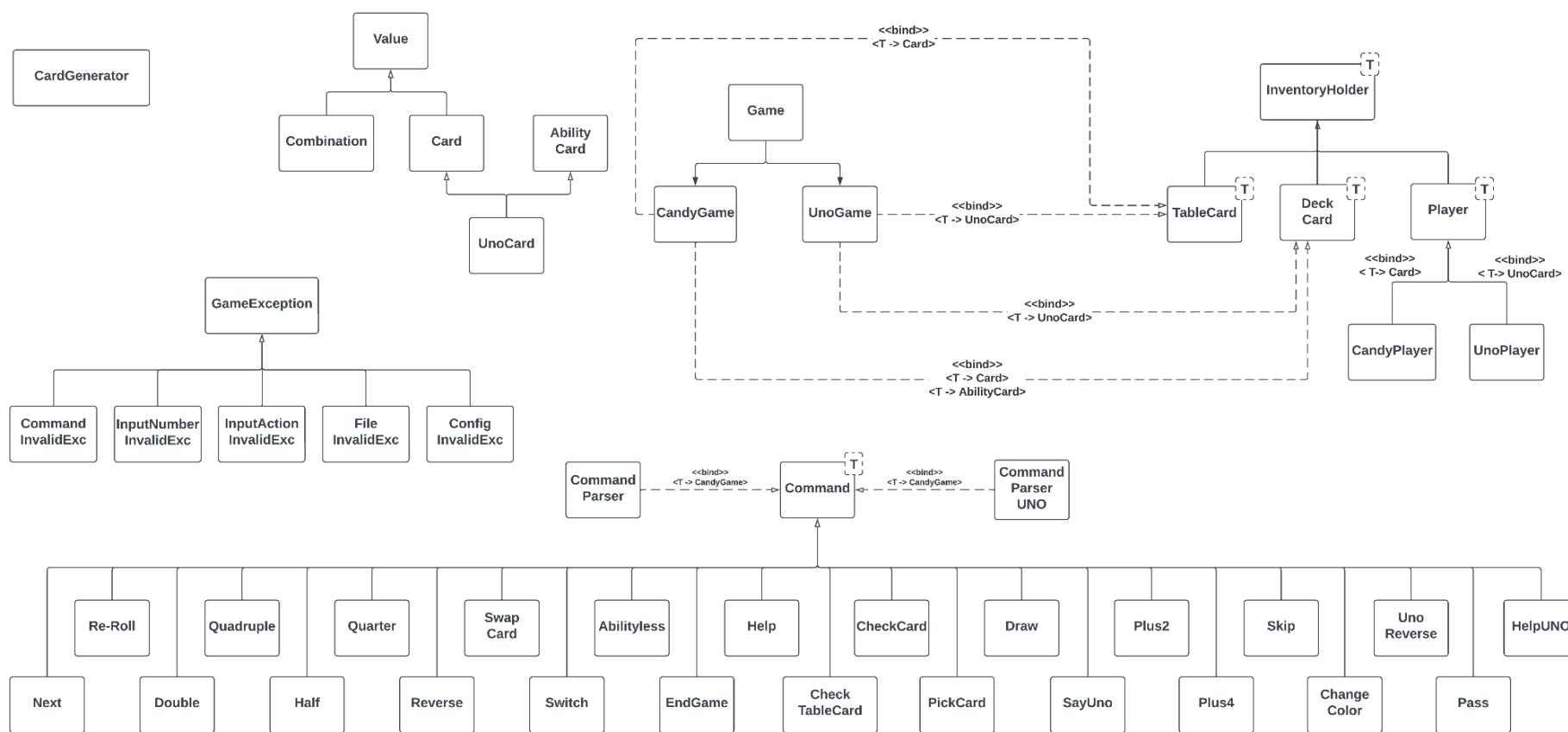
1. Diagram Kelas

Untuk membuat permainan Kompetisi Kartu ala Kerajaan Permen, konsep pemrograman berorientasi objek (*object-oriented programming*). Program yang dibuat dengan konsep orientasi objek terdiri dari kelas-kelas yang mengenkapsulasi data dan juga metode. Oleh karena itu, pada awal perancangan program, dibutuhkan *blueprint* objek dan kelas yang dibutuhkan. Rancangan *blueprint* tersebut digambarkan dalam sebuah diagram kelas. Diagram kelas merepresentasikan struktur setiap kelas beserta dengan atribut dan metodenya.

Berikut adalah hierarki dan diagram kelas yang dirancang untuk permainan Kompetisi Kartu ala Kerajaan Permen.

a. Hierarki Kelas

Rancangan hierarki kelas beserta hubungan antar kelas direpresentasikan dalam diagram hierarki kelas. Adapun rancangan hierarki kelas pada permainan Kompetisi Kartu ala Kerajaan Permen adalah sebagai berikut.



Gambar 1.1 Rancangan Hierarki Kelas

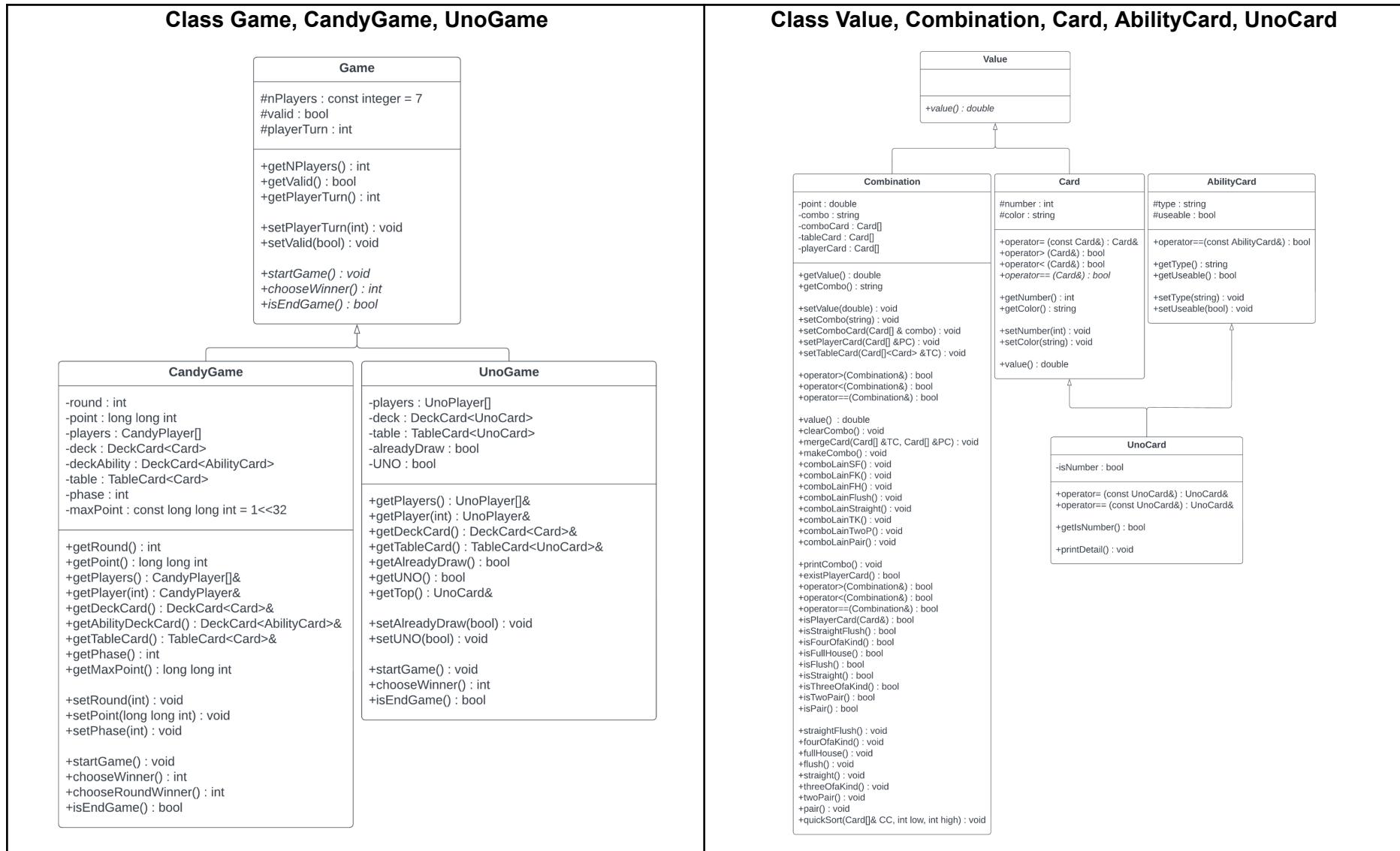
b. Diagram Kelas

Berdasarkan rancangan hierarki kelas yang telah dibuat, desain diagram kelas dapat dibuat. Pada diagram ini, terdapat rancangan atribut dan metode dari setiap kelas.

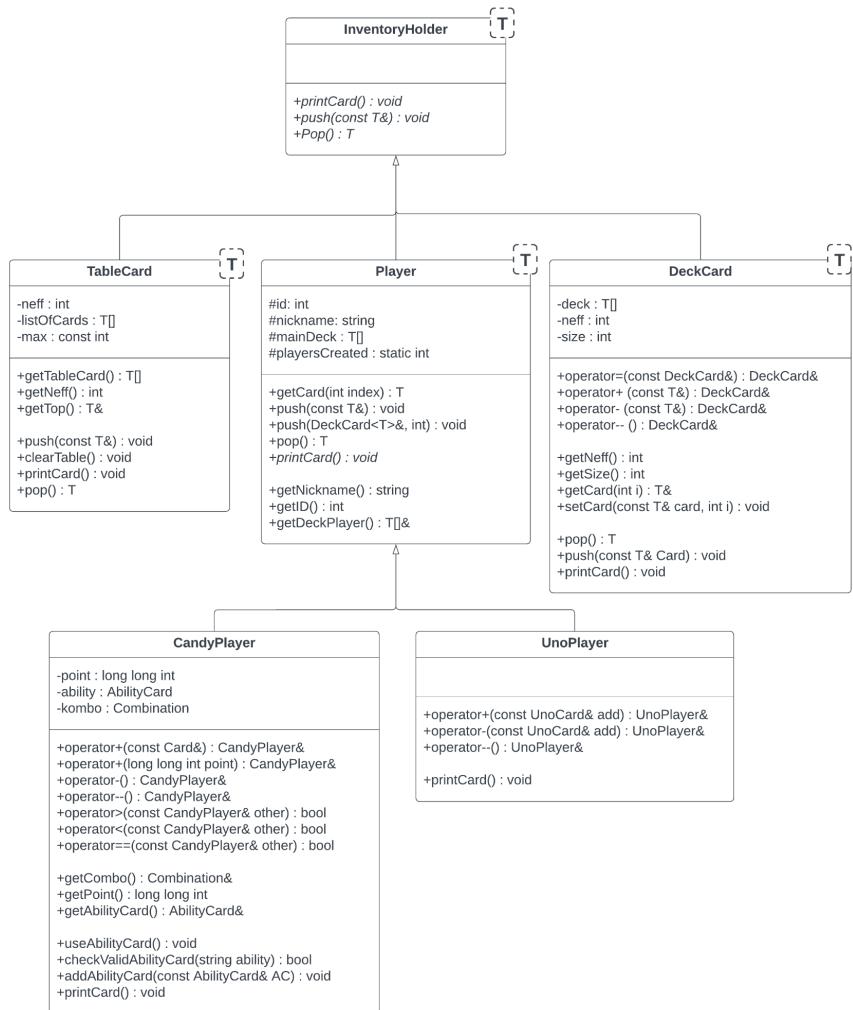
Adapun kelas-kelas yang terdefinisi pada permainan ini adalah sebagai berikut:

- | | | |
|--------------------|---------------------------|--------------------|
| 1. Game | 17. CommandInvalidExc | 33. Switch |
| 2. CandyGame | 18. InputNumberInvalidExc | 34. Abilityless |
| 3. UnoGame | 19. InputActionInvalidExc | 35. EndGame |
| 4. InventoryHolder | 20. FileInvalidExc | 36. Help |
| 5. TableCard | 21. ConfigInvalidExc | 37. CheckTableCard |
| 6. DeckCard | 22. CommandParser | 38. CheckCard |
| 7. Player | 23. CommandParserUNO | 39. PickCard |
| 8. CandyPlayer | 24. Command | 40. Draw |
| 9. UnoPlayer | 25. Next | 41. SayUno |
| 10. Value | 26. ReRoll | 42. Plus2 |
| 11. Combination | 27. Double | 43. Plus4 |
| 12. Card | 28. Quadruple | 44. Skip |
| 13. AbilityCard | 29. Half | 45. ChangeColor |
| 14. UnoCard | 30. Quarter | 46. UnoReverse |
| 15. CardGenerator | 31. Reverse | 47. Pass |
| 16. GameException | 32. SwapCard | 48. HelpUno |

Berikut adalah diagram kelas pada permainan Kompetisi Kartu ala Kerajaan Permen.



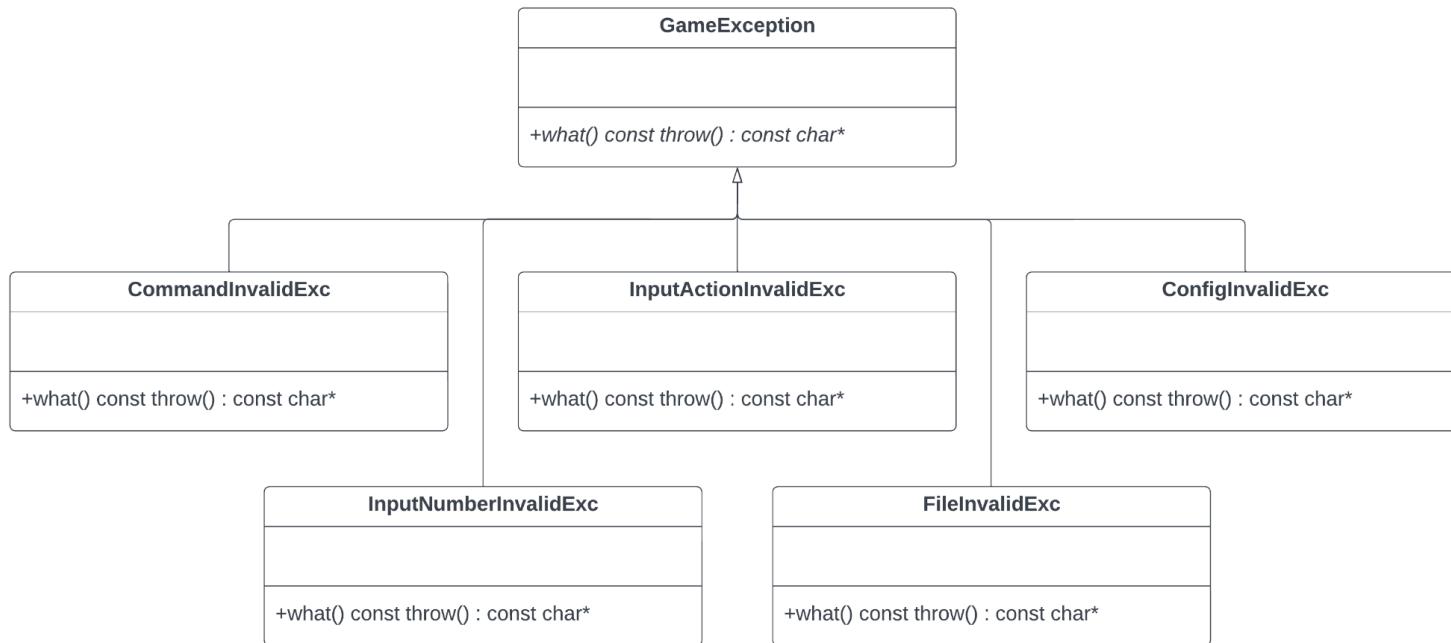
Class InventoryHolder, TableCard, DeckCard, Player, CandyPlayer, UnoPlayer



Class CardGenerator, CommandParser, CommandParserUNO



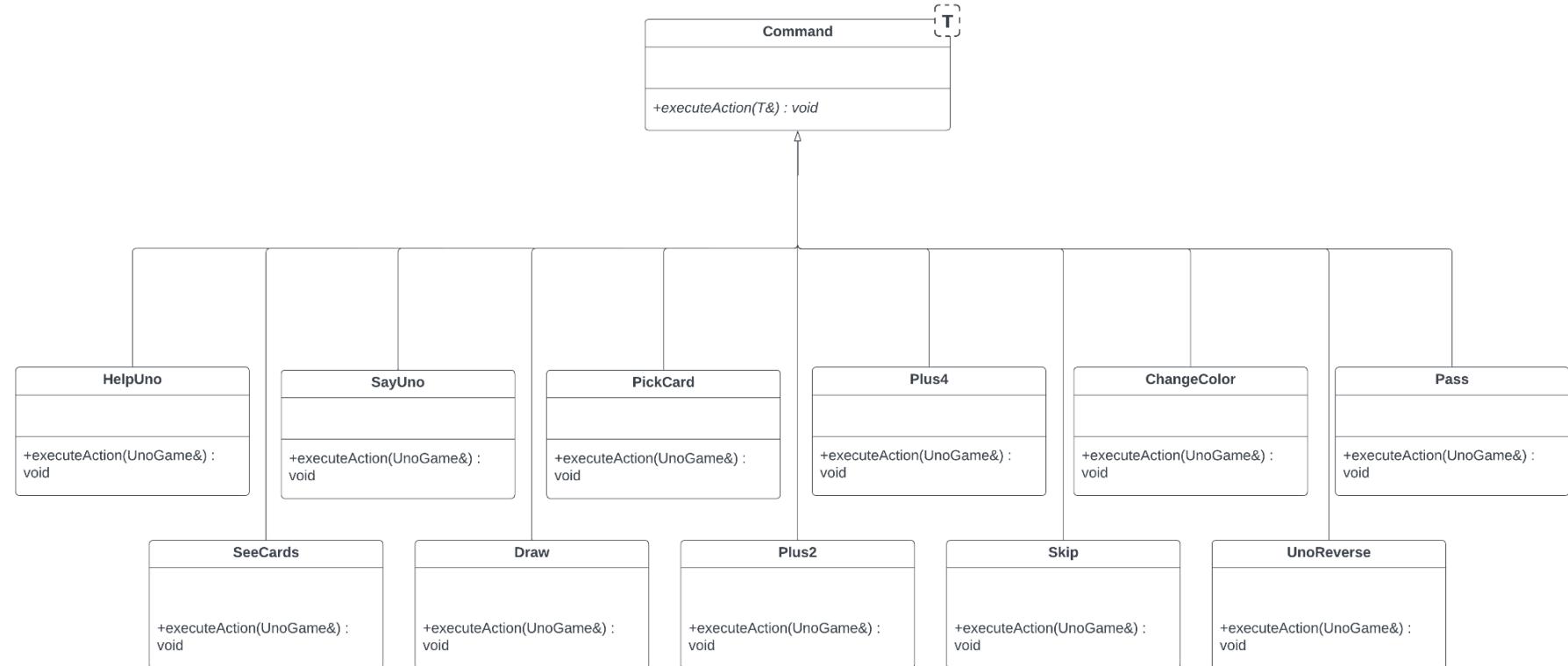
Class Exception



Class Command



Class Command UNO



c. Desain Kelas

Adapun detail dari rancangan diagram kelas dijelaskan sebagai berikut.

i. Game

Game merupakan kelas yang merepresentasikan permainan kartu. Kelas ini akan menjadi *parent* dari kelas CandyGame dan UnoGame. Atribut kelas Game merupakan data umum yang diperlukan oleh kedua permainan, yaitu jumlah pemain, giliran pemain, dan validasi permainan. Metode yang dienkapsulasi kelas ini berupa metode *getter*, *setter*, dan metode virtual startGame, chooseWinner, dan isEndGame. Metode-metode virtual tersebut akan didefinisikan pada kelas anaknya, yaitu CandyGame dan UnoGame.

ii. CandyGame

CandyGame merupakan kelas yang merepresentasikan permainan kartu Kerajaan Permen. Kelas ini merupakan kelas anak dari Game. Artinya, kelas CandyGame mewarisi seluruh atribut dan metode yang terdapat pada Game. Selain itu, kelas CandyGame juga memiliki atribut tambahan yang menyimpan data yang dibutuhkan selama permainan berlangsung, yaitu informasi tentang ronde, poin hadiah, list pemain, deck kartu, deck kartu *ability*, table card, banyaknya permainan yang sudah berlangsung (phase), dan poin hadiah maksimum. Adapun metode yang didefinisikan pada kelas ini berupa metode *getter* dan *setter* atribut tambahan, startGame, chooseWinner, chooseRoundWinner, dan isEndGame. Metode startGame merupakan metode yang akan menjalankan permainan kartu Kerajaan Permen dari awal hingga akhir permainan. Metode chooseWinner merupakan metode untuk memilih pemenang permainan, sedangkan metode chooseRoundWinner merupakan metode untuk memilih pemenang pada setiap ronde permainan. Metode isEndGame merupakan metode yang mendefinisikan kapan permainan berakhir, yaitu ketika poin hadiah sudah mencapai atau melebihi poin maksimum.

iii. UnoGame

UnoGame merupakan kelas yang merepresentasikan permainan kartu UNO. Kelas ini merupakan kelas anak dari Game, sehingga kelas ini mewarisi seluruh atribut dan method yang dienkapsulasi kelas Game. Selain itu, kelas UnoGame memiliki beberapa atribut tambahan, seperti list pemain, deck kartu Uno, table card, informasi apakah pemain telah *draw* kartu, dan informasi apakah pemain telah mengatakan "UNO". Kelas UnoGame juga memiliki metode lain, yaitu *getter* dan *setter* atribut tambahan, startGame, chooseWinner, dan isEndGame. Metode startGame merupakan metode yang akan menjalankan permainan kartu UNO dari awal hingga akhir permainan. Metode chooseWinner merupakan metode untuk memilih pemenang permainan, yaitu pemain pertama yang menghabiskan seluruh kartu pada deck-nya, sedangkan metode isEndGame mendefinisikan akhir permainan.

iv. **InventoryHolder**

InventoryHolder merupakan kelas yang merepresentasikan inventory berupa penempatan kartu sebagai *abstract base class* yang diturunkan oleh kelas TableCard, DeckCard, dan Player. Kelas ini diterapkan sebagai kelas generik yang digunakan untuk mengatur *inventory* dari kelas-kelas yang diturunkan beserta manipulasi dan menampilkan *inventory* tersebut dengan metode *pure virtual* push dan pop untuk penambahan dan pengurangan *inventory*, serta metode *pure virtual* printCard untuk menampilkan *inventory* baik table, deck, maupun player. Kelas-kelas yang memiliki *parent* kelas ini memiliki *inventory* masing-masing yang dapat menyimpan beberapa jenis kartu yaitu AbilityCard dan Card dari CandyGame karena sifat generik dari InventoryHolder.

v. **TableCard**

TableCard merupakan kelas yang merepresentasikan entitas yang ada pada meja permainan setiap saatnya. Kelas ini merupakan turunan dari *abstract base class* InventoryHolder karena menyimpan informasi kartu yang ada di meja. Kelas TableCard ini juga bersifat generik dikarenakan menyesuaikan jenis kartu untuk lebih dari satu jenis permainan.

vi. **DeckCard**

DeckCard merupakan kelas yang merepresentasikan deck kartu permainan, baik deck pada meja permainan maupun deck pemain. Kelas ini merupakan kelas generik turunan dari InventoryHolder. DeckCard menyimpan informasi kartu-kartu pada deck, baik kartu biasa, kartu *ability*, maupun kartu UNO, disesuaikan dengan jenis permainannya.

vii. **Player**

Player merupakan kelas yang merepresentasikan pemain yang terdaftar dalam permainan. Kelas ini merupakan kelas turunan dari InventoryHolder. Karena Player dapat digunakan dalam permainan kartu Kerajaan Permen dan permainan kartu UNO, maka kelas Player bertipe generik untuk jenis kartu yang disimpan pada deck pemain.

viii. **CandyPlayer**

CandyPlayer merupakan kelas turunan dari Player yang merepresentasikan pemain pada permainan kartu Kerajaan Permen. Jenis kartu yang disimpan pada deck pemain adalah kartu biasa (Card). CandyPlayer juga memiliki beberapa atribut tambahan yang relevan untuk permainan, seperti poin pemain, *ability*, serta kombo yang dimiliki oleh pemain.

ix. UnoPlayer

UnoPlayer merupakan kelas turunan dari Player yang merepresentasikan pemain pada permainan kartu UNO. Jenis kartu yang disimpan pada deck pemain adalah kartu UnoCard.

x. Value

Value merupakan *abstract base class*. Kelas ini merupakan kelas dasar untuk berbagai macam *value* yang dapat dieksekusi selama permainan, Candy Game. Kelas ini hanya mengenkapsulasi satu metode, yaitu value. Metode value merupakan metode *pure virtual* yang akan didefinisikan pada kelas anaknya.

xi. Combination

Combination merupakan kelas yang merepresentasikan combo yang dimiliki masing-masing pemain setiap akhir ronde ke-6. Kelas ini merupakan kelas turunan dari *abstract base class* Value karena menyimpan value dari combo setiap pemain. Kelas ini menyimpan point/value combo, nama combo, dan susunan combo card dari masing-masing pemain. Metode-metode yang ada pada kelas ini semuanya digunakan untuk membantu mencari combo-combo yang mungkin dari gabungan antara kartu pemain dan kartu table, beberapa metode tersebut adalah value, makeCombo, mergeCard, isStraightFlush, dan lain - lain. Untuk perhitungan value/point dari combo digunakan beberapa rumus berikut untuk tiap combonya:

1. High Card

```
point = konstanta + warna * 0.03  
konstanta: angka_kartu * 0.1  
warna: Hijau = 0, Biru = 1, Kuning = 2, Merah = 3
```

2. Pair

$point = konstanta + (kombinasi\ warna * 0.015) + 1.39$
 $konstanta: angka_kartu * 0.1$

kombinasi warna: Hijau Biru = 1, Hijau Kuning = 2, Biru Kuning = 3, Hijau Merah = 4, Biru Merah = 5, Kuning Merah = 6

3. Two Pair

$point = pair\ tertinggi + 2.78$
 $pair\ tertinggi = konstanta + (kombinasi\ warna * 0.015) + 1.39$
 $pair\ tertinggi = harus\ terdapat\ kontribusi\ kartu\ player\ pada\ pair\ tersebut$
 $konstanta: angka_kartu * 0.1$

kombinasi warna: Hijau Biru = 1, Hijau Kuning = 2, Biru Kuning = 3, Hijau Merah = 4, Biru Merah = 5, Kuning Merah = 6

4. Three of a Kind

$point = konstanta + (kombinasi\ warna * 0.02) + 5.56$
 $konstanta: angka_kartu * 0.1$

kombinasi warna : Hijau Biru Kuning = 1, Hijau Biru Merah = 2, Hijau Kuning Merah = 3, Biru Kuning Merah = 4

5. Straight

$point = highcard + 6.94$

highcard : point dari kartu player tertinggi yang berkontribusi dalam membuat combo straight (menggunakan rumus High Card)

6. Flush

$point = highcard + 8.33$

highcard : point dari kartu player tertinggi yang berkontribusi dalam membuat combo flush (menggunakan rumus High Card)

7. Full House

$point = highcard + 9.72$

highcard : point dari kartu player tertinggi yang berkontribusi dalam membuat combo full house (menggunakan rumus High Card)

8. Four of a Kind

$$\text{point} = \text{highcard} + 11.11$$

highcard : point dari kartu player tertinggi yang berkontribusi dalam membuat combo four of a kind (menggunakan rumus High Card)

9. Straight Flush

$$\text{point} = \text{highcard} + 12.5$$

highcard : point dari kartu player tertinggi yang berkontribusi dalam membuat combo straight flush (menggunakan rumus High Card)

xii. Card

Card merupakan kelas yang merepresentasikan kartu pada permainan kartu Kerajaan Permen. Kelas ini merupakan turunan dari *abstract base class* Value karena dapat mengembalikan value dari kartu. Kelas ini menyimpan nomor angka dan warna dari masing-masing kartu. Metode-metode yang ada pada kelas ini terdapat getter, setter, operator overloading, dan value.

xiii. AbilityCard

AbilityCard merupakan kelas yang merepresentasikan kartu *ability*. Kelas ini menyimpan informasi berupa tipe *ability* dan informasi apakah kartu tersebut dapat digunakan.

xiv. UnoCard

UnoCard merupakan kelas yang merepresentasikan kartu UNO. kelas UnoCard merupakan turunan dari kelas Card dan AbilityCard karena pada permainan UNO, kartu dapat berupa kartu angka berwarna, kartu *ability* berwarna, atau kartu *ability* tanpa warna. Oleh karena itu, kelas UnoCard menyimpan atribut *isNumber* sebagai informasi yang menandakan apakah kartu UNO merupakan kartu angka.

xv. CardGenerator

CardGenerator merupakan kelas pembangkit kartu permainan. Kelas ini berisi metode-metode untuk membaca *file* konfigurasi kartu, *randomizer* kartu deck, dan membangkitkan deck kartu *ability*.

xvi. GameException

GameException merupakan kelas yang merepresentasikan exception yang didefinisikan untuk permainan. Kelas ini merupakan kelas *parent* dari beberapa kelas jenis-jenis exception yang mungkin terjadi pada permainan.

xvii. CommandInvalidExc

CommandInvalidExc merupakan kelas turunan dari GameException. Kelas ini berisi metode yang dapat melakukan *throw* pesan *error* ketika pemain memasukkan perintah yang tidak valid.

xviii. InputNumberInvalidExc

InputNumberInvalidExc merupakan kelas turunan dari GameException. Kelas ini berisi metode yang dapat melakukan *throw* pesan *error* ketika pemain memasukkan angka yang tidak valid.

xix. InputActionInvalidExc

InputActionInvalidExc merupakan kelas turunan dari GameException. Kelas ini berisi metode yang dapat melakukan *throw* pesan *error* ketika pemain melakukan aksi yang tidak valid.

xx. FileInvalidExc

FileInvalidExc merupakan kelas turunan dari GameException. Kelas ini berisi metode yang dapat melakukan *throw* pesan *error* ketika pemain memasukkan masukan nama *file* yang tidak valid.

xi. ConfigInvalidExc

ConfigInvalidExc merupakan kelas turunan dari GameException. Kelas ini berisi metode yang dapat melakukan *throw* pesan *error* ketika pemain memasukkan *file* konfigurasi yang tidak valid.

xxii. CommandParser

CommandParser merupakan kelas yang berisi metode untuk *parsing* perintah yang dimasukkan oleh pemain dalam permainan Candy Game. Kelas ini hanya memiliki satu buah metode, yaitu parse dengan parameter string masukan perintah dari pemain dan mengembalikan Command sesuai dengan perintah yang dimasukkan oleh pemain.

xxiii. CommandParserUNO

CommandParserUNO merupakan kelas yang berisi metode untuk *parsing* perintah yang dimasukkan oleh pemain dalam permainan UNO Game. Kelas ini hanya memiliki satu buah metode, yaitu parse dengan parameter string masukan perintah dari pemain dan mengembalikan Command sesuai dengan perintah yang dimasukkan oleh pemain.

xxiv. Command

Command merupakan *abstract base class* berupa kelas generik. Kelas ini merupakan kelas dasar untuk berbagai macam *commands* yang dapat dieksekusi selama permainan, baik Candy Game maupun UNO Game. Kelas ini hanya mengenkapsulasi satu metode, yaitu executeAction dengan parameter bertipe generik yang dapat didefinisikan sebagai CandyGame atau UnoGame. Metode executeAction merupakan metode *pure virtual* yang akan didefinisikan pada setiap kelas anaknya.

xxv. Next

Next merupakan kelas yang merepresentasikan perintah NEXT. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah NEXT yang dimasukkan oleh pemain, yaitu mengganti giliran pemain menjadi pemain berikutnya.

xxvi. ReRoll

ReRoll merupakan kelas yang merepresentasikan perintah RE-ROLL. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah RE-ROLL yang dimasukkan oleh pemain, yaitu menggantikan dua kartu pemain dengan dua kartu baru yang diambil dari *deck*.

xxvii. Double

Double merupakan kelas yang merepresentasikan perintah DOUBLE. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah DOUBLE yang dimasukkan oleh pemain, yaitu mengandakan poin hadiah.

xxviii. Quadruple

Quadruple merupakan kelas yang merepresentasikan perintah QUADRUPLE. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah QUADRUPLE yang dimasukkan oleh pemain, yaitu mengalikan poin hadiah sebanyak empat kali.

xxix. Half

Half merupakan kelas yang merepresentasikan perintah HALF. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah HALF yang dimasukkan oleh pemain, yaitu menurunkan poin hadiah menjadi setengah kali dari sebelumnya.

xxx. Quarter

Quarter merupakan kelas yang merepresentasikan perintah QUARTER. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah QUARTER yang dimasukkan oleh pemain, yaitu menurunkan poin hadiah menjadi satu perempat kali dari sebelumnya.

xxxi. Reverse

Reverse merupakan kelas yang merepresentasikan perintah REVERSE. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah REVERSE yang dimasukkan oleh pemain, yaitu membalikkan turn untuk turn selanjutnya dan untuk pemain-pemain yang belum melakukan aksi pada turn tersebut.

xxxii. SwapCard

SwapCard merupakan kelas yang merepresentasikan perintah SWAPCARD. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah SWAPCARD yang dimasukkan oleh pemain, yaitu menukar masing-masing satu kartu dari dua pemain selain dirinya sendiri.

xxxiii. Switch

Switch merupakan kelas yang merepresentasikan perintah SWITCH. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah SWITCH yang dimasukkan oleh pemain, yaitu menukar 2 kartu diri sendiri dengan 2 kartu pemain lainnya.

xxxiv. Abilityless

Abilityless merupakan kelas yang merepresentasikan perintah ABILITYLESS. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah ABILITYLESS yang dimasukkan oleh pemain, yaitu mematikan *ability* yang dimiliki oleh pemain lain.

xxxv. EndGame

EndGame merupakan kelas yang merepresentasikan perintah ketika permainan berakhir. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah yang terjadi ketika permainan berakhir, yaitu menampilkan *leaderboard* dan menentukan pemenang permainan.

xxxvi. Help

Help merupakan kelas yang merepresentasikan bantuan untuk pemain pada permainan CandyGame. Kelas ini merupakan *inheritance* dari *abstract class* Command karena merupakan salah satu command yang dapat digunakan oleh pemain. Kelas HelpUno digunakan untuk memberikan list-list command apa saja yang dapat pemain panggil saat bermain permainan CandyGame.

xxxvii. CheckTableCard

CheckTableCard merupakan kelas yang merepresentasikan perintah CHECKTABLECARD. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah CHECKTABLECARD yang dimasukkan oleh pemain, yaitu menampilkan table card pada permainan.

xxxviii. CheckCard

CheckCard merupakan kelas yang merepresentasikan perintah CHECKCARD. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe CandyGame. Metode executeAction akan mengeksekusi perintah CHECKCARD yang dimasukkan oleh pemain, yaitu menampilkan card yang dimiliki pemain.

xxxix. PickCard

PickCard merupakan kelas yang merepresentasikan perintah PICKCARD. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah PICKCARD yang dimasukkan oleh pemain, yaitu memilih kartu yang berada pada deck pemain untuk ditaruh di atas table card.

xl. Draw

Draw merupakan kelas yang merepresentasikan perintah DRAW. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah DRAW yang dimasukkan oleh pemain, yaitu mengambil satu kartu dari deck kartu permainan.

xli. SayUno

SayUno merupakan kelas yang merepresentasikan perintah SAYUNO. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah SAYUNO yang dimasukkan oleh pemain, yaitu mengatakan kata “UNO” ketika pemain tersisa sebuah kartu agar tidak terkena penalti *draw* dua kartu (apabila tidak mengatakan “UNO”).

xlii. Plus2

Plus2 merupakan kelas yang merepresentasikan perintah PLUS2. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah PLUS2 yang dimasukkan oleh pemain, yaitu pemain selanjutnya akan mendapatkan 2 kartu tambahan dari deck card.

xliii. Plus4

Plus4 merupakan kelas yang merepresentasikan perintah PLUS4. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah PLUS4 yang dimasukkan oleh pemain, yaitu pemain selanjutnya akan mendapatkan 4 kartu tambahan dari deck card dan memilih warna kartu baru untuk turn selanjutnya.

xliv. Skip

Skip merupakan kelas yang merepresentasikan perintah SKIP. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah SKIP yang

dimasukkan oleh pemain, yaitu pemain selanjutnya akan dilompati gilirannya sehingga giliran langsung didapatkan oleh 2 pemain selanjutnya.

xlv. ChangeColor

ChangeColor merupakan kelas yang merepresentasikan perintah CHANGECOLOR. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah CHANGECOLOR, yaitu mengubah warna kartu sesuai keinginan pemain ketika pemain tersebut mengeluarkan kartu change color atau kartu plus 4.

xvi. UnoReverse

UnoReverse merupakan kelas yang merepresentasikan perintah UNOREVERSE. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah UNOREVERSE yang dimasukkan oleh pemain, yaitu mengubah arah urutan permainan (misalnya dari searah jarum jam menjadi berlawanan arah jarum jam, atau sebaliknya).

xvii. Pass

Pass merupakan kelas yang merepresentasikan perintah PASS. Kelas ini merupakan kelas anak dari Command. Metode executeAction didefinisikan dengan parameter bertipe UnoGame. Metode executeAction akan mengeksekusi perintah PASS yang dimasukkan oleh pemain, yaitu pemain yang memasukkan perintah pass akan melewati turn-nya dengan syarat telah melakukan draw terlebih dahulu.

xviii. HelpUno

HelpUno merupakan kelas yang merepresentasikan bantuan untuk pemain pada permainan UNO. Kelas ini merupakan inheritance dari abstract class Command karena merupakan salah satu command yang dapat digunakan oleh pemain. Kelas HelpUno digunakan untuk memberikan list-list command apa saja yang dapat pemain panggil saat bermain permainan UNO.

Alasan kami memilih desain OOP yang sudah kami buat adalah kami memaksimumkan efisiensi pembuatan game kartu secara general. Berdasarkan hasil analisis dari kelompok kami, sangat mudah untuk membuat game kartu baru berdasarkan desain OOP yang telah dibuat. Pertama, kami membuat deck card secara generic sehingga kartu yang bisa ditampung oleh suatu deck card bisa berbagai jenis.

Kedua, kami membuat base class berupa game yang memiliki atribut-atribut dan metode-metode standar yang biasa terdapat pada permainan kartu sehingga untuk membuat game baru cukup melakukan ekstensi dari kelas game. Ketiga, kelas table card juga kami buat secara generic untuk memudahkan peletakan berbagai jenis kartu sehingga dapat digunakan untuk berbagai permainan kartu. Keempat, kami juga membuat base class player yang memiliki atribut dan metode standar yang biasa dimiliki oleh suatu player dalam permainan sehingga untuk membuat jenis pemain baru untuk permainan lain, bisa langsung ekstensi dari base class player. Player juga dibuat secara generic sehingga player dapat memegang berbagai jenis kartu yang diperlukan dalam permainan yang sedang dimainkannya. Kelima, terdapat kelas abstrak command untuk melakukan berbagai command yang diperlukan oleh permainan yang telah dibuat. Command dibuat secara generic juga sehingga bisa dispesialisasikan untuk suatu game saja. Dari kelima kelebihan ini, untuk membuat suatu permainan kartu baru sudah terarah sehingga jauh lebih mudah.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep *inheritance* dan *polymorphism* digunakan di banyak kelas pada program kami. Kelas yang menggunakan *inheritance* dan *polymorphism* adalah InventoryHolder dan anak-anaknya (TableCard, DeckCard, Player), Player dan anak-anaknya (CandyPlayer, UnoPlayer), Game (CandyGame, UnoGame), Value dan anak-anaknya (Combination, Card), Command dan anak-anaknya, dan GameException beserta anak-anaknya. Kami juga menerapkan konsep *multiple inheritance* pada *derived class* UnoCard yang memiliki *parent* Card dan AbilityCard. Hal ini disebabkan oleh UnoCard yang dapat berupa kartu biasa dan kartu *ability*. Konsep *inheritance* sangat diperlukan dalam program ini untuk membuat banyak kelas yang memiliki beberapa sifat/atribut yang sama sehingga mempermudah pembuatan dan mengurangi pengulangan kode. Konsep *polymorphism* juga digunakan untuk mempermudah pengolahan pada objek dengan ekstensi dari tipe kelas *parent*-nya. Kedua konsep ini sangat terpakai pada kelas-kelas yang telah disebutkan di atas untuk mempermudah pengolahan berbagai jenis objek yang memiliki beberapa sifat yang sama. Contohnya adalah seperti kelas GameException yang merupakan *abstract base class* dan memiliki tipe kelas anak yang beragam, dimana *polymorphism* diterapkan untuk mengimplementasikan berbagai jenis *throw* sehingga *exception* bisa *di-catch* oleh *parent*-nya saja dan bisa mengeluarkan *error* yang *di-throw*. Kelas InventoryHolder yang merupakan *abstract base class* juga memiliki tiga tipe anak yang sangat sering dipakai dalam sebuah permainan kartu dan masing-masing memiliki metode yang sama namun berbeda cara implementasi yaitu menambah dan membuang kartu.

```
1 #ifndef COMMAND_HPP
2 #define COMMAND_HPP
3
4 #include <iostream>
5 #include "../Game/candyGame.hpp"
6 #include "../Inventory/children/DeckCard/deckCard.hpp"
7 #include "../Exception/exception.h"
8
9 using namespace std;
10
11 /* Abstract Base Class Command */
12 template <class T>
13 class Command {
14     public:
15         /* Dtor */
16         ~Command(){};
17
18     /* Pure virtual method */
19     virtual void executeAction(T&) = 0; // Execute action/command/ability
20 };
21
22 #endif
```

```
1 #ifndef GAME_EXCEPTION_H
2 #define GAME_EXCEPTION_H
3
4 #include <stdexcept>
5 #include <iostream>
6 using namespace std;
7
8 /* Exception Parent */
9 class GameException : public exception{
10     public:
11         /* Pure virtual method */
12         virtual const char* what() const throw()=0;
13 };
14
15 /* Exception Children */
16 class CommandInvalidExc : public GameException{
17     public:
18         const char* what() const throw(); // Throw if command invalid
19 };
20
21 class InputNumberInvalidExc : public GameException{
22     public:
23         const char* what() const throw(); // Throw if number invalid
24 };
25
26 class InputActionInvalidExc : public GameException{
27     public:
28         const char* what() const throw(); // Throw if action invalid
29 };
30
31 class FileInvalidExc : public GameException{
32     public:
33         const char* what() const throw(); // Throw if file not found
34 };
35
36 class ConfigInvalidExc : public GameException{
37     public:
38         const char* what() const throw(); // Throw if config file format invalid
39 };
40
41 #endif
```

```
1 #ifndef INVENTORY HOLDER_HPP
2 #define INVENTORY HOLDER_HPP
3 #include "../Value/Card/abilityCard.hpp"
4 #include "../Value/Card/children/unoCard.hpp"
5 #include "../Value/Card/card.hpp"
6 #include <iostream>
7 using namespace std;
8
9 /* Abstract Base Class */
10 template <class T>
11 class InventoryHolder {
12     public:
13         /* Pure Virtual Methods */
14         virtual void printCard() = 0;
15         virtual void push(const T&) = 0;
16         virtual T pop() = 0;
17 };
18
19 template class InventoryHolder<Card>;
20 template class InventoryHolder<AbilityCard>;
21 template class InventoryHolder<UnoCard>;
22
23 #endif
```

```
1 #ifndef TABLE_CARD_HPP
2 #define TABLE_CARD_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../Value/Card/card.hpp"
7 #include "../DeckCard/deckCard.hpp"
8
9 using namespace std;
10
11 template <class T>
12 class TableCard : public InventoryHolder<T> {
13     private:
14         int neff; /* Determine how many cards on table */
15         vector<T> listOfCards; /* List of cards */
16         const int max; /* Determine the maximum amount of cards on table */
17
18     public:
19         /* Ctor */
20         TableCard(int max);
21         TableCard(const TableCard& other);
22
23         /* Getter */
24         vector<T>& getTableCard();
25         int getNeff();
26         T& getTop();
27
28         /* Other methods */
29         void push(const T& C);
30         void clearTable();
31         void printCard();
32         T pop();
33     };
34
35 template class TableCard<Card>;
36 template class TableCard<UnoCard>;
37
38 #endif
```

```

1 #ifndef PLAYER_HPP
2 #define PLAYER_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../..../inventoryHolder.hpp"
7 #include "../..../Value/Card/deckCard.hpp"
8 #include "../..../Value/Card/abilityCard.hpp"
9 #include "../..../Value/Combination/combination.hpp"
10
11 using namespace std;
12
13 template <class T>
14 class Player: public InventoryHolder<T>{
15     protected:
16         int id; /* Player's ID */
17         string nickname; /* Player's nickname */
18         vector<T> mainDeck; /* Player's deck card */
19         static int playersCreated; /* Determine how many players have been created */
20
21     public:
22         /* Ctor */
23         Player();
24         Player(string nickname);
25
26         /* Other Methods */
27         T getCard(int index);
28         void push(const T& PC);
29         void push(DeckCard<T>& DC, int count);
30         T pop();
31         virtual void printCard() = 0;
32
33         /* Getter */
34         string getNickname();
35         int getID();
36         vector<T>& getDeckPlayer();
37     };
38
39 template class Player<Card>;
40 template class Player<AbilityCard>;
41 template class Player<UnoCard>;
42
43 #endif

```

```

1 #ifndef DECK_CARD_HPP
2 #define DECK_CARD_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../..../inventoryHolder.hpp"
7 #include "../..../Value/Card/card.hpp"
8 #include "../..../Value/Card/children/unoCard.hpp"
9 #include "../..../Value/Card/abilityCard.hpp"
10
11 using namespace std;
12
13 template <class T>
14 class DeckCard: public InventoryHolder<T>{
15     private:
16         vector<T> deck; /* List of cards */
17         int neff; /* Determine how many cards are currently in the deck */
18         int size; /* Determine the maximum size of the deck */
19
20     public:
21         /* Ctor */
22         DeckCard();
23         DeckCard(int neff, int size);
24
25         /* Operator overloading */
26         DeckCard& operator=(const DeckCard& dc);
27         DeckCard& operator+(const T& add);
28         DeckCard& operator-(const T& min);
29         DeckCard& operator--();
30
31         /* Getter */
32         int getNeff();
33         int getSize();
34         T& getCard(int i);
35
36         /* Setter */
37         void setCard(const T &card, int i);
38
39         /* Other methods */
40         T pop();
41         void push(const T& Card);
42         void printCard();
43     };
44
45 #endif

```

2.2. Method/Operator Overloading

Konsep *operator overloading* digunakan di beberapa kelas pada program yaitu kelas Player, DeckCard, dan Combination. Untuk operator yang mengembalikan boolean, umumnya digunakan dalam sebuah metode yang memerlukan perbandingan atau fungsi generik yang telah dibuat, misalnya untuk mencari nilai maksimum. Metode yang memerlukan perbandingan bisa seperti membandingkan kartu untuk menentukan kartu mana yang *value*-nya lebih tinggi, membandingkan Player untuk menentukan Player mana yang poinnya lebih tinggi, dan membandingkan kombo mana yang hasil perhitungannya lebih tinggi. Atau pada permainan UNO, digunakan juga operator== untuk menentukan apakah kartu tersebut dapat diletakkan pada table card atau tidak. Untuk operator yang mengembalikan *reference* terhadap dirinya, seperti + dan -, digunakan untuk penambahan atau pengurangan suatu elemen. Contohnya, pada Player dan DeckCard, keduanya sama-sama memegang kartu sehingga diperlukan operator + dan - untuk menarik dan menambahkan kartu. Dengan penggunaan *operator overloading*, penulisan dapat lebih mudah dimengerti.

Konsep method overloading digunakan di kelas. Salah satu method yang memanfaatkan konsep ini adalah push. Push sendiri sering dilakukan oleh Player mengingat sering terjadinya pertambahan kartu pada kelas tersebut. Maka dari itu, untuk menghindari repetitif penulisan kode untuk penambahan kartu pada player, digunakan method overloading dengan argumen DeckCard dan count (banyaknya kartu yang ingin ditambahkan ke player) sehingga jika kita ingin menambahkan beberapa kartu sekaligus dalam suatu aksi, kita cukup memanggil method overloading ini. Kami juga mengimplementasikan method overloading pada operator overloading. Contohnya pada CandyPlayer, pertambahan yang dilakukan ke CandyPlayer tidak hanya berupa objek kartu saja, tetapi juga point dari game. Oleh karena itu, operator + memiliki dua signature yang berbeda, yang pertama untuk penambahan kartu dan yang kedua untuk penambahan point.

```
1 /* Generic function to find the max value*/
2 template <class T>
3 T maxValue(vector<T>& x){
4     T m = x[0];
5     for (T i : x){
6         if (i > m){
7             m = i;
8         }
9     }
10    return m;
11 }
```

```
1 // Restart Game Player Card
2 for(int i = 0; i < 7; i++){
3     this->players[i] = this->players[i] + this->deck.pop();
4     this->players[i] = this->players[i] + this->deck.pop();
5 }
```

```
1 // Push randomize cards to deck card
2 for(int i = 0 ; i < 52; i++){
3     DC = DC + randomize[i];
4 }
```

```
1 // Restart Game
2     this->round = 0;
3     for(int i = 0; i < 7; i++){
4         this->players[i] = this->players[i] - this->players[i].getDeckPlayer().back();
5         this->players[i] = this->players[i] - this->players[i].getDeckPlayer().back();
6         this->players[i].getCombo().clearCombo();
7         this->players[i].getAbilityCard().setType("");
8         this->players[i].getAbilityCard().setUsable(false);
9     }
10    this->table.clearTable();
```

```
1 // Buang Kartu Dari Deck
2 while(this->deck.getNeff() != 0){
3     this->deck = this->deck - this->deck.getCard(0);
4 }
```

```

1 bool Combination::operator<(Combination& other) {
2     return this->point < other.point;
3 }
4
5 bool Combination::operator>(Combination& other) {
6     return this->point > other.point;
7 }
8
9 bool Combination::operator==(Combination& other) {
10    return this->point == other.point;
11 }

```

```

1 bool Card::operator>(Card& c){
2     return this->value() > c.value();
3 }
4
5 bool Card::operator<(Card& c){
6     return this->value() < c.value();
7 }
8
9 bool Card::operator==(const Card& c){
10    return (this->number == c.number && this->color == c.color);
11 }

```

```

1 bool CandyPlayer::operator>(const CandyPlayer& other){
2     return this->point > other.point;
3 }
4
5 bool CandyPlayer::operator<(const CandyPlayer& other){
6     return this->point < other.point;
7 }
8
9 bool CandyPlayer::operator==(const CandyPlayer& other){
10    return (this->point == other.point);
11 }

```

```

1 if (!(Game.getTop() == SCard)){
2     cout << "\nKartu tidak bisa dipakai. Silahkan pilih kartu lain!" << endl;
3 } else {
4     validCard = true;
5 }

```

```

1 CandyPlayer::CandyPlayer(DeckCard<Card>& DC, string nickname): Player(nickname){
2     this->point = 0;
3     this->push(DC, 2);
4 }

```

```

1 /* Choose round winner and print */
2     int roundWinner = this->chooseRoundWinner();
3     this->players[roundWinner] = this->players[roundWinner] + this->point;

```

```

1 /* Operators */
2 CandyPlayer& CandyPlayer::operator+(const Card& add){
3     this->push(add);
4     return *this;
5 }
6
7 CandyPlayer& CandyPlayer::operator+(long long int point){
8     this->point += point;
9     return *this;
10 }

```

```
 1 template <class T>
 2 void Player<T>::push(const T& PC){
 3     this->mainDeck.push_back(PC);
 4 }
 5
 6 template <class T>
 7 void Player<T>::push(DeckCard<T>& DC, int count){
 8     if(DC.getNeff() >= count){
 9         for(int i = 0; i < count; i++){
10             this->mainDeck.push_back(DC.pop());
11         }
12     }
13 }
```

2.3. Template & Generic Functions

Konsep *template* dan *generic functions* diimplementasikan pada dua fungsi yang dibuat untuk keperluan program. Dengan adanya *template generic function* ini, kita tidak perlu mengimplementasikan ulang algoritma yang sama untuk tipe yang berbeda. Untuk fungsi pertama, digunakan untuk mencari nilai tertinggi dari tipe data tersebut. Tentu harus didefinisikan juga “nilai” dari tipe data tersebut apa, misalnya nilai dari Player berarti point dari Player tersebut. Kasus ini sudah diatasi dengan digunakannya *operator overloading* untuk perbandingan yang diperlukan. Untuk fungsi kedua, digunakan untuk mencari indeks dari nilai tertinggi tersebut. Contoh penggunaannya adalah untuk mencari kombo High Card juga pemenang dari game dan terdapat pada kode di bawah untuk lebih lanjut.

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 /* Generic function to find the max value*/
6 template <class T>
7 T maxValue(vector<T>& x){
8     T m = x[0];
9     for (T i : x){
10         if (i > m){
11             m = i;
12         }
13     }
14     return m;
15 }
16
17 /* Generic function to find the index of the max value */
18 template <class T>
19 int findIndexMaxValue(vector<T>& x){
20     T maximum = x[0];
21     int idx = 0;
22     for(int i = 1; i < x.size(); i++){
23         if(x[i] > maximum){
24             maximum = x[i];
25             idx = i;
26         }
27     }
28     return idx;
29 }

```



```

1 void Combination::comboLainPair() {
//Mengatur point/value, nama, dan isi dari combo
lain jika ternyata combo pair sebelumnya berasal
dari kartu meja seluruhnya
2     Card highest = maxValue(this->playerCard);
3     this->comboCard.clear();
4     this->comboCard.push_back(highest);
5     this->setValue(this->value());
6     this->setCombo("High Card");
7 }

```



```

1 /* Choose index of the round winner */
2 int CandyGame::chooseRoundWinner() {
3     vector<Combination> combos;
4     for(int i = 0; i < 7; i++){
5         combos.push_back(this->players[i].getCombo());
6     }
7     return findIndexMaxValue(combos);
8 }

```

2.4. Exception

Konsep exception digunakan untuk mengatasi error yang mungkin disebabkan pada metode suatu kelas maupun pada main program. Terdapat lima kasus yang kami definisikan sebagai *error* yaitu *command invalid* (command yang tidak terdefinisi), *input number invalid* (pilihan angka yang tidak tersedia), *input action invalid* (untuk mengatasi input yang tidak sesuai dengan yang diminta seperti mengisi string pada variabel integer), *file invalid* (file tidak ditemukan), dan *config invalid* (konfigurasi file tidak sesuai format yang ditentukan). Oleh karena itu, untuk setiap kelas didefinisikan **exception** untuk melakukan *throw* deskripsi dari *error*-nya. Maka, dibuatkan *abstract base class* untuk *exception handling* yaitu GameException dan diterapkan *polymorphism* untuk masing-masing anaknya. Kelebihan dari implementasi ini adalah ketika

melakukan *catching*, program tidak perlu mendefinisikan *catch* untuk setiap jenis *error*, sehingga pada parameter *catch*, cukup menggunakan *parent*-nya, yaitu GameException yang akan memanggil *throw* dari anaknya (dengan konsep virtual).

```

1 #ifndef GAME_EXCEPTION_H
2 #define GAME_EXCEPTION_H
3
4 #include <stdexcept>
5 #include <iostream>
6 using namespace std;
7
8 /* Exception Parent */
9 class GameException : public exception{
10 public:
11     /* Pure virtual method */
12     virtual const char* what() const throw()=0;
13 };
14
15 /* Exception Children */
16 class CommandInvalidExc : public GameException{
17 public:
18     const char* what() const throw(); // Throw if command invalid
19 };
20
21 class InputNumberInvalidExc : public GameException{
22 public:
23     const char* what() const throw(); // Throw if number invalid
24 };
25
26 class InputActionInvalidExc : public GameException{
27 public:
28     const char* what() const throw(); // Throw if action invalid
29 };
30
31 class FileInvalidExc : public GameException{
32 public:
33     const char* what() const throw(); // Throw if file not found
34 };
35
36 class ConfigInvalidExc : public GameException{
37 public:
38     const char* what() const throw(); // Throw if config file format invalid
39 };
40
41 #endif

```

```

1 #include "..../headers/Exception/exception.h"
2
3 /* GameException */
4
5 /* Command Invalid Exception */
6 const char* CommandInvalidExc::what() const throw(){
7     return "Command tidak valid!\n";
8 }
9
10 /* File Not Found Invalid Exception */
11 const char* FileInvalidExc::what() const throw(){
12     return "File tidak ditemukan!";
13 }
14
15 /* Input File Invalid Exception */
16 const char* ConfigInvalidExc::what() const throw(){
17     return "File config tidak sesuai format!";
18 }
19
20 /* Input Number Invalid Exception */
21 const char* InputNumberInvalidExc::what() const throw(){
22     return "Angka yang dimasukkan tidak valid!\n";
23 }
24
25 /* Input Action Invalid Exception */
26 const char* InputActionInvalidExc::what() const throw(){
27     return "Input tidak valid!\n";
28 }

```

```

1 /* Read config or generate order cards */
2     while(action != "y" && action != "n" && action != "no" && action != "yes"){
3         try {
4             cout << "\nApakah urutan kartu ingin dibaca dari file? (y or n) : ";
5             cin >> action;
6
7             if(action == "y" || action == "yes"){
8                 bool validDeck = false;
9                 while(!validDeck){
10                     try{
11                         string pathfile = "./config/";
12                         string inputPath;
13                         cout << "\nMasukkan nama file konfigurasi: ";
14                         cin >> inputPath;
15                         pathfile += inputPath + ".txt";
16                         this->deck = CG.readFile(pathfile);
17                         validDeck = true;
18                     } catch(GameException& err2){
19                         cout << err2.what() << endl;
20                     }
21                 }
22             } else if (action == "n" || action == "no"){
23                 this->deck = CG.randomizeCard();
24             } else {
25                 throw InputActionInvalidExc();
26             }
27         } catch (GameException& err){
28             cout << err.what() << endl;
29         }
30     }

```

```

1 /* Read order cards .txt file */
2 DeckCard<Card> CardGenerator::readFile(string pathfile){
3     // Initialize variables and object
4     string line;
5     ifstream Read(pathfile);
6     DeckCard<Card> DC;
7
8     // Throw exception if read file failed
9     if(!Read.good()){
10         throw FileInvalidExc();
11     }
12
13     // Read .txt file per line
14     while(getline(Read, line)){
15         // Initialize variables to create card
16         int number;
17         string sNumber = "";
18         string color = "";
19         int idx = 0;
20
21         // Read and Validate Number
22         while(idx < line.length()){
23             if(line[idx] - '0' >= 0 && line[idx] - '0' < 10){
24                 sNumber += line[idx];
25             } else {
26                 if(atoi(sNumber.c_str())){
27                     number = atoi(sNumber.c_str());
28                     if(number < 1 || number > 13){
29                         // Throw error if number out of range
30                         throw ConfigInvalidExc();
31                     }
32                 } else {
33                     // Throw if its not a number
34                     throw ConfigInvalidExc();
35                 }
36                 break;
37             }
38             idx++;
39         }
40
41         // Read and Validate Color
42         idx++;
43         while(idx < line.length() && line[idx] != '\n' && line[idx] != '\r'){
44             color += line[idx];
45             idx++;
46         }
47
48         // Throw error if color unrecognized
49         if(color != "Red" && color != "Yellow" && color != "Blue" && color != "Green"){
50             throw ConfigInvalidExc();
51         } else {
52             // Push new card to deck
53             DC = DC + Card(number, color);
54         }
55     }
56
57     Read.close();
58
59     return DC;
60 }

```

2.5. C++ Standard Template Library

Hampir semua class memanfaatkan C++ Standard Template Library (STL) untuk memanipulasi dan menyimpan data. Salah satu yang sering dipakai untuk menerapkan I/O adalah `<iostream>` untuk melakukan `cin` dan `cout`. Lalu, untuk membaca *file* konfigurasi, digunakan `<fstream>` dan digunakan juga `<time.h>` untuk menerapkan random yang tidak berpola pada *randomizer* urutan kartu. Untuk penyimpanan *data/collection*, digunakan STL yaitu `<vector>`, `<map>`, dan `<pair>` untuk berbagai objek pada program. Penjelasan lebih lanjutnya ada di sebagai berikut.

a. Map

Map digunakan pada program ini untuk mempermudah pengaturan kartu player dan table card menjadi suatu combo yang ingin dibentuk. Contohnya, pada potongan kode di bawah, digunakan collection map untuk mempermudah perhitungan kartu berdasarkan warna. Untuk combo flush sendiri, tentu kita perlu menghitung apakah terdapat lima kartu dengan warna yang sama untuk membentuk combo flush. Dengan penggunaan map, kita dapat memahami kode dengan mudah karena indeks dari collection memiliki tipe string yang merepresentasikan warna tersebut. Tentunya penggunaan map ini tidak hanya di flush saja, namun terdapat juga di berbagai pencarian combo lainnya.

```

1 void Combination::flush() { //Mengatur combo card menjadi kartu - kartu combo flush
2     map<string, int> dictWarna;
3     map<string,int>::iterator itr;
4     string warna;
5     dictWarna["Green"] = 0;
6     dictWarna["Red"] = 0;
7     dictWarna["Yellow"] = 0;
8     dictWarna["Blue"] = 0;
9
10    for (int i = 0; i < this->comboCard.size(); i++) {
11        if (this->comboCard[i].getColor() == "Green") {
12            dictWarna["Green"]++;
13        }
14        else if (this->comboCard[i].getColor() == "Blue") {
15            dictWarna["Blue"]++;
16        }
17        else if (this->comboCard[i].getColor() == "Yellow") {
18            dictWarna["Yellow"]++;
19        }
20        else {
21            dictWarna["Red"]++;
22        }
23    }
24
25    for (itr = dictWarna.begin(); itr != dictWarna.end(); itr++) {
26        if (itr->second >= 5) {
27            warna = itr->first;
28        }
29    }
30
31    vector<Card> combo;
32    for (int i = 0; i < this->comboCard.size(); i++) {
33        if (this->comboCard[i].getColor() == warna) {
34            combo.push_back(this->comboCard[i]);
35        }
36    }
37    while (combo.size() > 5) {
38        combo.erase(combo.begin());
39    }
40    this->comboCard.clear();
41    this->setComboCard(combo);
42 }

```

```

1 bool Combination::isStraightFlush() {
2     //Mengecek apakah combo straight flush dapat dibentuk
3     map<string, int> dictWarna;
4     map<string,int>::iterator itr;
5     dictWarna["Green"] = 0;
6     dictWarna["Red"] = 0;
7     dictWarna["Yellow"] = 0;
8     dictWarna["Blue"] = 0;
9
10    for (int i = 0; i < this->comboCard.size(); i++) {
11        if (this->comboCard[i].getColor() == "Green") {
12            dictWarna["Green"]++;
13        }
14        else if (this->comboCard[i].getColor() == "Blue") {
15            dictWarna["Blue"]++;
16        }
17        else if (this->comboCard[i].getColor() == "Yellow") {
18            dictWarna["Yellow"]++;
19        }
20        else {
21            dictWarna["Red"]++;
22        }
23    }
24    bool flush = false;
25    string warna;
26    for (itr = dictWarna.begin(); itr != dictWarna.end(); itr++) {
27        if (itr->second >= 5) {
28            flush = true;
29            warna = itr->first;
30        }
31    }
32    if (flush) {
33        vector<Card> warnaKartu;
34        for (int i = 0; i < this->comboCard.size(); i++) {
35            if (this->comboCard[i].getColor() == warna) {
36                warnaKartu.push_back(this->comboCard[i]);
37            }
38        }
39        bool temp = false;
40        int idx = warnaKartu.size()-1;
41        int count = 0;
42        while (idx > 0) {
43            if (warnaKartu[idx].getNumber() == warnaKartu[idx-1].getNumber() + 1) {
44                count++;
45            }
46            else if (warnaKartu[idx].getNumber() == warnaKartu[idx-1].getNumber()) {
47                count += 0;
48            }
49            else {
50                count = 0;
51            }
52            idx--;
53        }
54        if (count == 4) {
55            temp = true;
56            break;
57        }
58    }
59    return temp;
60 }
61
62 else {
63     return false;
64 }
65
66 }

```

b. Pair

Pair digunakan pada program ini untuk method yang memanipulasi objek berpasangan. Penggunaan pair disini untuk membantu kemudahan dalam pertukaran antar objek yang dimiliki player. Contohnya, pada kasus ini terdapat ability SwapCard untuk menukar salah satu kartu dari salah satu pemain dan salah satu kartu dari pemain lainnya. Pertukaran ini juga dilakukan secara detail yaitu memerhatikan yang ditukar apakah kartu kiri atau kanan dari pemain. Oleh karena itu, penggunaan pair disini memudahkan pembedaan player dan kartu yang ingin ditukar oleh pemakai ability SwapCard.

```

1  /* Swap other player's cards with another player's cards */
2  void SwapCard::executeAction(CandyGame& Game){
3      int playerTurn = 0;
4      CandyPlayer& playernow = Game.getPlayer(playerTurn);
5      if (playernow.getAbilityCard().getType() != "SWAPCARD"){
6          cout << "\nEits, kamu tidak memiliki kartu SWAPCARD!\n" << endl;
7      } else if (playernow.getAbilityCard().getUseable()){
8          cout << endl << playernow.getNickname() << " melakukan SWAPCARD." << endl;
9          pair<int,int> playerSwap;
10         vector<int> indexPlayers;
11         // Choose 1st player to be swapped
12         cout << "Silahkan pilih pemain yang kartunya ingin anda tukar:" << endl;

```

c. Vector

Vektor digunakan pada program ini untuk mempermudah penyimpanan suatu tipe objek serta memanipulasi masuknya dan keluarnya objek pada penyimpanan tersebut. Penggunaan vektor utama disini untuk mempermudah bentuk sebuah array yang dinamis serta mengelola objek dalam vektor itu sendiri dikarenakan sudah terdapat variasi metode yang diperlukan dalam mengatur objek dalam vektor. Contoh pada kasus dibawah ini terdapat vector yang menyimpan sejumlah objek bertipe Card dan AbilityCard dengan jumlah penyimpanan yang sifatnya dinamis seperti vektor yang menyimpan sejumlah Card pada metode mergeCard ini merupakan gabungan dari kartu milik player serta kartu dari meja sehingga alokasi ukuran vektor yang digunakan harus ditambah lagi. Namun, dengan penggunaan vektor, jumlah tersebut secara langsung berubah karena sifat dinamisnya dan beberapa metode seperti push_back atau erase digunakan untuk mengalokasikan objek - objek ke dalam maupun keluar vektor tersebut.

```

1 void Combination::mergeCard(vector <Card> &TC, vector <Card> &PC) {
    //Menggabungkan kartu player dan kartu meja
    2     vector <Card> PC1;
    3     vector <Card> TC1;
    4     vector <Card> CC;
    5     for (int i = 0; i < 5; i++) {
    6         CC.push_back(TC[i]);
    7         TC1.push_back(TC[i]);
    8     }
    9
    10    for (int i = 0; i < 2; i++) {
    11        CC.push_back(PC[i]);
    12        PC1.push_back(PC[i]);
    13    }
    14    this->clearCombo();
    15    // Sorting card berdasarkan angka dan warnanya
    16
    17    this->quicksort(CC, 0, CC.size()-1);
    18    this->setComboCard(CC);
    19    this->setPlayerCard(PC1);
    20    this->setTableCard(TC1);
    21 }

```

```

1 /* Generate ability deck */
2 DeckCard<AbilityCard> CardGenerator::generateAbilityDeck(){
3     // Initialize Variables and Object
4     DeckCard<AbilityCard> DAC;
5     vector<AbilityCard> temp;
6     string abilities[7] = {"RE-ROLL", "QUADRUPLE", "QUARTER", "REVERSE", "SWAPCARD",
7     "SWITCH", "ABILITYLESS"};
8
9     // Push ability cards to array
10    for(string ability : abilities){
11        temp.push_back(AbilityCard(ability));
12    }
13
14    // Pick random index and push to final deck
15    for(int i = 0 ; i < 7; i++){
16        int randomIndex = rand() % (7-i);
17        DAC = DAC + temp[randomIndex];
18        temp.erase(temp.begin() + randomIndex);
19    }
20
21    return DAC;
22 }

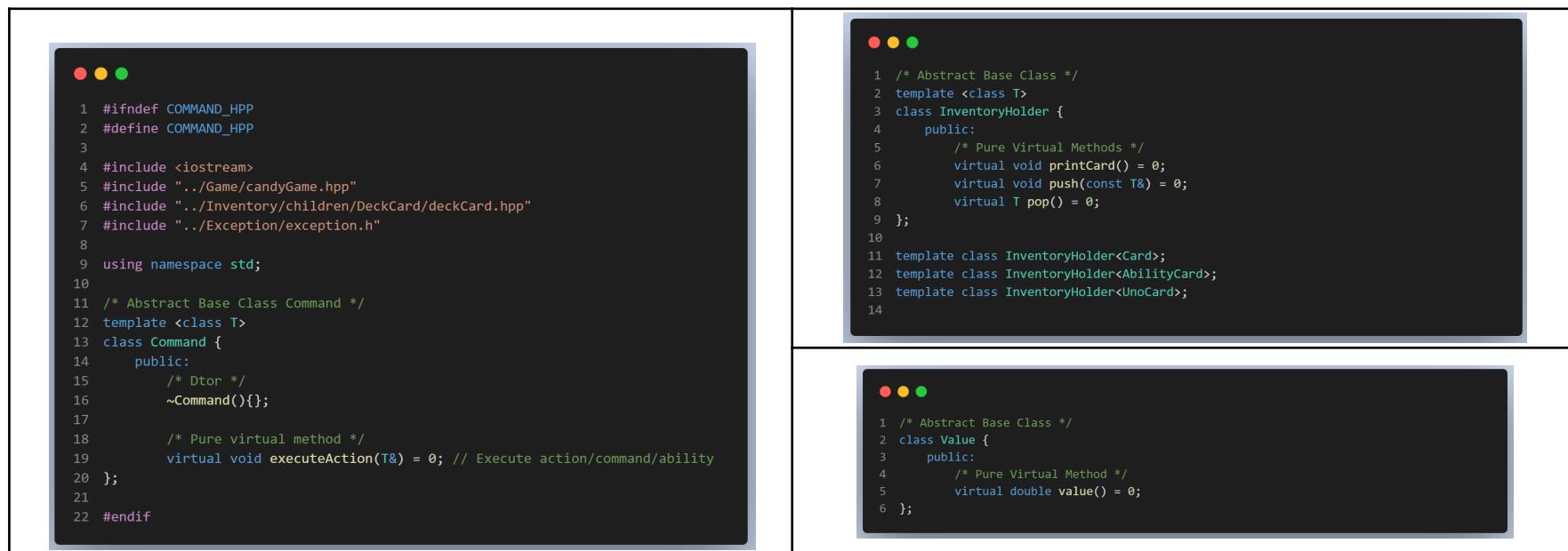
```

2.6. Konsep OOP lain

a. Abstract Base Class

Konsep abstract base class diterapkan pada beberapa kelas yang dibuat yaitu InventoryHolder, Value, Command, dan GameException. InventoryHolder memiliki beberapa pure virtual method seperti printCard, push, dan pop. Hal ini disebabkan turunan-turunan InventoryHolder memiliki atribut yang sama yaitu sebuah container dari objek kartu. Container tersebut memiliki implementasi yang berbeda-beda namun dengan tujuan yang sama. Untuk Value sendiri juga sama, masing-masing turunan memiliki metode value namun

masing-masing implementasi berbeda. Begitu juga untuk Command dan GameException. Untuk abstract base class dari ability, kami menggabungkannya dengan abstract base class Command karena menurut kelompok kami dan saran dari asisten bahwa mereka memiliki tanggung jawab yang sama yaitu melakukan suatu efek terhadap game sehingga abstract base class dari ability-ability adalah Command. Potongan kode seperti di bawah ini.



```

1 #ifndef COMMAND_HPP
2 #define COMMAND_HPP
3
4 #include <iostream>
5 #include "../Game/candyGame.hpp"
6 #include "../Inventory/children/DeckCard/deckCard.hpp"
7 #include "../Exception/exception.h"
8
9 using namespace std;
10
11 /* Abstract Base Class Command */
12 template <class T>
13 class Command {
14     public:
15         /* Dtor */
16         ~Command(){}
17
18         /* Pure virtual method */
19         virtual void executeAction(T&) = 0; // Execute action/command/ability
20 };
21
22 #endif

```



```

1 /* Abstract Base Class */
2 template <class T>
3 class InventoryHolder {
4     public:
5         /* Pure Virtual Methods */
6         virtual void printCard() = 0;
7         virtual void push(const T&) = 0;
8         virtual T pop() = 0;
9     };
10
11 template class InventoryHolder<Card>;
12 template class InventoryHolder<AbilityCard>;
13 template class InventoryHolder<UnoCard>;
14

```



```

1 /* Abstract Base Class */
2 class Value {
3     public:
4         /* Pure Virtual Method */
5         virtual double value() = 0;
6     };

```

b. Composition

Composition (komposisi) adalah sebuah konsep dalam OOP dimana suatu objek menggunakan satu atau lebih objek lain sebagai anggota dari objek itu sendiri. Dalam contoh di bawah ini, objek CandyPlayer yang merupakan pemain dari permainan CandyGame yang memiliki atribut atau variabel objek lain berupa AbilityCard dan Combination yang merupakan objek lain. Hal ini menunjukkan bahwa kelas CandyPlayer dikomposisikan dari objek AbilityCard dan Combination sehingga ketika objek CandyPlayer dikonstruksikan, maka akan tercipta juga objek - objek lain pada atributnya.

```
1 class CandyPlayer: public Player<Card>{
2     private:
3         long long int point; /* Determine the player's point */
4         AbilityCard ability; /* Player's ability card */
5         Combination kombo; /* Player's combo */
```

c. Collection

Collection (koleksi) adalah sebuah konsep struktur data yang digunakan untuk menyimpan serta mengelola kumpulan objek - objek dalam suatu tipe data. Koleksi ini sering digunakan untuk mengelola data dalam jumlah besar atau dinamis. Beberapa penerapan koleksi dalam program ini antara lain vektor, set, dan map. Contoh salah satu penerapan collection di bawah ini adalah penggunaan vektor untuk dengan tipe data dalamnya generik. Penggunaan collection ini diperlukan karena dinamika alokasi data yang ada dalam keperluan permainan seperti Card dan Combination yang membutuhkan pengelolaan data dalam jumlah tertentu yang bersifat dinamis.

```
1 template <class T>
2 class DeckCard: public InventoryHolder<T>{
3     private:
4         vector<T> deck; /* List of cards */
5         int neff; /* Determine how many cards are currently in the deck */
6         int size; /* Determine the maximum size of the deck */
```

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Generic Class

Konsep kelas generik diterapkan pada program ini untuk memudahkan implementasi lebih dari satu jenis permainan, yaitu permainan kartu Kerajaan Permen (CandyGame) dan permainan kartu UNO (UnoGame). Adapun kelas generik yang terdefinisi pada program ini antara lain kelas InventoryHolder, TableCard, DeckCard, Player, dan Command.

Pada kelas InventoryHolder, *template* generik T dimanfaatkan sebagai tipe elemen list kartu pada kelas turunan generik TableCard, DeckCard, dan Player. Hal ini perlu dilakukan karena TableCard, DeckCard, dan Player pada permainan kartu Kerajaan Permen (CandyGame) dan permainan kartu UNO (UnoGame) serupa, hanya berbeda jenis kartu yang dipakai. Pada permainan CandyGame, jenis kartu yang digunakan adalah Card, sedangkan pada permainan UnoGame, jenis kartu yang digunakan adalah UnoCard.

DeckCard dan TableCard juga didefinisikan sebagai kelas generik yang dapat dimanfaatkan untuk berbagai jenis kartu. Pada CandyGame, terdapat DeckCard dengan jenis kartu Card dan AbilityCard, serta TableCard dengan jenis Card. Sementara itu, pada UnoGame, terdapat DeckCard dan TableCard dengan jenis kartu UnoCard.

Kelas Player juga merupakan kelas turunan dari InventoryHolder dan merupakan kelas generik untuk dua jenis kartu. Hal ini disebabkan karena Player memiliki atribut DeckCard yang menyimpan kartu dengan jenis yang berbeda untuk permainan yang berbeda. Pada permainan CandyGame, Player memiliki atribut DeckCard dengan jenis Card, sedangkan pada permainan UnoGame, Player memiliki atribut DeckCard dengan jenis UnoCard.

```
● ● ●  
1 #ifndef INVENTORY HOLDER_HPP  
2 #define INVENTORY HOLDER_HPP  
3 #include "../Value/Card/abilityCard.hpp"  
4 #include "../Value/Card/children/unoCard.hpp"  
5 #include "../Value/Card/card.hpp"  
6 #include <iostream>  
7 using namespace std;  
8  
9 /* Abstract Base Class */  
10 template <class T>  
11 class InventoryHolder {  
12     public:  
13         /* Pure Virtual Methods */  
14         virtual void printCard() = 0;  
15         virtual void push(const T&) = 0;  
16         virtual T pop() = 0;  
17     };  
18  
19 template class InventoryHolder<Card>;  
20 template class InventoryHolder<AbilityCard>;  
21 template class InventoryHolder<UnoCard>;  
22  
23 #endif
```

```

1 #ifndef PLAYER_HPP
2 #define PLAYER_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../../inventoryHolder.hpp"
7 #include "../DeckCard/deckCard.hpp"
8 #include "../../../../../Value/Card/abilityCard.hpp"
9 #include "../../../../../Value/Combination/combination.hpp"
10
11 using namespace std;
12
13 template <class T>
14 class Player: public InventoryHolder<T>{
15     protected:
16         int id; /* Player's ID */
17         string nickname; /* Player's nickname */
18         vector<T> mainDeck; /* Player's deck card */
19         static int playersCreated; /* Determine how many players have been created */
20
21     public:
22         /* Ctor */
23         Player();
24         Player(string nickname);
25
26         /* Other Methods */
27         T getCard(int index);
28         void push(const T & PC);
29         void push(DeckCard<T>& DC, int count);
30         T pop();
31         virtual void printCard() = 0;
32
33         /* Getter */
34         string getNickname();
35         int getID();
36         vector<T>& getDeckPlayer();
37     };
38
39 template class Player<Card>;
40 template class Player<AbilityCard>;
41 template class Player<UnoCard>;
42
43 #endiff

```

```

1 #ifndef DECK_CARD_HPP
2 #define DECK_CARD_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../../inventoryHolder.hpp"
7 #include "../../../../../Value/Card/card.hpp"
8 #include "../../../../../Value/Card/children/unoCard.hpp"
9 #include "../../../../../Value/Card/abilityCard.hpp"
10
11 using namespace std;
12
13 template <class T>
14 class DeckCard: public InventoryHolder<T>{
15     private:
16         vector<T> deck; /* List of cards */
17         int neff; /* Determine how many cards are currently in the deck */
18         int size; /* Determine the maximum size of the deck */
19
20     public:
21         /* Ctor */
22         DeckCard();
23         DeckCard(int neff, int size);
24
25         /* Operator overloading */
26         DeckCard& operator=(const DeckCard& dc);
27         DeckCard& operator+(const T& add);
28         DeckCard& operator-(const T& min);
29         DeckCard& operator--();
30
31         /* Getter */
32         int getNeff();
33         int getSize();
34         T& getCard(int i);
35
36         /* Setter */
37         void setCard(const T &card, int i);
38
39         /* Other methods */
40         T pop();
41         void push(const T & Card);
42         void printCard();
43     };
44
45 #endiff

```

```
1 #ifndef COMMAND_HPP
2 #define COMMAND_HPP
3
4 #include <iostream>
5 #include "../Game/candyGame.hpp"
6 #include "../Inventory/children/DeckCard/deckCard.hpp"
7 #include "../Exception/exception.h"
8
9 using namespace std;
10
11 /* Abstract Base Class Command */
12 template <class T>
13 class Command {
14     public:
15         /* Virtual dtor */
16         ~Command(){};
17
18         /* Pure virtual method */
19         virtual void executeAction(T&) = 0; // Execute action/command/ability
20 };
21
22 #endif
```

```
1 #ifndef TABLE_CARD_HPP
2 #define TABLE_CARD_HPP
3
4 #include <iostream>
5 #include <vector>
6 #include "../Value/Card/card.hpp"
7 #include "../DeckCard/deckCard.hpp"
8
9 using namespace std;
10
11 template <class T>
12 class TableCard : public InventoryHolder<T> {
13     private:
14         int neff; /* Determine how many cards on table */
15         vector<T> listOfCards; /* List of cards */
16         const int max; /* Determine the maximum amount of cards on table */
17
18     public:
19         /* Ctor */
20         TableCard(int max);
21         TableCard(const TableCard& other);
22
23         /* Getter */
24         vector<T>& getTableCard();
25         int getNeff();
26         T& getTop();
27
28         /* Other methods */
29         void push(const T& c);
30         void clearTable();
31         void printCard();
32         T pop();
33     };
34
35 template class TableCard<Card>;
36 template class TableCard<UnoCard>;
37
38 #endif
```

```
● ● ●

1 #ifndef INVENTORY HOLDER_HPP
2 #define INVENTORY HOLDER_HPP
3 #include "../Value/Card/abilityCard.hpp"
4 #include "../Value/Card/children/unoCard.hpp"
5 #include "../Value/Card/card.hpp"
6 #include <iostream>
7 using namespace std;
8
9 /* Abstract Base Class */
10 template <class T>
11 class InventoryHolder {
12     public:
13         /* Pure Virtual Methods */
14         virtual void printCard() = 0;
15         virtual void push(const T&) = 0;
16         virtual T pop() = 0;
17     };
18
19 template class InventoryHolder<Card>;
20 template class InventoryHolder<AbilityCard>;
21 template class InventoryHolder<UnoCard>;
22
23 #endif
```

3.1.2. Game Kartu UNO

Permainan kartu UNO dapat dimainkan oleh beberapa pemain. Prinsip permainan ini adalah menimpa kartu teratas pada table card dengan kartu yang serupa, baik dari segi angka, warna, maupun *ability*. Terdapat tiga jenis tumpukan kartu, yaitu deck card, table card, dan player deck card. Deck card adalah tumpukan kartu pada deck yang berjumlah 108 kartu. Table card adalah tumpukan kartu yang telah dikeluarkan selama permainan berlangsung. Player deck card adalah tumpukan kartu yang dipegang oleh masing-masing pemain. Setiap pemain memiliki tujuh buah kartu yang dibagikan secara acak pada player deck card. Kartu pertama pada table card merupakan kartu yang diambil dari

deck card (kartu pertama dipastikan bukan kartu Plus4 atau Change Color). Pemain pertama akan mengeluarkan sebuah kartu yang memiliki kesamaan dengan kartu teratas pada table card. Kartu yang valid untuk dikeluarkan adalah kartu dengan angka yang sama dengan kartu teratas table card, kartu dengan warna yang sama dengan kartu teratas table card, kartu dengan *ability* yang sama dengan kartu teratas table card, maupun kartu tanpa warna (Plus4 dan Change Color). Pemain yang menghabiskan kartu paling cepat menjadi pemenang permainan.

Kartu UNO terbagi menjadi dua jenis, yaitu kartu angka dan kartu *ability*. Selain itu, setiap kartu juga diidentifikasi dengan warna. Pada permainan UNO, ada empat warna kartu, yaitu biru, hijau, merah, dan kuning. Setiap kartu angka pasti memiliki salah satu warna, sedangkan kartu *ability* ada yang memiliki warna dan ada yang tidak. Kartu angka pada permainan UNO memiliki simbol angka dalam jangkauan 0 sampai 9. Adapun kartu *ability* yang terdapat pada permainan UNO adalah kartu Plus 2, Plus 4, Reverse, Skip, dan Change Color. Kartu Plus 2 merupakan kartu *ability* berwarna yang membuat pemain berikutnya mengambil dua buah kartu dari deck, sehingga giliran pemain yang terkena efek Plus 2 dilewati. Kartu Plus 4 merupakan kartu *ability* tanpa warna yang membuat pemain berikutnya mengambil empat buah kartu dari deck, sehingga giliran pemain yang terkena efek Plus 4 dilewati. Pemain yang mengeluarkan kartu Plus 4 juga dapat menentukan warna untuk pemain selanjutnya. Kartu Reverse merupakan kartu *ability* berwarna yang mengubah arah putaran permainan. Kartu Skip merupakan kartu *ability* berwarna yang membuat giliran pemain berikutnya dilewati. Kartu Change Color merupakan kartu *ability* tanpa warna yang dapat mengubah warna kartu permainan untuk pemain berikutnya, sehingga pemain berikutnya harus mengeluarkan kartu dengan warna yang sesuai dengan keinginan pemain yang menggunakan kartu Change Color.

Apabila pada gilirannya seorang pemain tidak memiliki kartu yang serupa dengan kartu teratas table card, maka pemain harus mengambil satu buah kartu dari deck. Apabila kartu yang diambil dari deck tersebut serupa dengan kartu teratas table card, maka pemain dapat mengeluarkan kartu tersebut. Jika tidak, maka giliran pemain dilewati.

Ketika kartu pada player deck seorang pemain bersisa satu kartu, maka pemain tersebut harus mengatakan “UNO”. Apabila pemain tidak mengatakan “UNO” ketika dia hanya memiliki satu kartu atau ketika pemain mengatakan “UNO” ketika kartunya masih lebih dari satu, maka pemain terkena penalti mengambil dua buah kartu dari deck.

Pada program ini, implementasi permainan kartu UNO diterapkan dengan memanfaatkan *abstract base class* dan kelas generik yang telah didefinisikan sebelumnya. Kelas-kelas yang digunakan dalam membangun permainan UNO antara lain adalah UnoGame, UnoPlayer, UnoCard, DeckCard, TableCard, CardGenerator, Exception beserta turunannya, CommandParserUNO, dan juga Command beserta turunannya untuk perintah permainan kartu UNO.

Kelas UnoGame merupakan kelas turunan dari Game. Kelas UnoPlayer merupakan kelas turunan dan memanfaatkan kelas generik Player. Kelas UnoCard merupakan kelas turunan dari Card dan AbilityCard karena jenis kartu pada permainan UNO merupakan kombinasi antara kartu angka dan kartu *ability* (beberapa kartu *ability* memiliki atribut warna). Kelas DeckCard dan TableCard digunakan sebagai kelas generik untuk jenis UnoCard. Kelas CardGenerator juga digunakan untuk membaca *file* konfigurasi dan *shuffle* kartu pada deck. Kelas Exception beserta turunannya digunakan untuk melemparkan pesan *error* apabila terdapat masukan yang tidak valid. Kelas CommandParserUNO menerjemahkan perintah masukan dari pemain menjadi aksi yang dilakukan pada permainan. Kelas Command juga digunakan sebagai kelas dasar untuk beberapa kelas turunan perintah permainan UNO. Adapun kelas turunan Command yang digunakan pada permainan UNO adalah PickCard, Draw, Pass, SayUno, ChangeColor, Plus2, Plus4, Skip, UnoReverse, CheckCard, dan HelpUno.

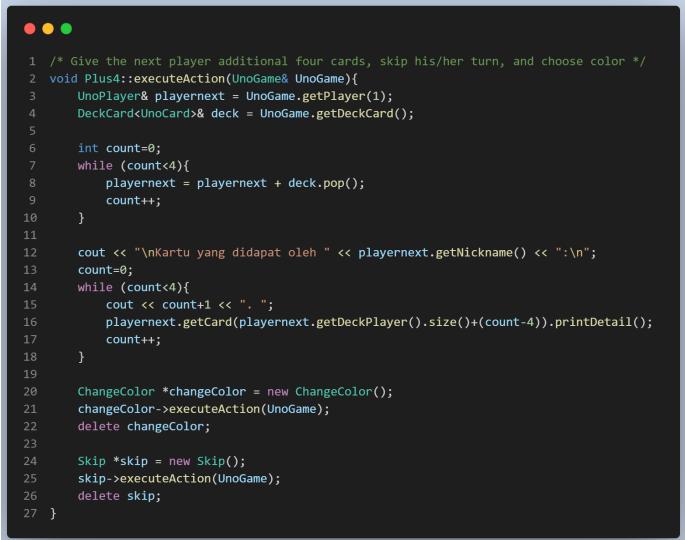
Berikut adalah beberapa contoh pengimplementasian dari Command dan Ability yang tersedia pada game UNO yang kami terapkan pada program kami.

```

1  /* Choose card to use */
2  void PickCard::executeAction(UnoGame& Game){
3      // Validasi ada kartu yang bisa dipakai atau tidak
4      if(!checkValid(Game)){
5          cout << "\nMaaf, kamu tidak punya kartu yang bisa dipakai. Silahkan draw!" << endl;
6      } else {
7          int input_number;
8          UnoPlayer& playernow = Game.getPlayer(0);
9          UnoCard SCard;
10
11         // Print kartu yang ada dulu
12         cout << "\nKartu yang dimiliki player sekarang: " << endl;
13         playernow.printCard();
14
15         bool validCard = false;
16         // Validasi input dan kartu
17         while(!validCard){
18             try {
19                 cout << "\nSilahkan pilih nomor kartu yang ingin digunakan : ";
20                 cin >> input_number;
21                 if(cin.fail()){
22                     cin.clear();
23                     cin.ignore(numeric_limits<streamsize>::max(), '\n');
24                     throw InputActionInvalidExc();
25                 }
26                 if(!(input_number-1 >= 0 && input_number-1 < playernow.getDeckPlayer().size())){
27                     throw InputNumberInvalidExc();
28                 } else {
29                     // Validasi input kartu yang bisa dikeluarkan
30                     SCard = playernow.getDeckPlayer()[input_number-1];
31                     if (!Game.getTop() == SCard){
32                         cout << "\nKartu tidak bisa dipakai. Silahkan pilih kartu lain!" << endl;
33                     } else {
34                         validCard = true;
35                     }
36                 }
37             } catch(GameException& err){
38                 cout << endl << err.what() << endl;
39             }
40         }
41
42         /* add to deck card, retrieve from player deck */
43         TableCard<UnoCard>& tableCard = Game.getTableCard();
44         tableCard.push(playernow.getCard(input_number-1));
45         playernow.getDeckPlayer().erase(playernow.getDeckPlayer().begin() + input_number-1);
46
47         if (!SCard.getNumber()){
48             if (SCard.getType() == "PLUS2"){
49                 Plus2 *plus2 = new Plus2();
50                 plus2->executeAction(Game);
51             }
52             else if (SCard.getType() == "PLUS4"){
53                 Plus4 *plus4 = new Plus4();
54                 plus4->executeAction(Game);
55             }
56             else if (SCard.getType() == "SKIP"){
57                 Skip *skip = new Skip();
58                 skip->executeAction(Game);
59             }
60             else if (SCard.getType() == "UNOREVERSE"){
61                 UnoReverse *unoReverse = new UnoReverse();
62                 unoReverse->executeAction(Game);
63             }
64             else if (SCard.getType() == "CHANGECOLOR"){
65                 ChangeColor *changeColor = new ChangeColor();
66                 changeColor->executeAction(Game);
67             }
68         }
69
70         Game.setValid(true);
71         Game.setAlreadyDraw(false);
72     }
73 }
```

```

1  /* Execute change color */
2  void ChangeColor::executeAction(UnoGame& UnoGame){
3      cout << "\nSilakan pilih warna untuk diganti." << endl;
4      cout << "1. Red" << endl;
5      cout << "2. Green" << endl;
6      cout << "3. Blue" << endl;
7      cout << "4. Yellow" << endl;
8
9      int colorInput;
10
11     do {
12         try{
13             cout << "Pilihan : ";
14             cin >> colorInput;
15             if(cin.fail()){
16                 cin.clear();
17                 cin.ignore(numeric_limits<streamsize>::max(), '\n');
18                 throw InputActionInvalidExc();
19             }
20             if(colorInput < 1 || colorInput > 4){
21                 throw InputNumberInvalidExc();
22             }
23         } catch(GameException& err){
24             cout << err.what() << endl;
25         }
26     } while(colorInput < 1 || colorInput > 4);
27
28     TableCard<UnoCard>& tableCards = UnoGame.getTableCard();
29     UnoCard& topCard = tableCards.getTop();
30
31     if (colorInput<1 || colorInput>4){
32         throw InputNumberInvalidExc();
33     } else {
34         if (colorInput==1){
35             topCard.setColor("Red");
36         } else if (colorInput==2){
37             topCard.setColor("Green");
38         } else if (colorInput==3){
39             topCard.setColor("Blue");
40         } else {
41             // colorInput == 4
42             topCard.setColor("Yellow");
43         }
44     }
45 }
```



```

1 /* Give the next player additional four cards, skip his/her turn, and choose color */
2 void Plus4::executeAction(UnoGame& UnoGame){
3     UnoPlayer& playernext = UnoGame.getPlayer(1);
4     DeckCard<UnoCard>& deck = UnoGame.getDeckCard();
5
6     int count=0;
7     while (count<4){
8         playernext = playernext + deck.pop();
9         count++;
10    }
11
12    cout << "\nKartu yang didapat oleh " << playernext.getNickname() << ":"< n";
13    count=0;
14    while (count<4){
15        cout << count+1 << ". ";
16        playernext.getCard(playernext.getDeckPlayer().size()+(count-4)).printDetail();
17        count++;
18    }
19
20    ChangeColor *changeColor = new ChangeColor();
21    changeColor->executeAction(UnoGame);
22    delete changeColor;
23
24    Skip *skip = new Skip();
25    skip->executeAction(UnoGame);
26    delete skip;
27 }

```



```

1 bool UnoCard::operator==(const UnoCard& other){
2     if (this->isNumber && other.isNumber){
3         // Keduanya kartu angka
4         if (this->number == other.number || this->color == other.color){
5             return true; // true jika angka sama atau warna sama
6         } else {
7             return false;
8         }
9     } else {
10         // salah satunya atau keduanya kartu ability
11         if (this->isNumber){
12             // kartu pertama angka, kartu kedua ability
13             if (this->color == other.color || other.color == "Black"){
14                 return true; // true jika warna sama atau ability card tanpa warna
15             } else {
16                 return false;
17             }
18         } else if (other.isNumber){
19             // kartu pertama ability, kartu kedua angka
20             if (this->color == other.color || this->color == "Black"){
21                 return true; // true jika warna sama atau ability card tanpa warna
22             } else {
23                 return false;
24             }
25         } else {
26             // keduanya kartu ability
27             if (this->color == other.color || this->type == other.type || this->color
28 == "Black" || other.color == "Black"){
29                 return true;
30             } else {
31                 return false;
32             }
33         }
34     }

```

3.2. Bonus Kreasi Mandiri

3.2.1. Help Command

Kelompok kami memutuskan untuk menerapkan sebuah bonus fitur berupa command HELP untuk mencetak kumpulan dari command-command yang dapat dilakukan oleh pemain. Tujuan command ini diciptakan adalah untuk memudahkan pemain memasukkan perintah permainan. Karena program kami terdiri dari dua jenis permainan, maka command HELP juga didefinisikan untuk dua jenis permainan tersebut, yaitu kelas Help untuk permainan CandyGame dan juga kelas HelpUno untuk permainan UnoGame. Meskipun didefinisikan sebagai dua kelas yang berbeda, perintah yang dimasukkan pemain sama, yaitu “HELP”.

```

1  /* Print all available commands to use in UnoGame */
2 void HelpUno::executeAction(UnoGame &unogame) {
3     cout << "\n
Berikut adalah command-command yang dapat digunakan oleh pemain: "
4     << endl;
5     cout <<
"1. DRAW : command ini digunakan untuk mengambil kartu dari deck kartu"
6     << endl;
7     cout <<
"2. PASS : command ini digunakan apabila pemain memutuskan untuk tidak menaruh kartu"
8     << endl;
9     cout <<
"3. SAYUNO : command ini digunakan apabila pemain hanya memiliki 1 kartu saja (penjelasan lebih lanjut ada di rules game UNO)"
10    << endl;
11    cout <<
"4. PICKCARD : command ini digunakan untuk mengambil kartu dari deck kartu"
12    << endl;
13    cout <<
"5. CHECKCARD : command ini digunakan untuk mengecek kartu apa saja yang dimiliki pemain
\n" << endl;
14 }

```

```

1  /* Print all available commands to use */
2 void Help::executeAction(CandyGame &candyGame) {
3     cout << "\n
Berikut adalah beberapa command yang dapat dilakukan pemain" << endl;
4     cout <<
"1. NEXT : command ini tidak melakukan apa-apa dan giliran diteruskan ke pemain selanjutnya"
5     << endl;
6     cout <<
"2. DOUBLE : command ini digunakan untuk membuat poin hadiah berjumlah 2x dari sebelumnya"
7     << endl;
8     cout <<
"3. CHECKCARD : command ini digunakan untuk melihat kartu yang dimiliki pemain"
9     << endl;
10    cout <<
"4. CHECKTABLECARD : command ini digunakan untuk melihat kartu yang terdapat pada table"
11    << endl;
12    cout <<
"5. HALF : command ini digunakan untuk membuat poin hadiah berjumlah 0.5x dari sebelumnya"
13    << endl;
14    cout <<
"6. RE-ROLL : command ini hanya dapat dilakukan apabila pemain memiliki Re-Roll Card, dimana command ini akan membuang 2 kartu dari main deck pemain dan mengambil ulang 2 kartu dari deck kartu"
15    << endl;
16    cout <<
"7. QUADRUPLE : command ini hanya dapat digunakan apabila pemain memiliki Quadruple Card, dimana command ini akan membuat poin hadiah berjumlah 4x dari sebelumnya"
17    << endl;
18    cout <<
"8. QUARTER : command ini hanya dapat digunakan apabila pemain memiliki Quarter Card, dimana command ini akan membuat poin hadiah berjumlah 0.25x dari sebelumnya"
19    << endl;
20    cout <<
"9. REVERSE : command ini hanya dapat digunakan apabila pemain memiliki Reverse Card, dimana command ini akan memutar arah giliran pada giliran selanjutnya"
21    << endl;
22    cout <<
"10. SWAPCARD : command ini hanya dapat digunakan apabila pemain memiliki SwapCard Card, dimana command ini akan menukar 1 kartu main deck pemilik lain dengan 1 kartu main deck milik pemain lain. Tidak boleh ditukar dengan kartu main deck diri sendiri"
23    << endl;
24    cout <<
"11. SWITCH : command ini hanya dapat digunakan apabila pemain memiliki Switch Card, dimana command ini akan menukar 2 kartu main deck milik diri sendiri dengan 2 kartu main deck milik pemain lain. Harus menukar milik diri sendiri dengan pemain lain"
25    << endl;
26    cout <<
"12. ABILITYLESS : command ini hanya dapat digunakan apabila pemain memiliki Abilityless Card, dimana command ini akan mematikan kartu ability pemain lain
\n" << endl;
27 }

```

3.2.2. Check Card Command

Kelompok kami memutuskan untuk menerapkan sebuah bonus fitur berupa command CHECKCARD unntuk mencetak kartu-kartu yang dimiliki oleh pemain. Perintah ini hanya dapat dipanggil pada permainan CandyGame (permainan UnoGame sudah memiliki perintah PickCard untuk mengecek kartu dan mengeluarkan kartu ke table card).

```
Sekarang adalah giliran pemain 2  
Command : CHECKCARD  
  
Kartu Pemain 2 :  
1. 11 Green  
2. 1 Yellow  
3. RE-ROLL Ability Card
```

3.2.3. Check Table Card Command

Kelompok kami memutuskan untuk menerapkan sebuah bonus fitur berupa command CHECKCARD untuk mencetak kartu-kartu yang ada pada meja. Perintah ini hanya dapat dipanggil pada permainan CandyGame (karena pada UnoGame, informasi yang dibutuhkan hanya kartu teratas pada table card).

```
RONDE 6  
  
Sekarang adalah giliran pemain 6  
Command : CHECKTABLECARD  
  
Isi dari table card sekarang :  
1. 4 Blue  
2. 2 Green  
3. 2 Red  
4. 10 Green  
5. 9 Red
```

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Combination (Termasuk Perhitungan Rumus)	13521127, 13521143	13521127, 13521143
Inventory Holder	13521145	13521145
Card Generator	13521124	13521124
Game	13521124	13521124
Exception	13521059	13521059
Command	13521059	13521059

5. Lampiran

Tautan Repository Github:

https://github.com/maikeljh/Tubes1_OOP_OOPeZ

Tautan Dokumentasi Rumus Perhitungan Combo:

<https://docs.google.com/spreadsheets/d/1sJ0wlFmKf-c-DB5quVO0UZ26KuUB23nqb4OgkxjvLXI/edit?usp=sharing>

Foto Kelompok:



Kode Kelompok : OEB

Nama Kelompok : OOPeZ

1. 13521059 / Arleen Chrysantha Gunardi
2. 13521124 / Michael Jonathan Halim
3. 13521127 / Marcel Ryan Antony
4. 13521143 / Raynard Tanadi
5. 13521145 / Kenneth Dave Bahana

Asisten Pembimbing : Aditya Bimawan (NIM. 13519064)

1. Konten Diskusi

- a. Pertanyaan: Masih kurang mengerti mengenai fungsi kelas abstrak InventoryHolder.

Jawaban: Harus didefinisikan *virtual function*-nya, misal push, pop, atau getsize didefinisikan di InventoryHolder-nya. Boleh juga menambahkan atribut.

- b. Pertanyaan: Apakah ability card boleh bukan turunan dari card?

Jawaban: Boleh saja karena tidak ada di spek minimal.

- c. Pertanyaan: Apakah ada pengurangan poin apabila terdapat atribut yang tidak terpakai untuk ke depannya?

Jawaban: Kalau sedikit-sedikit seharusnya tidak masalah tidak di-minus juga.

- d. Pertanyaan: Generic function untuk kartu mengembalikan nilainya bagaimana ya kak?

Jawaban: Bagian mekanisme ada pembobotan kartu. Untuk ngebandingin kartu aja generic function di kartunya. Jadi, kalau tidak ada combo sama sekali, maka mengambil 1 kartu terbesar di tangan pemain sebagai high card.

- e. Pertanyaan: Untuk mainnya, itu interaksi antara objek atau kita bikin mainnya di dalam metode pada kelas game?

Jawaban: Bikinnya di game saja karena ini membuat game, berhubung bonus juga bisa bikin game baru, jadi kalau isi mainnya minimal banget juga tidak apa-apa.

f. Pertanyaan (kakaknya): Untuk memberikan command ke player class kalian bagaimana?

Jawaban: Objek command itu untuk parser command yang diketik dan melakukan aksi yang dipilih player. Nanti pemain tu akan mengetik perintah yang pengen dilakukan jadi perlu ada validasi. Kalau sudah valid, akan mengembalikan integer. Integer itu command id dan dimiliki unik untuk masing-masing.

g. Pertanyaan (kakanya): Terus kalo useAction useAbility itu untuk apa?

Jawaban: Itu untuk membedakan mana yang aksi mana yang ability. Ini di command ada pembagian useAction sama useAbility.

h. Pertanyaan: Karena setiap ronde bertambah kartu pada table card, pada ronde 6 apakah player masih bisa melakukan aksi?

Jawaban: Masih bisa karena saat ronde 5 selesai, seharusnya sudah ada 5 kartu pada table card, kemudian aksi terakhir ronde 6 baru poin akhir dibandingkan.

i. Pertanyaan: Apakah array of player cards boleh langsung pake vektor, tanpa alokasi pointer?

Jawaban: Boleh.

j. Pertanyaan: Untuk ketentuan buat create object, apakah harus pakai pointer (seperti new Game()) atau boleh langsung?

Jawaban: Sesuai kebutuhan saja, tidak ada kewajibannya. Kalau butuh menggunakan pointer, pakailah pointer.

k. Pertanyaan: Apakah ada saran untuk pemakaian map kira-kira untuk kasus seperti kartu itu di bagian mana ya?

Jawaban: Bisa untuk leaderboard, seperti skor pemainnya gitu.

l. Pertanyaan: Untuk asistensi minimal 1 kali untuk setiap minggu atau tidak ya kak?

Jawaban: Minimal sekali selama penggeraan tubes. Jadi kalau merasa ini sudah cukup, tidak perlu lagi juga tidak apa-apa.

m. Pertanyaan: Apabila butuh asistensi lagi caranya bagaimana ya kak?

Jawaban: Langsung kontak saja.

n. Pertanyaan: Untuk config urutan kartunya apakah bebas atau ditentukan asisten ya kak?

Jawaban: Ditanya dulu ya (belum bisa dijawab) mengenai config.

o. Pertanyaan: Untuk implementasi generic itu apakah suatu metode dalam class atau diluar dari class?

Jawaban: Seharusnya diluar objek karena generic minimal player atau kombo kartu bisa saja menerima integer, jadi generic kayaknya diluar objek aja.

Pertanyaan lanjut: Mungkin seperti di main.cpp, buat file khusus untuk generic functionnya gitu ya kak?

Jawaban lanjut: Iya, bisa.

2. Tindak Lanjut

- a. Karena pada dasarnya diharuskan untuk membuat kelas abstrak berupa InventoryHolder, maka kelompok kami memutuskan untuk membuat *pure virtual methods* berupa push dan pop pada InventoryHolder karena sama-sama membungkus suatu koleksi kartu.
- b. AbilityCard menjadi sebuah kelas sendiri karena sifatnya yang berbeda dengan kartu biasa.
- c. Kami menghilangkan pendefinisian atribut yang tidak terpakai.
- d. Kami membuat *generic function* dan *operator overloading* untuk mencari nilai terbesar dari kartu, player, dan kombo.
- e. Kami membuat main khusus di dalam setiap kelas game yang menjalankan program utama permainan yang dimainkan sehingga program utama hanya berisi penentuan game yang ingin dimainkan.
- f. Karena aksi yang dilakukan berdasarkan *ability* dan *command* sama-sama memberikan efek kepada kelas permainan, maka kami memutuskan untuk membuat *abstract base class* berupa Command saja yang menerapkan polimorfisme kepada kelas-kelas turunan yang menerapkan *ability* dan *command* pada game.

- g. Karena kami berencana untuk membuat game utama dan game UNO, maka kami membuat kelas-kelas utama kami generik seperti Player, InventoryHolder, DeckCard, dan lainnya untuk menampung jenis-jenis kartu yang berbeda pada permainan yang berbeda sehingga ketika nanti membuat game baru, akan jauh lebih mudah untuk melakukan ekstensi dan pengimplementasiannya.
 - h. Kami banyak merevisi hierarki kelas menjadi sebagai berikut.

