

Kode Kelompok : SST

Nama Kelompok : firSST

1. 13521001 / Angger Ilham A
2. 13521008 / Jason Rivalino
3. 13521013 / Eunice Sarah Siregar
4. 13521014 / Muhamad Syauqi Jannatan
5. 13521016 / Laila Bilbina Khoiru Nisa
6. 13521031 / Fahrian Afdholi

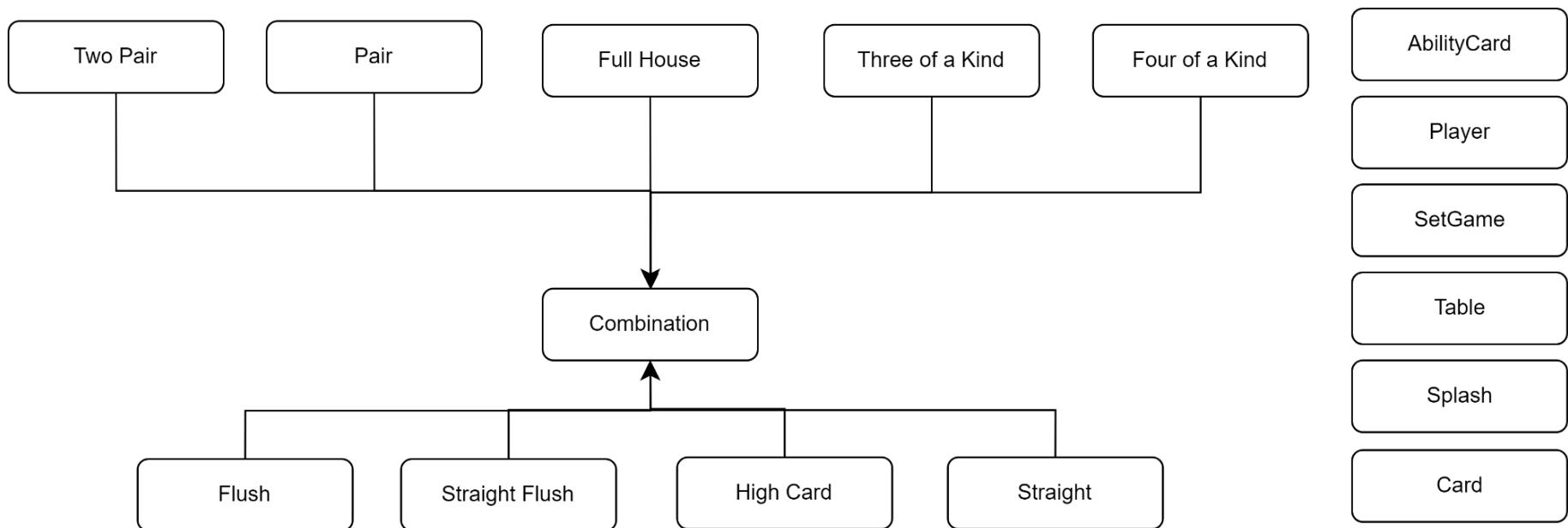
Asisten Pembimbing : Muhammad Tito Prakasa

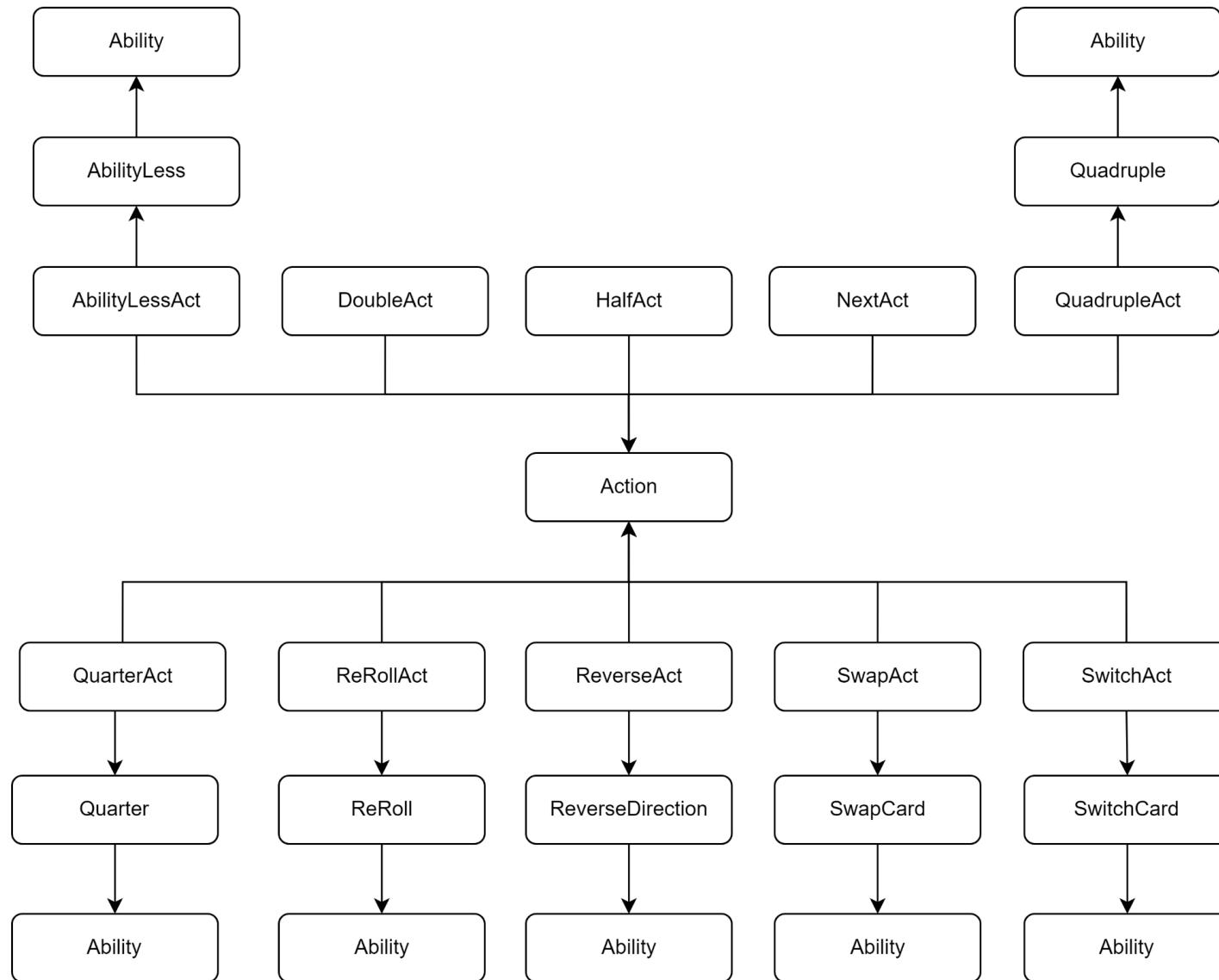
1. Diagram Kelas

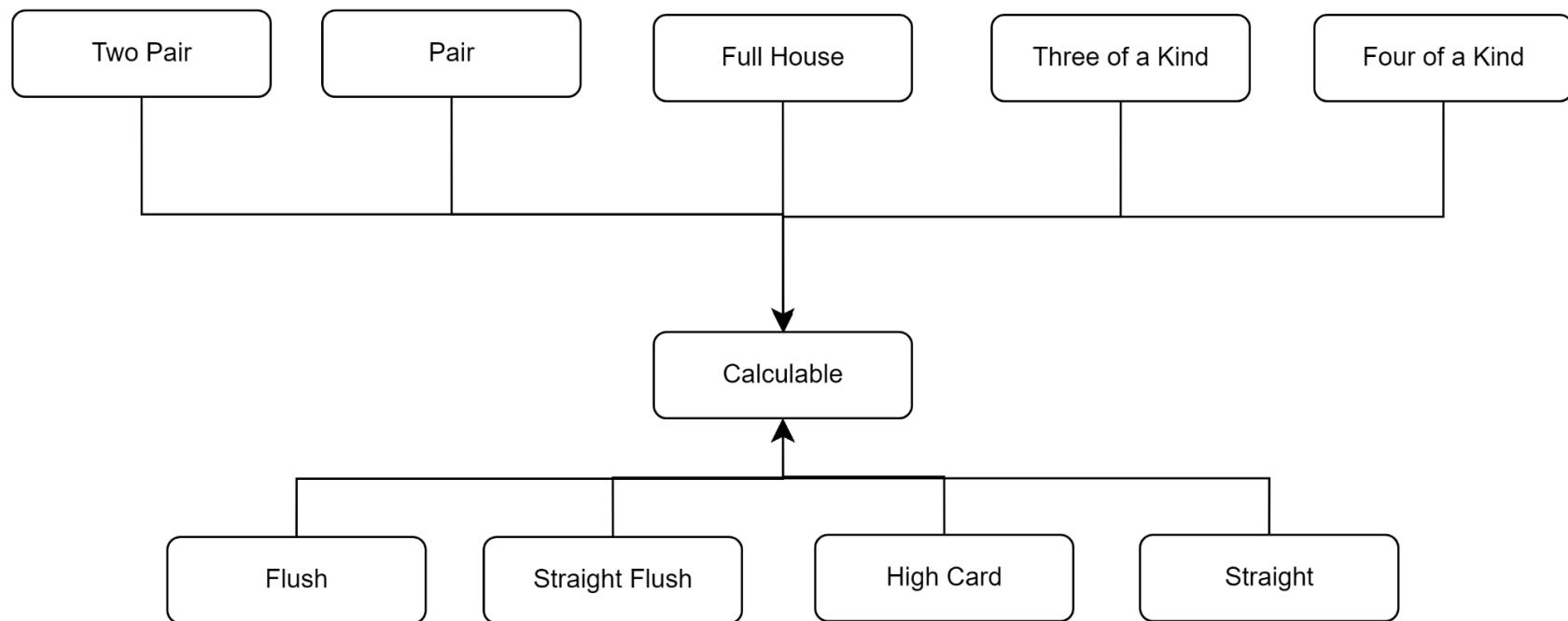
Diagram kelas dibuat secara manual menggunakan aplikasi draw.io. Diagram kelas berisi kelas-kelas utama (belum termasuk kelas untuk error handling) yang digunakan dalam file main.cpp. Setiap kelas berisi semua atribut dan method dalam kelas tersebut. Berikut adalah nama-nama kelas yang digunakan dalam main.cpp :

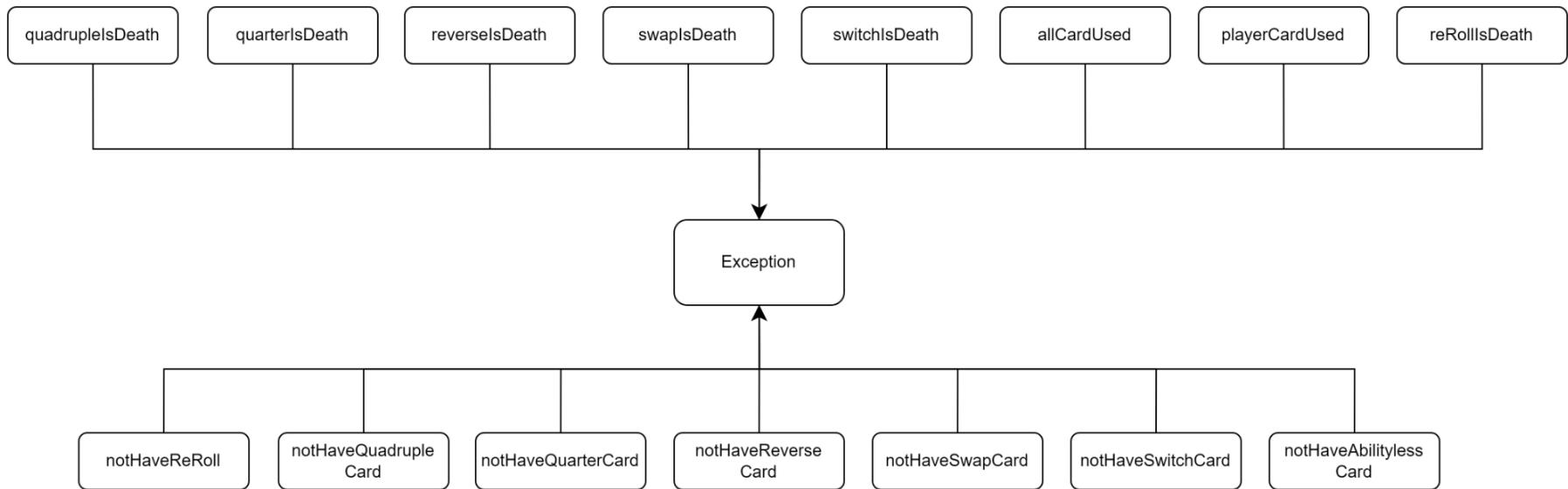
1. AbilityCard
2. Card
3. Calculable
4. Combination
5. Exception
6. Action
7. Player
8. SetGame
9. Splash
10. Table

1.1 Hierarki Kelas

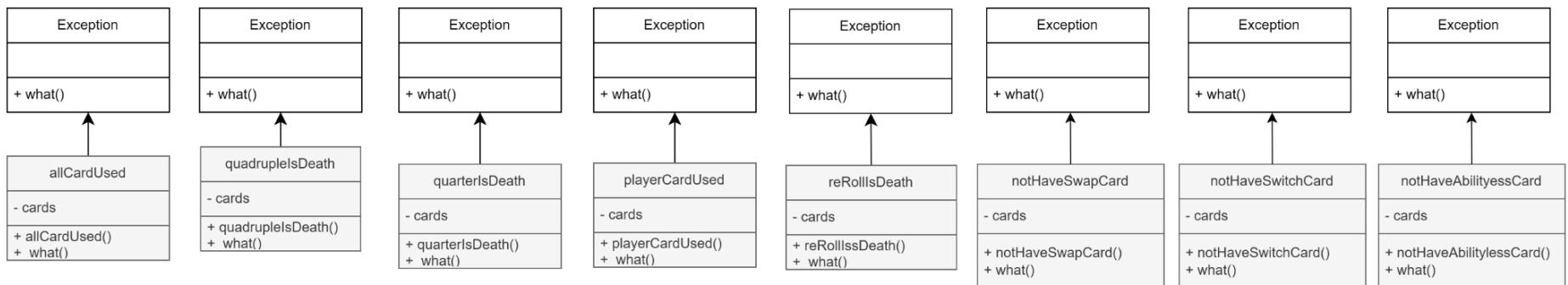
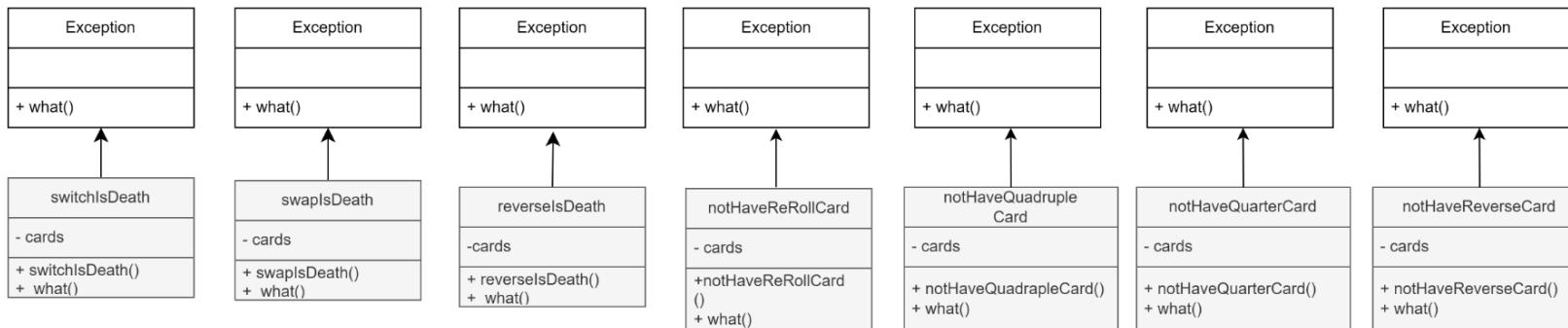




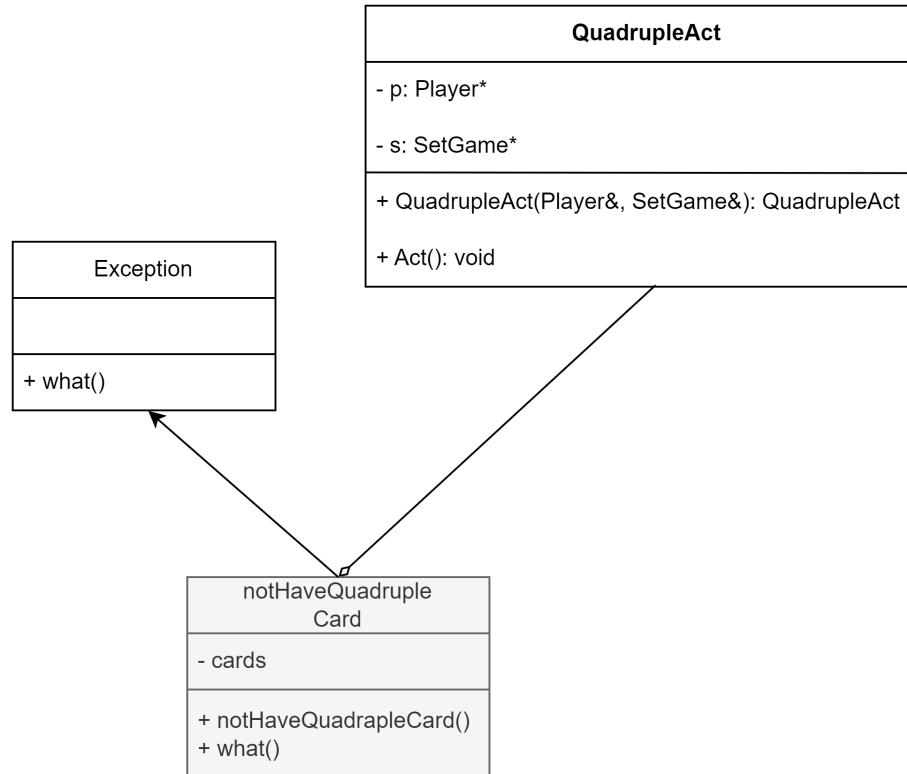




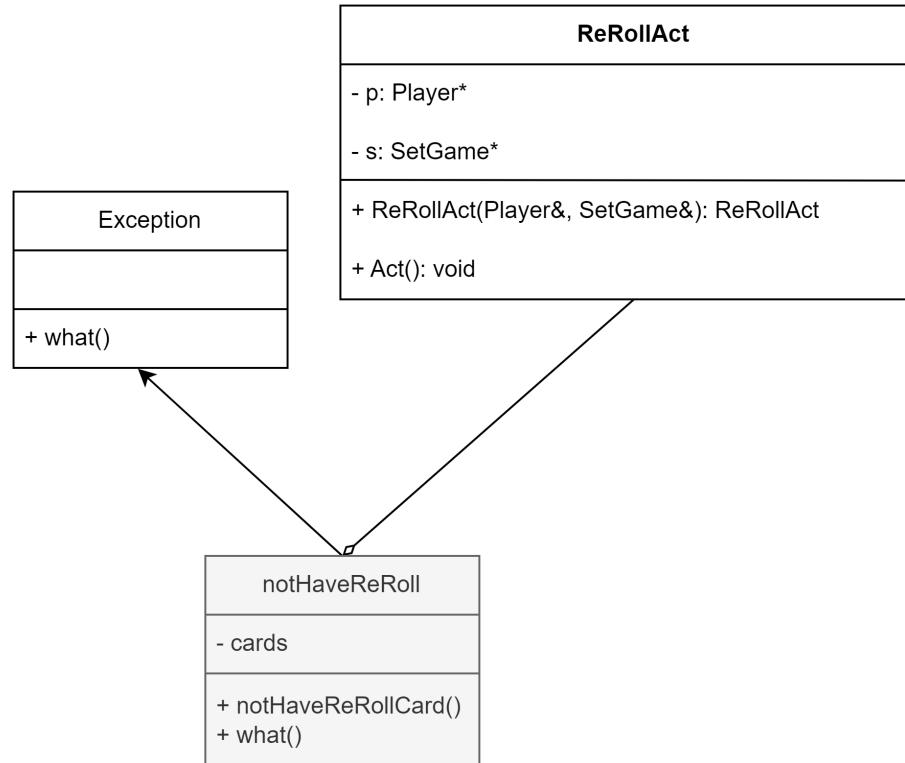
1.2 Kelas Exception dan Turunannya



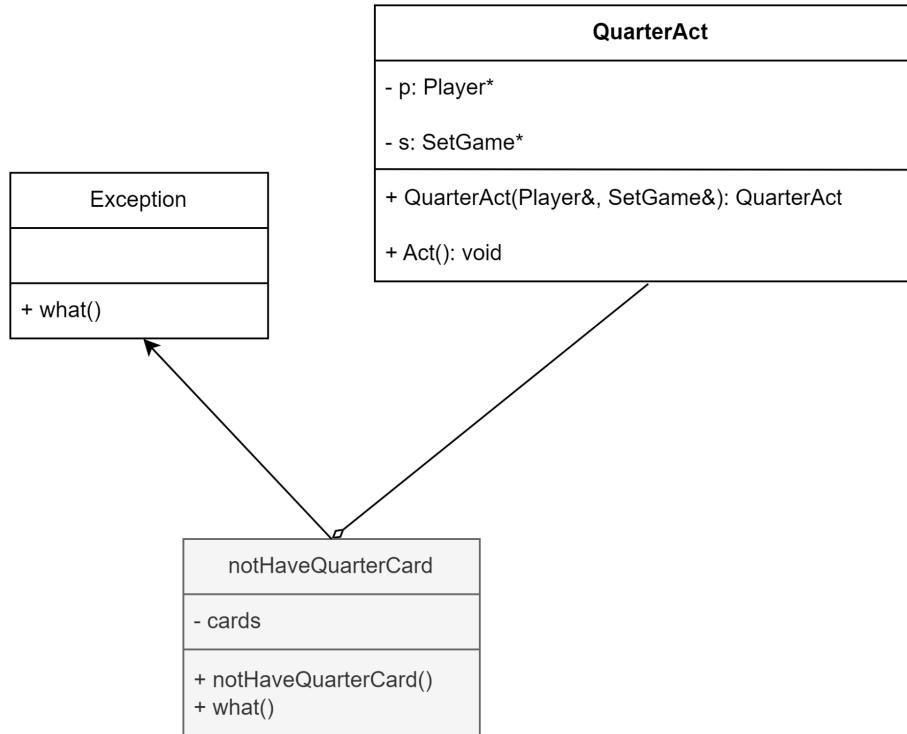
1.3 Kelas notHaveQuadrupleCard



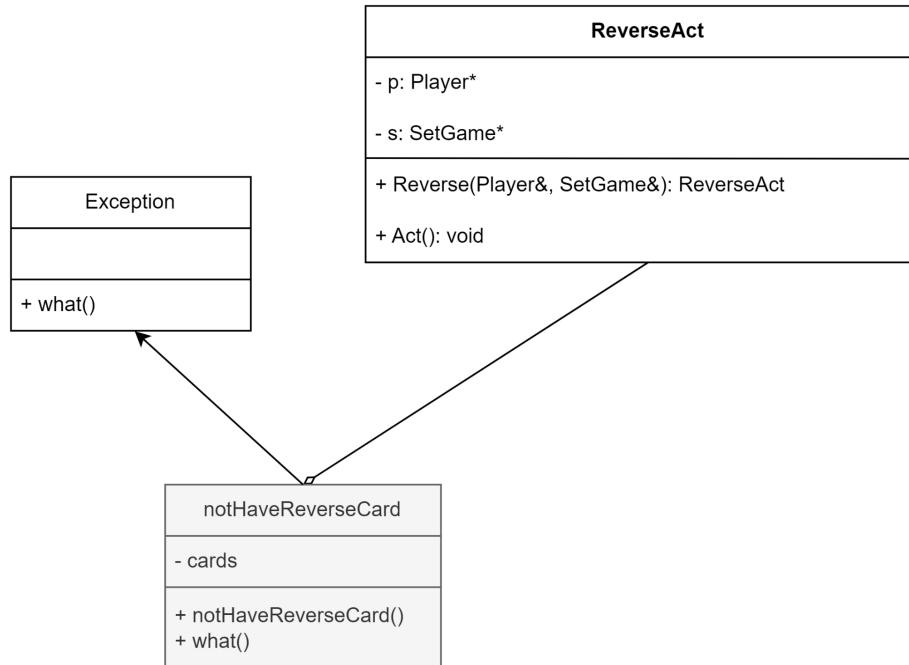
1.4 Kelas notHaveReRoll



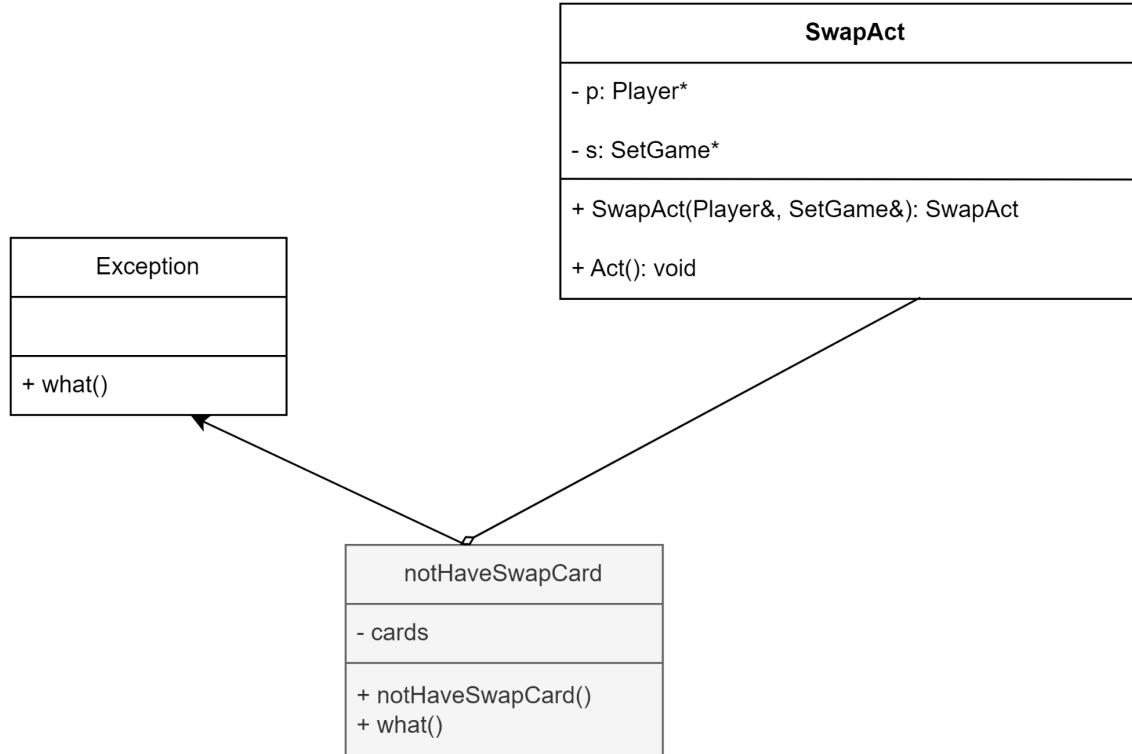
1.5 Kelas notHaveQuarterCard



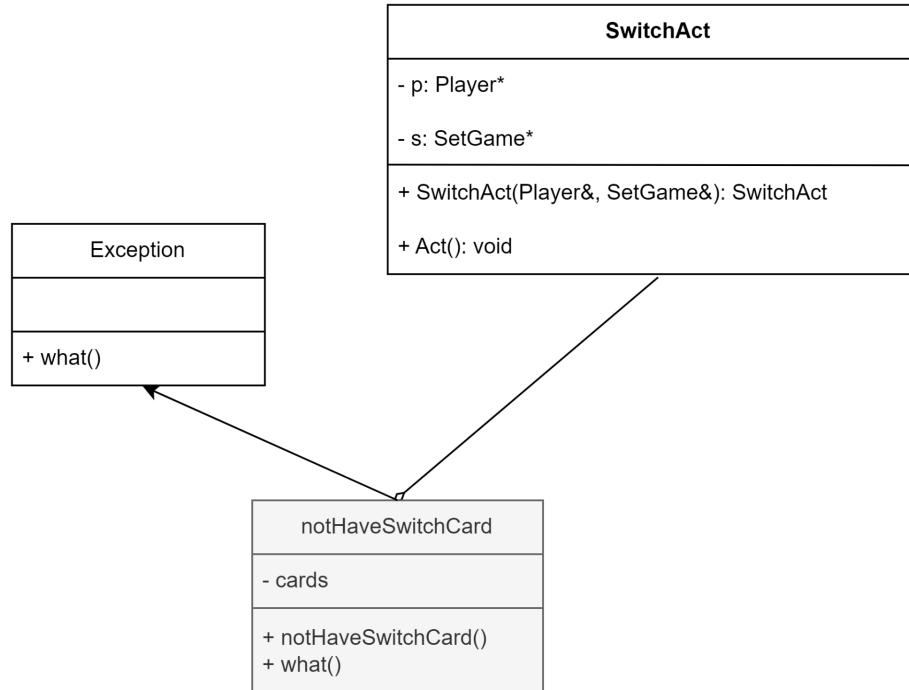
1.6 Kelas notHaveReverseCard



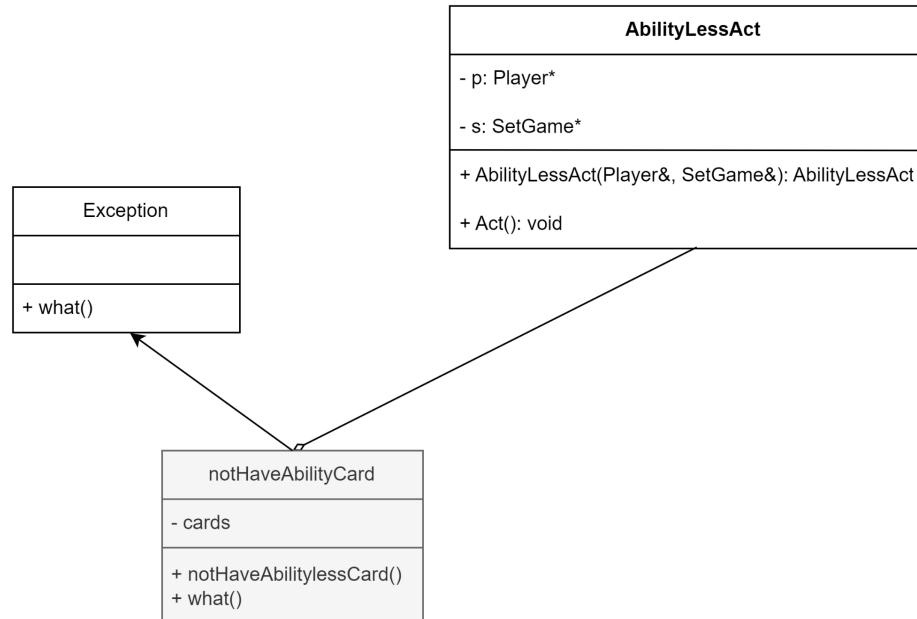
1.7 Kelas notHaveSwapCard



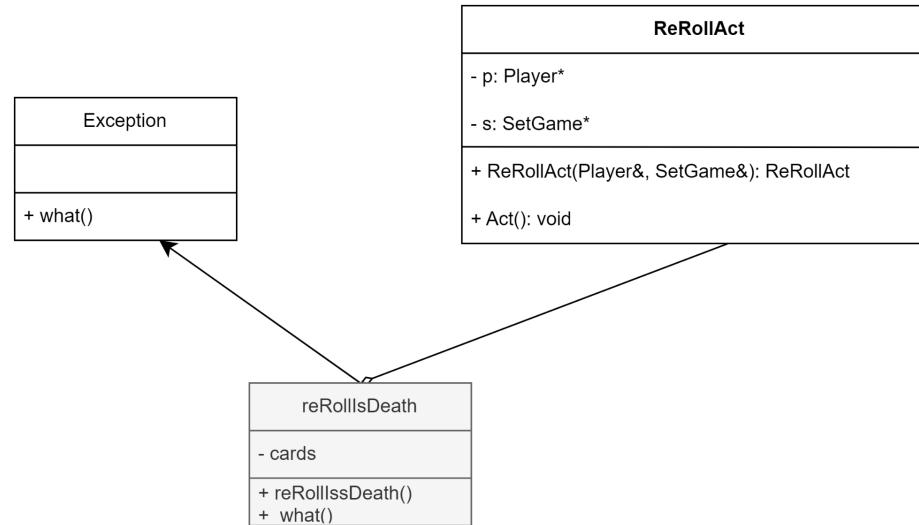
1.8 Kelas notHaveSwitchCard



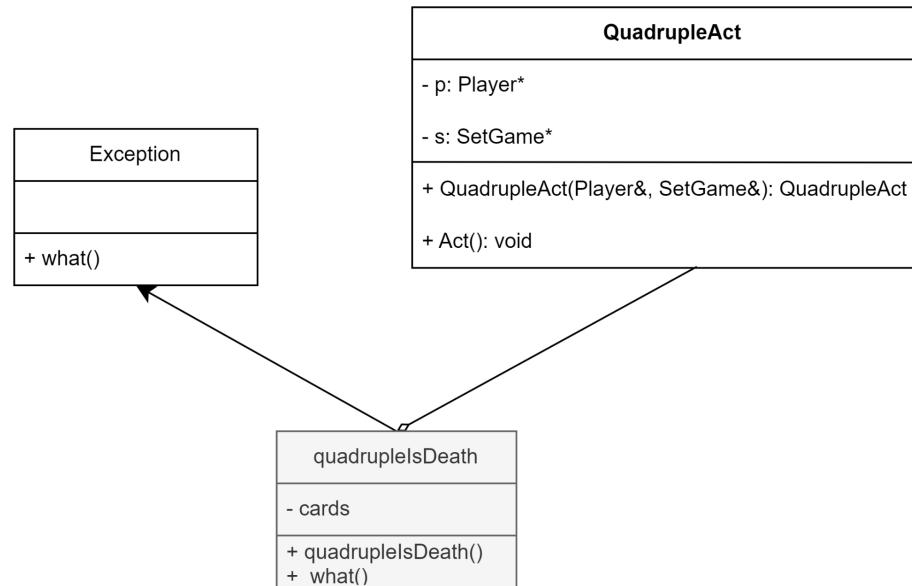
1.9 Kelas notHaveAbilityCard



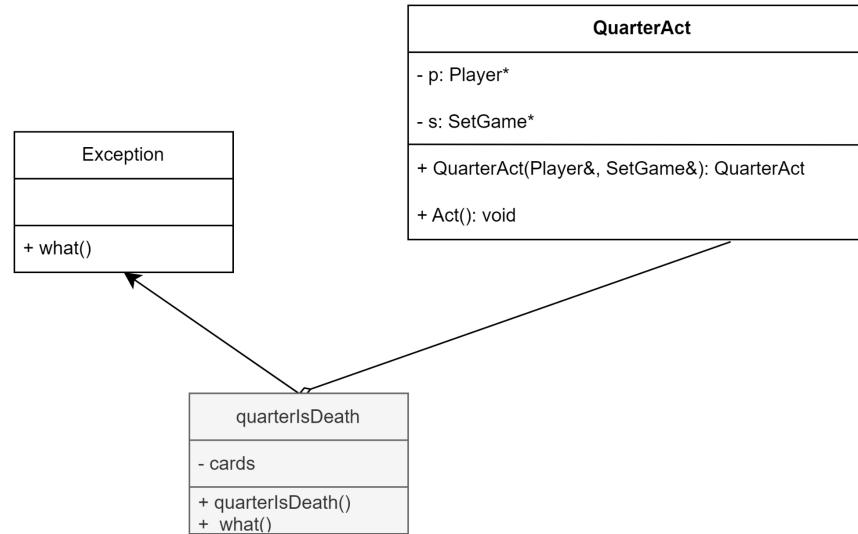
1.10 Kelas reRollIsDeath



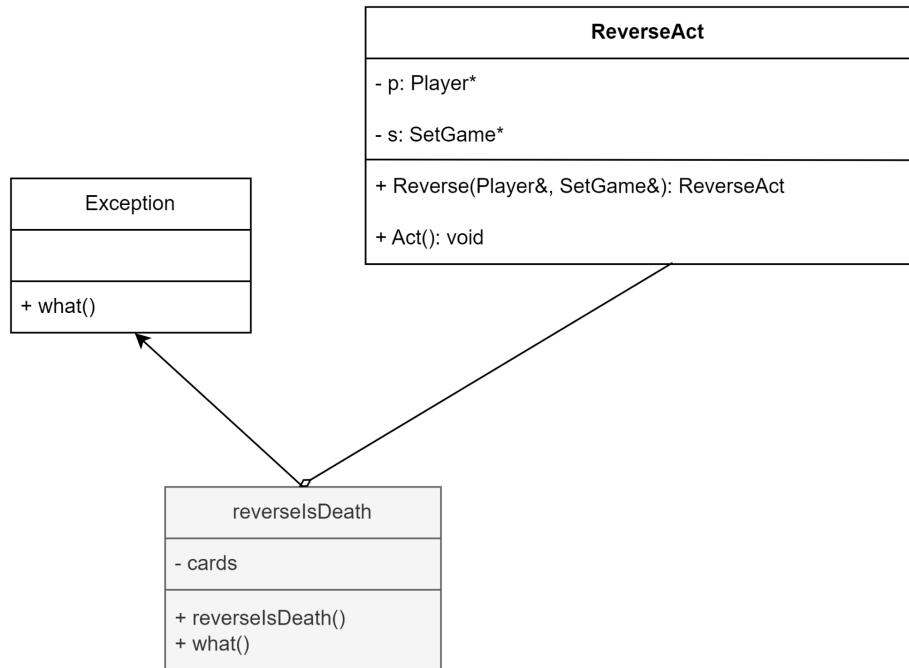
1.11 Kelas quadruplesDeath



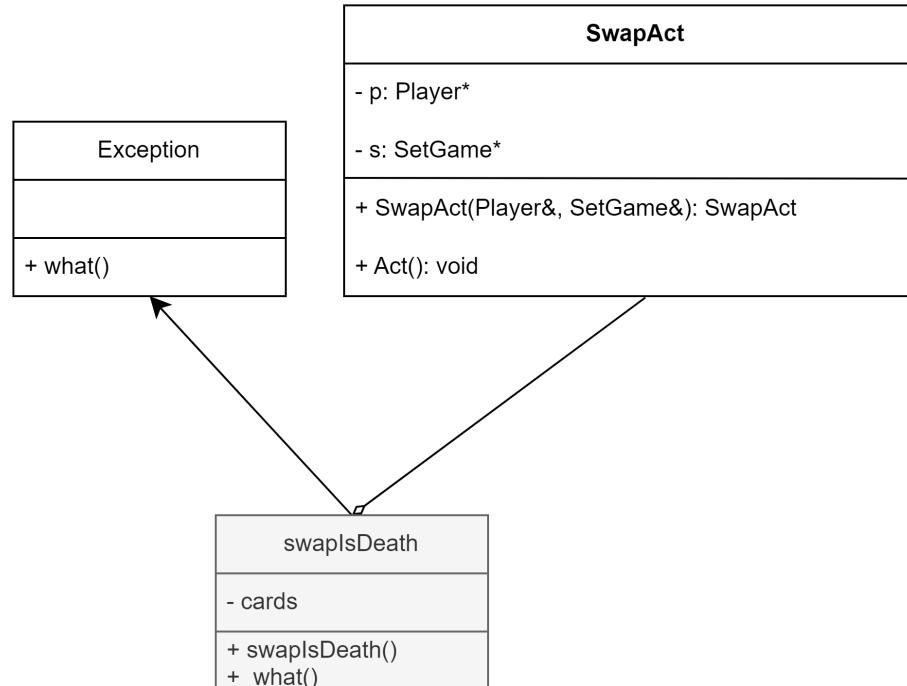
1.12 Kelas quarterIsDeath



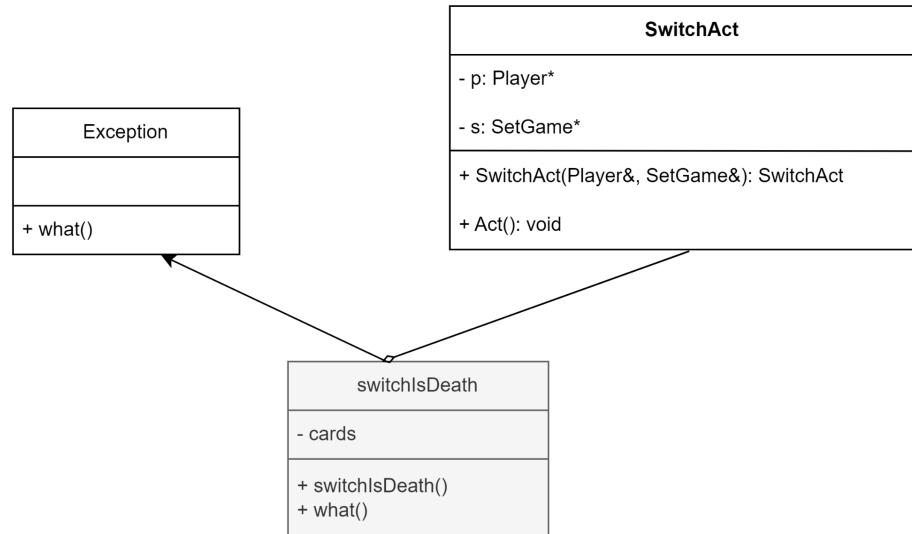
1.13 Kelas reverselsDeath



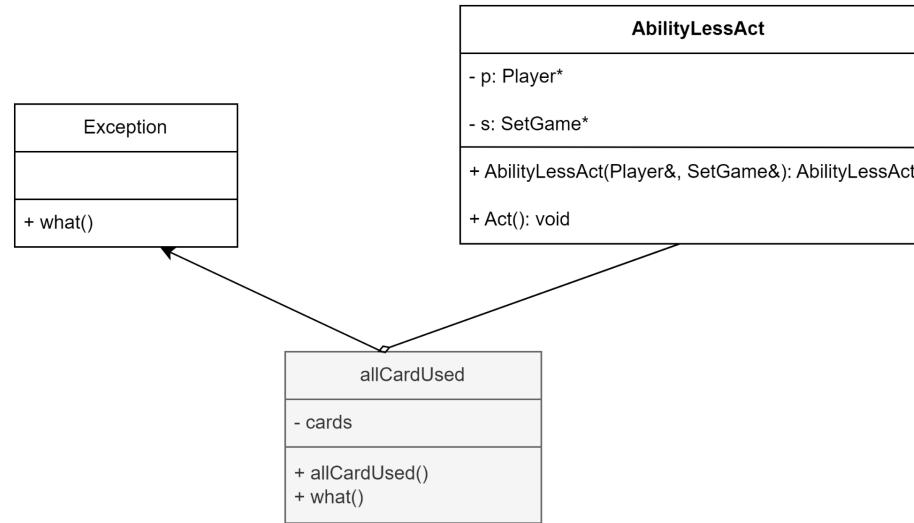
1.14 Kelas swapIsDeath



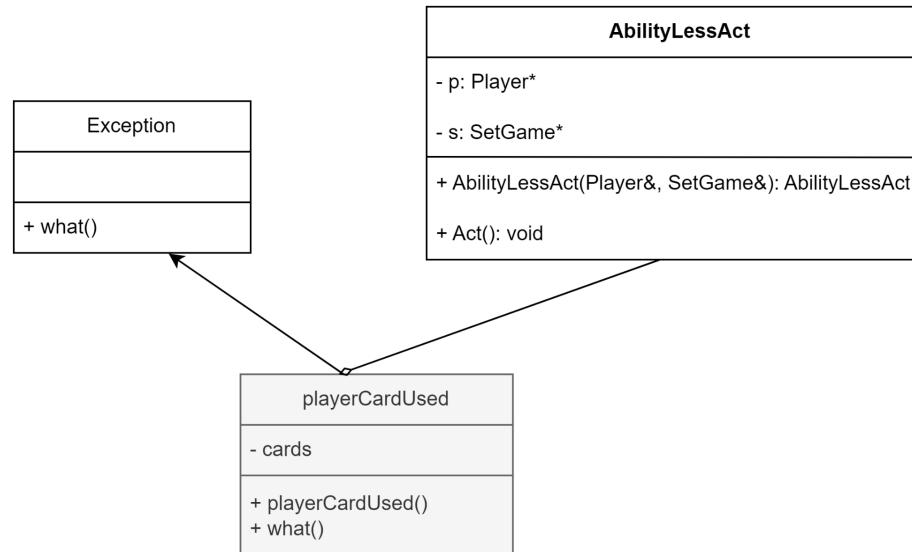
1.15 Kelas switchIsDeath



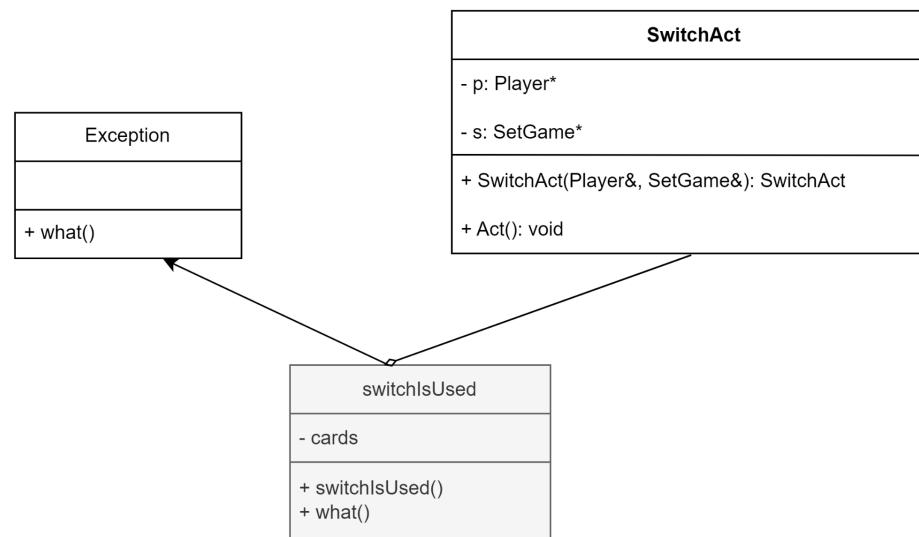
1.16 Kelas allCardUsed



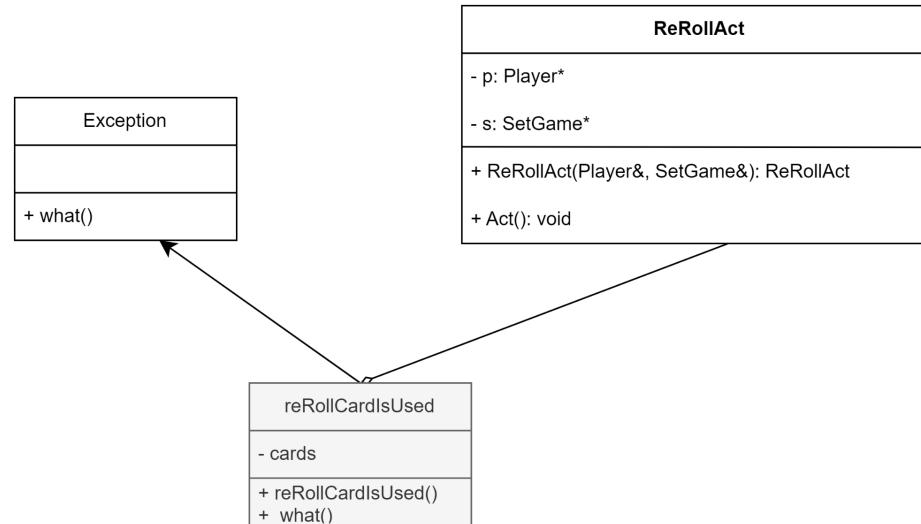
1.17 Kelas playerCardUsed



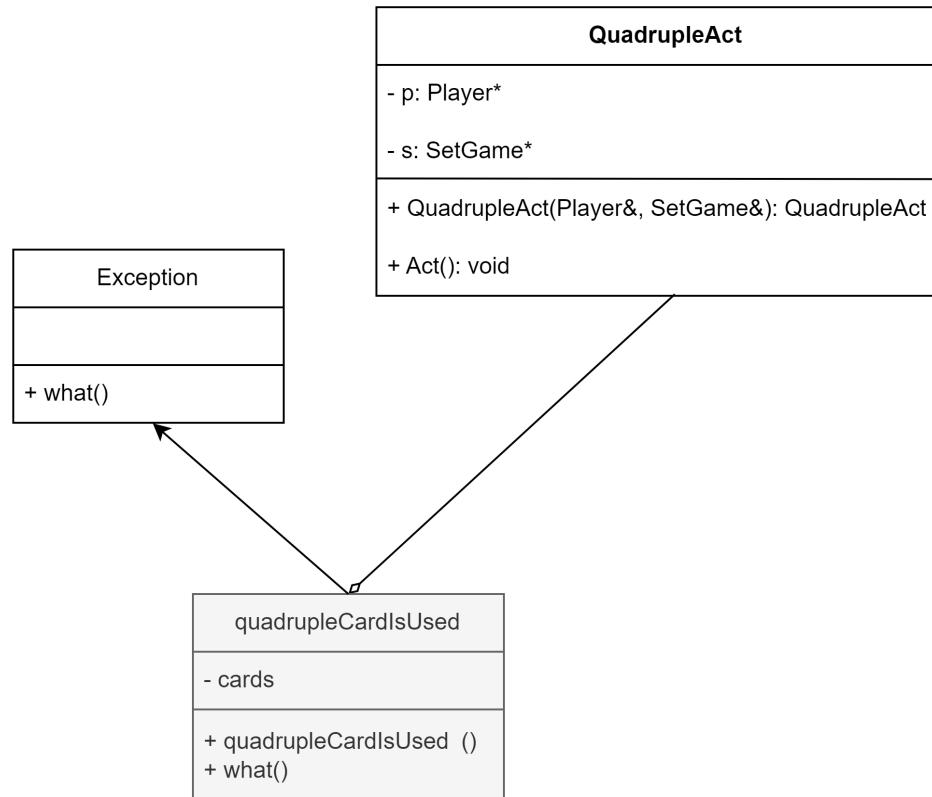
1.18 Kelas switchIsUsed



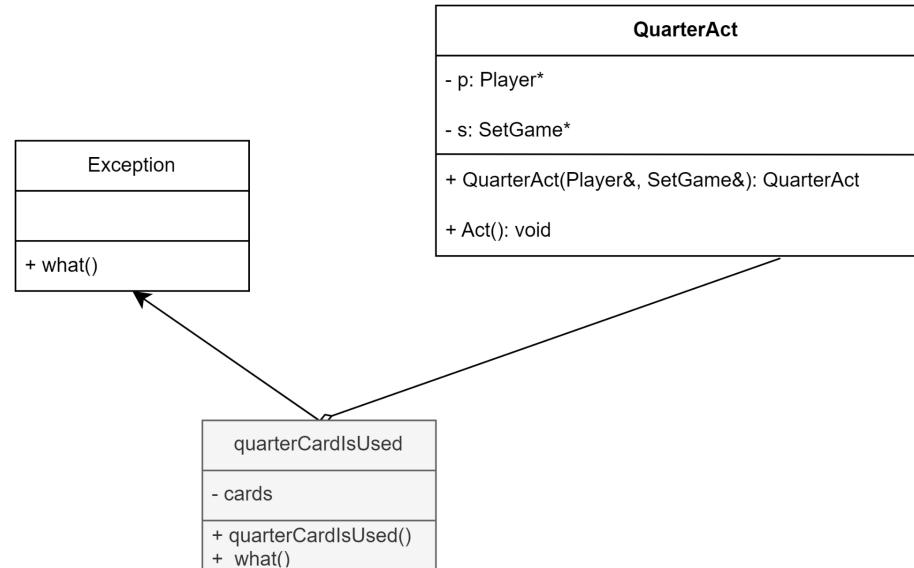
1.19 Kelas reRollIsUsed



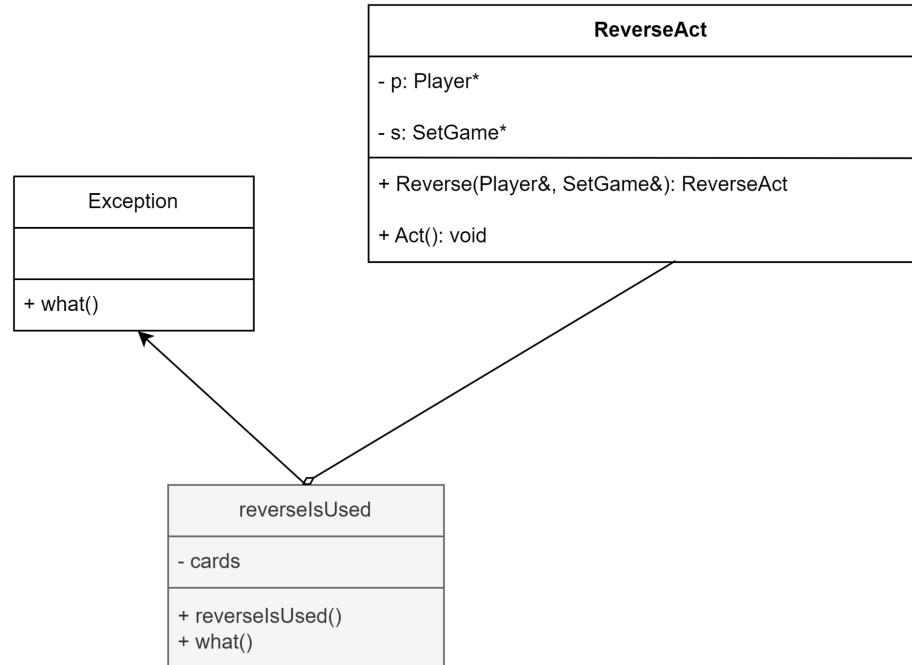
1.20 Kelas quadrupleCardIsUsed



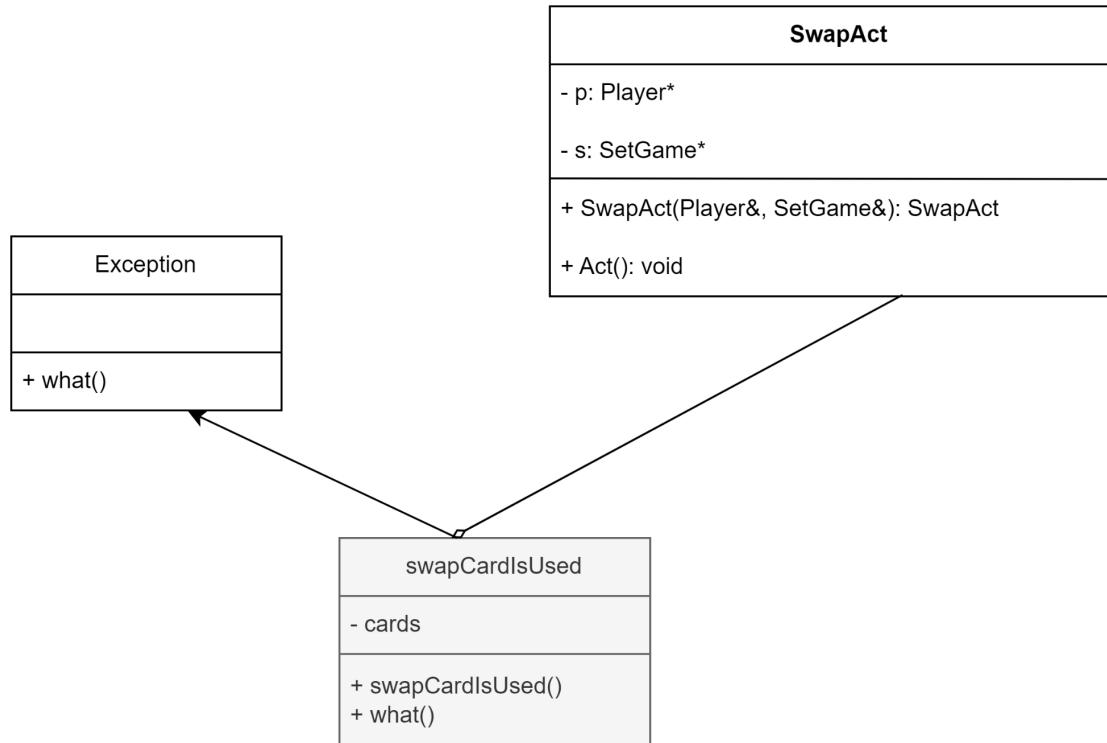
1.21 Kelas quarterCardIsUsed



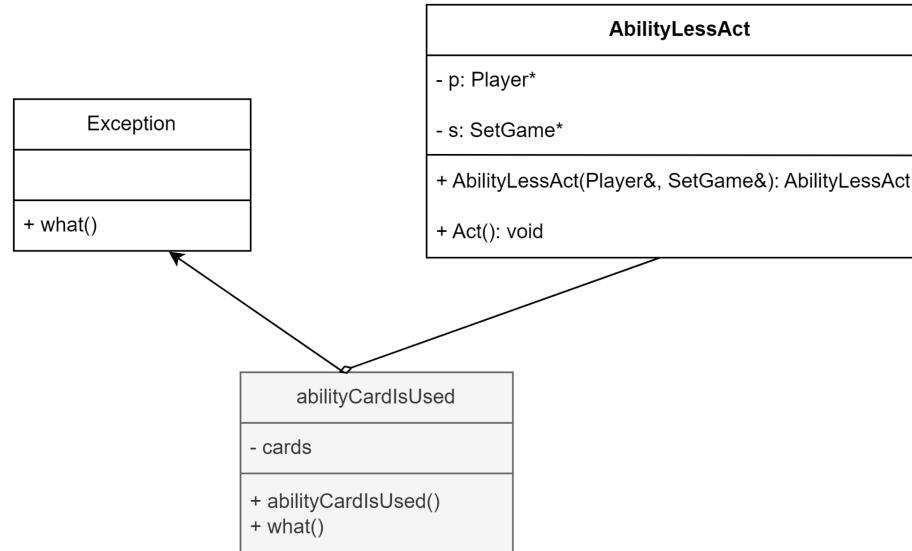
1.22 Kelas reverseCardIsUsed



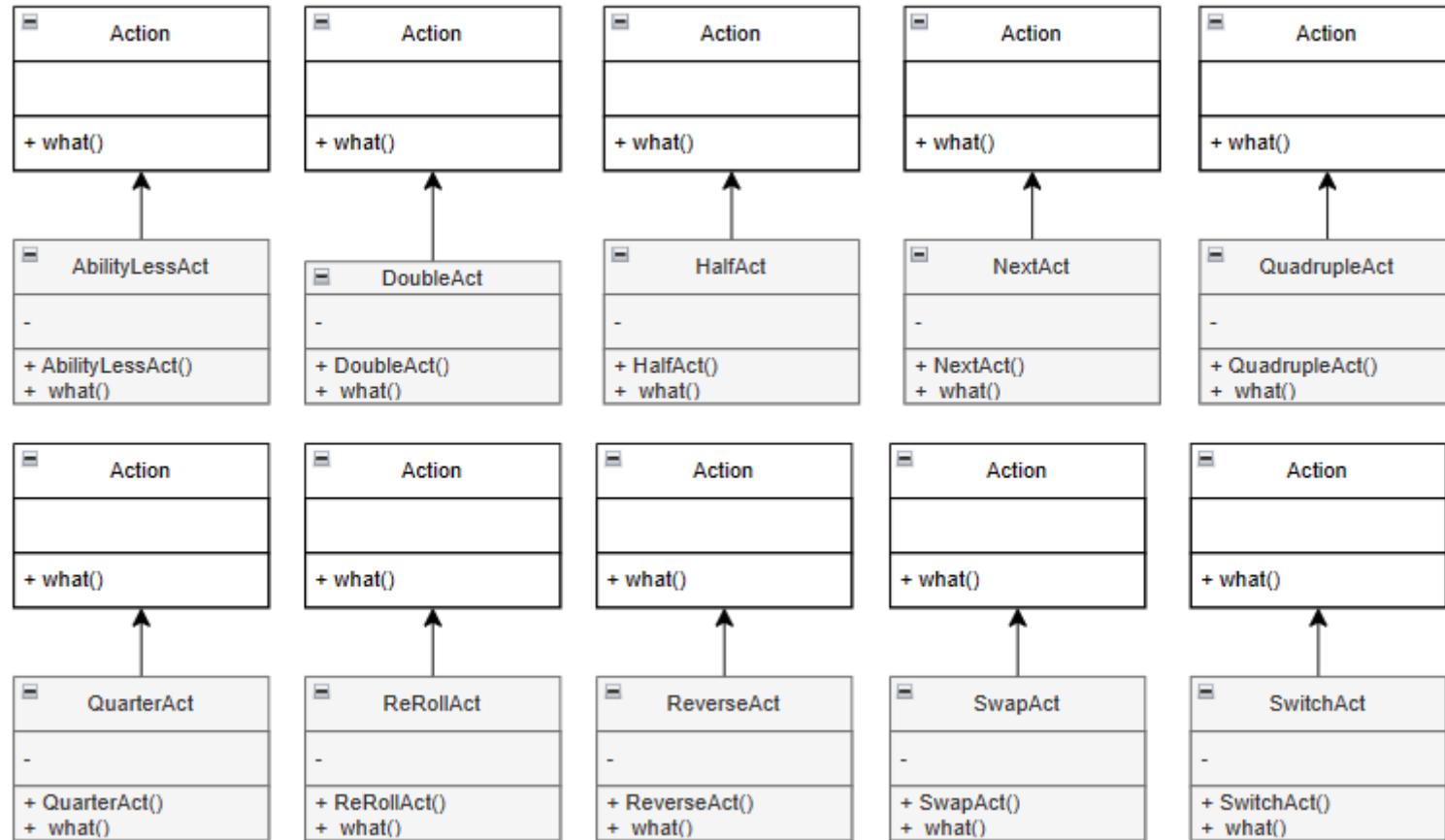
1.23 Kelas swapCardIsUsed



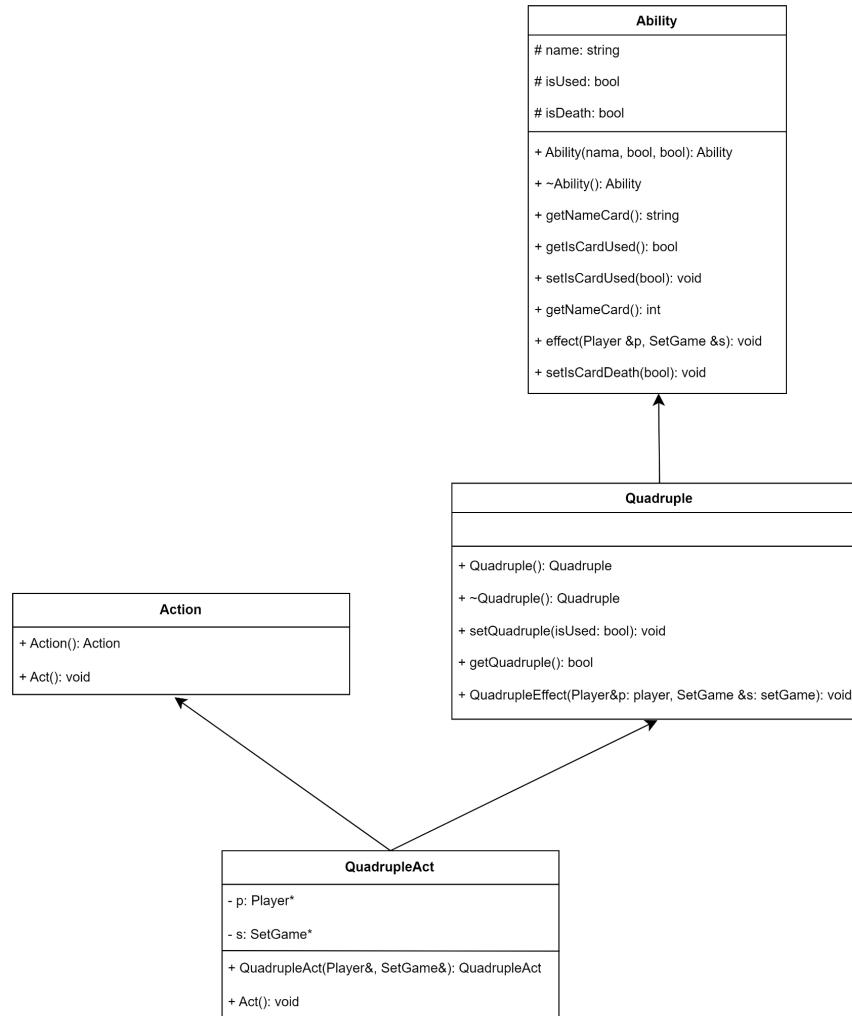
1.24 Kelas abilityCardIsUsed



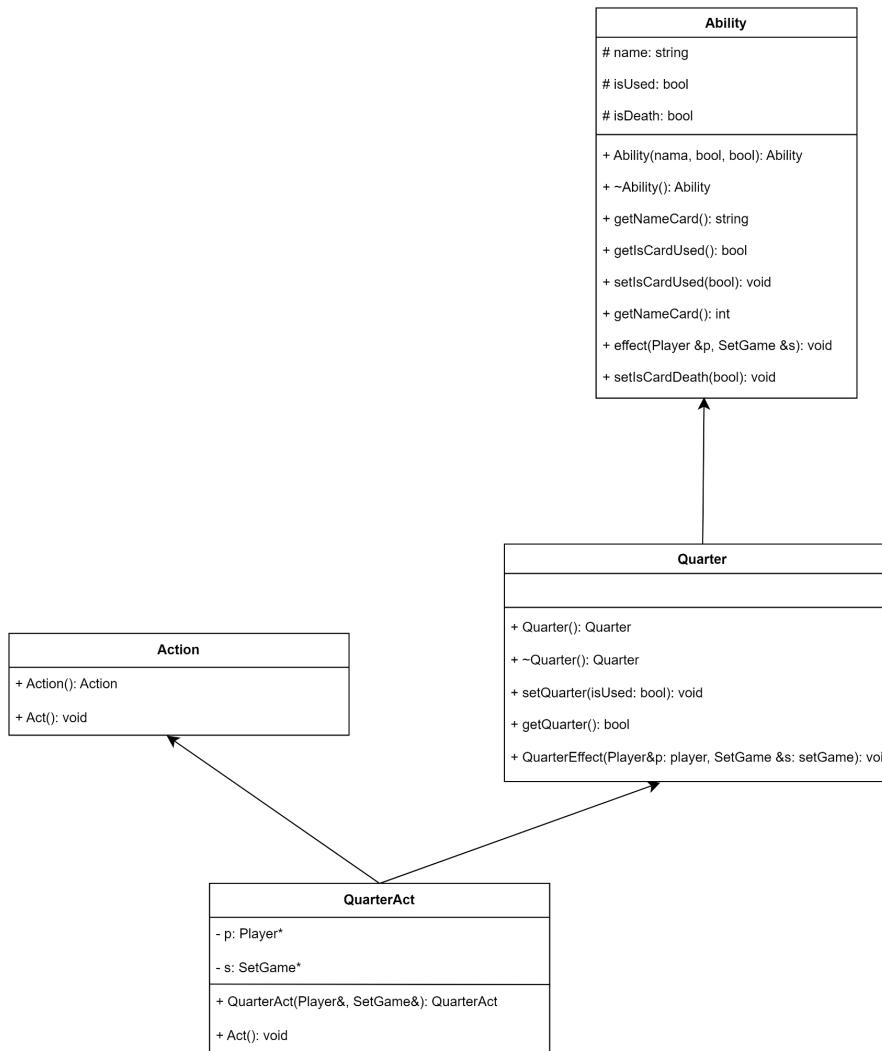
1.25 Kelas Action dan Turunannya



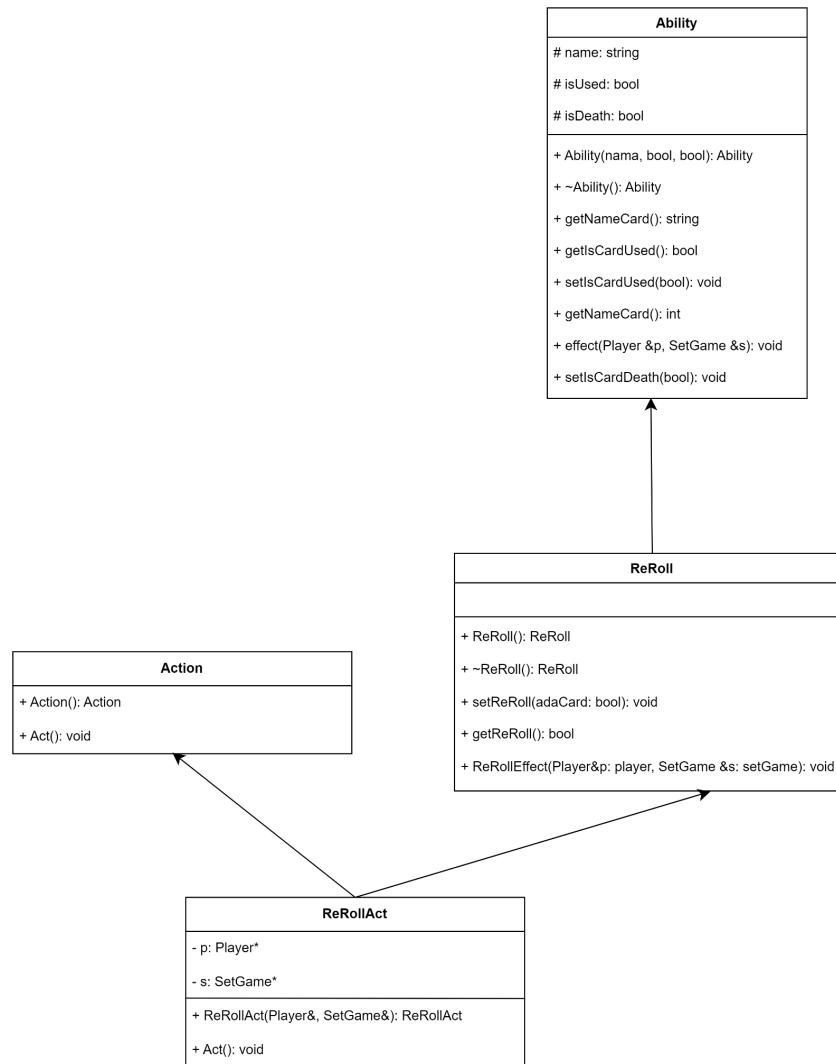
1.26 Kelas QuadrupleAct



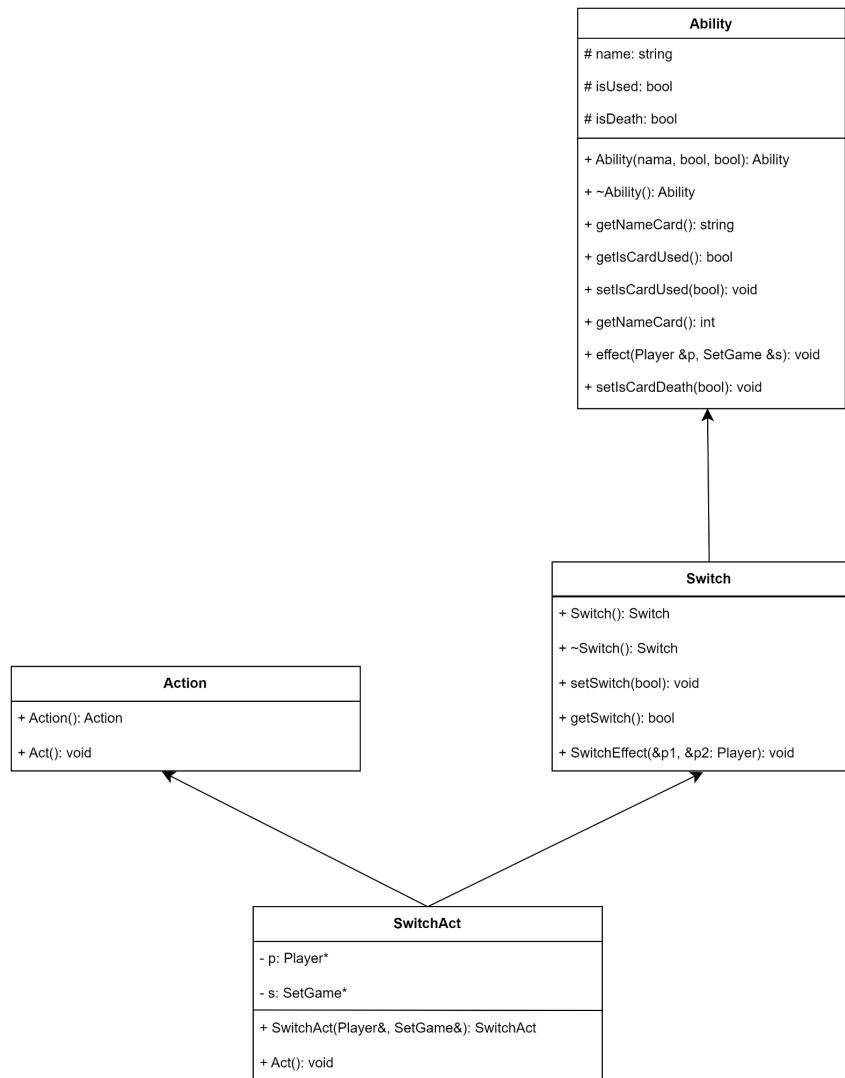
1.27 Kelas QuarterAct



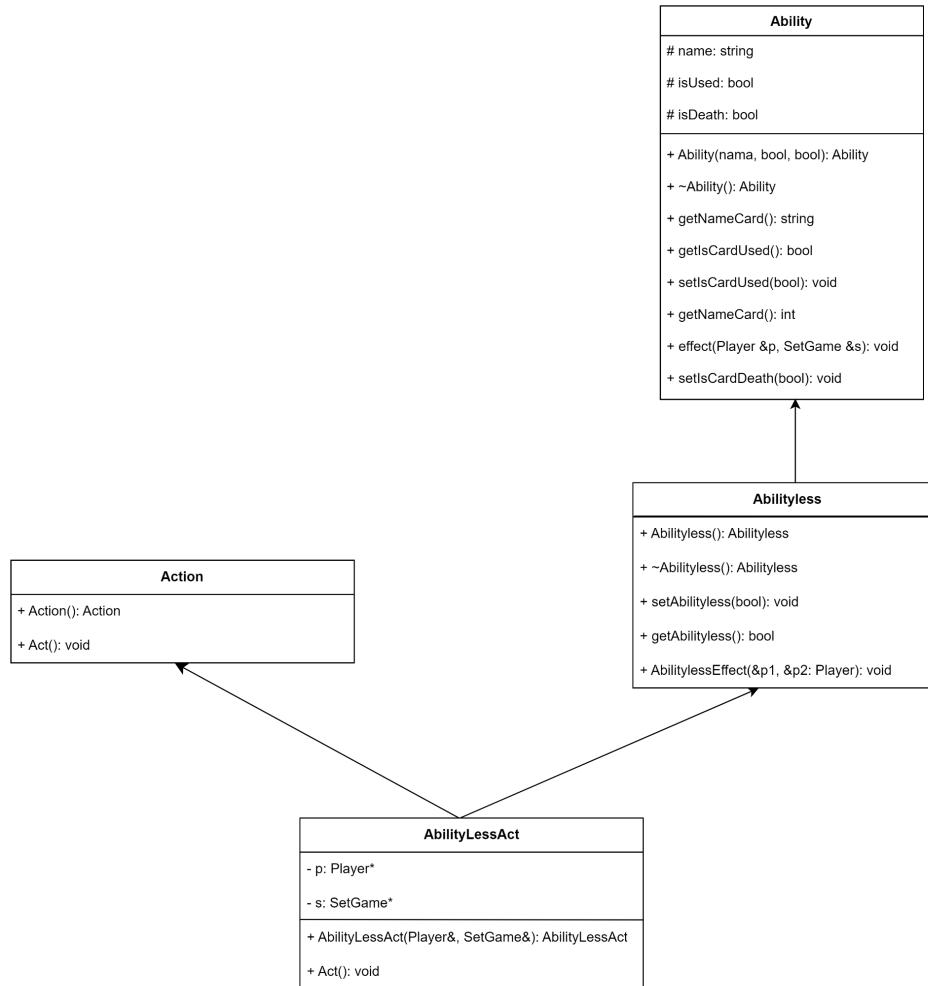
1.28 Kelas ReRollAct



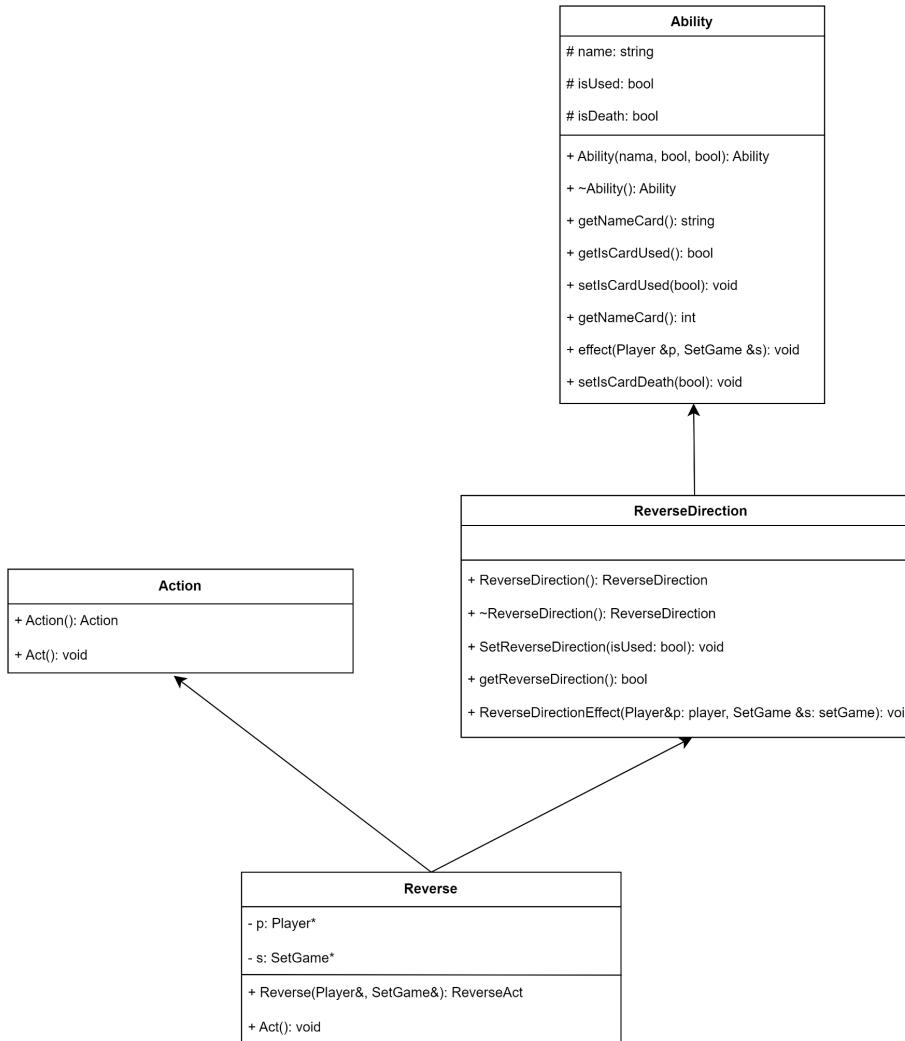
1.29 Kelas SwitchAct



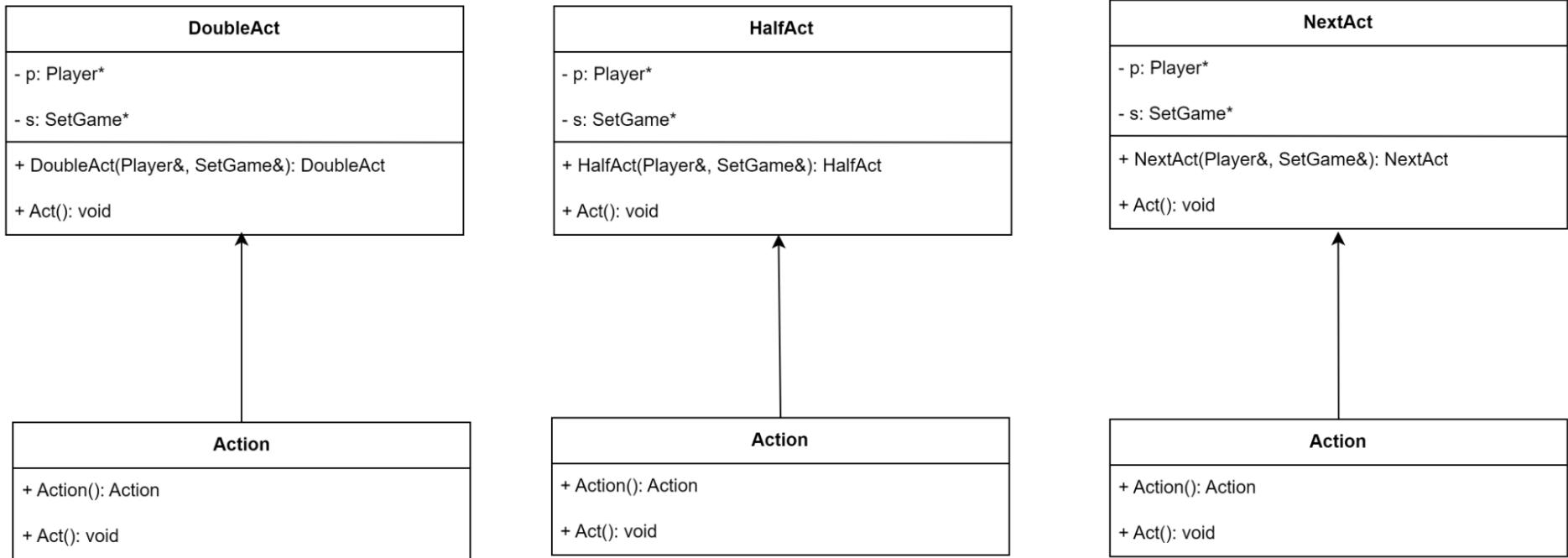
1.30 Kelas AbilityLessAct



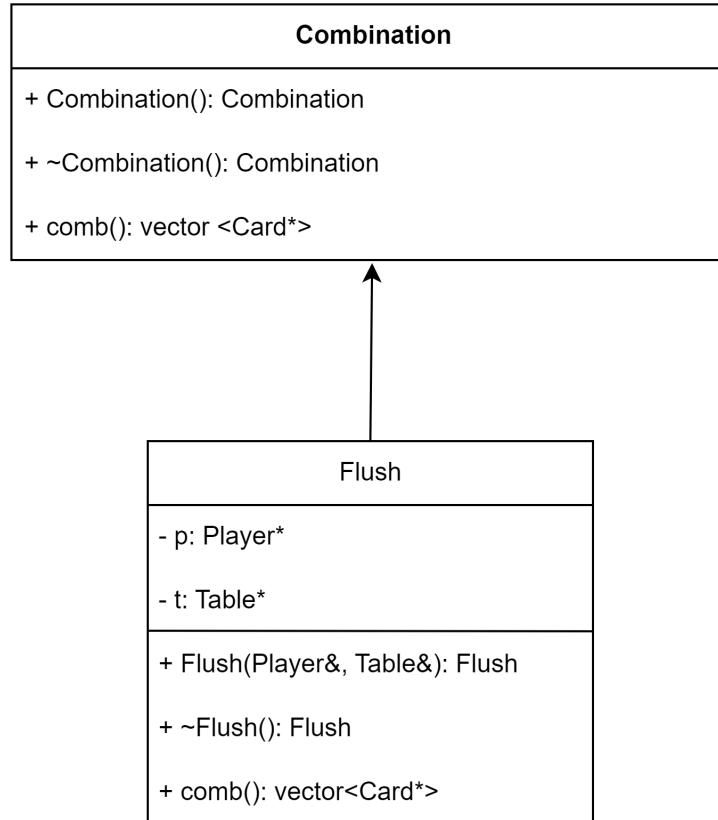
1.31 Kelas ReverseAct



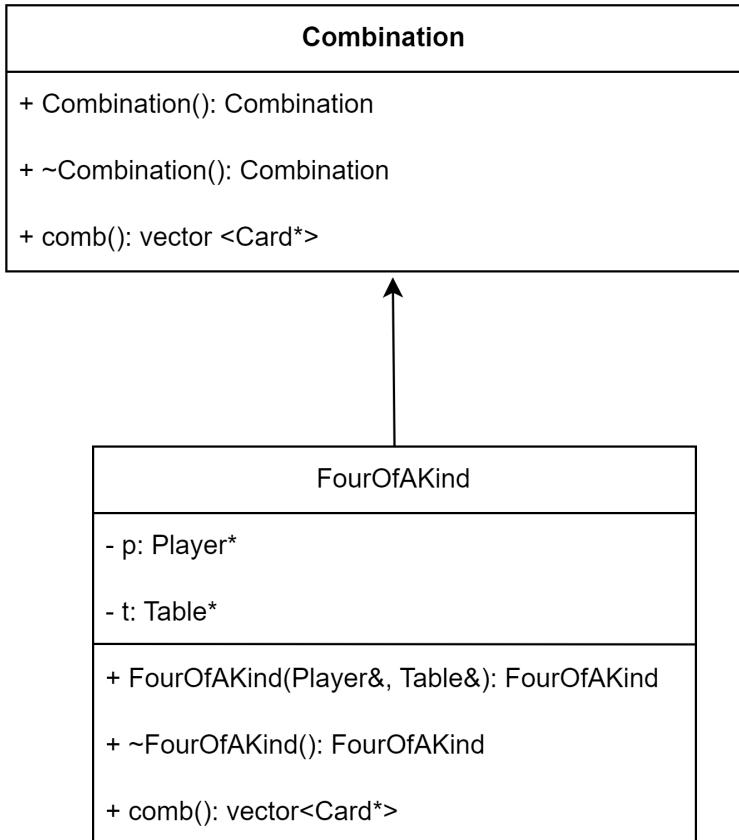
1.32 Kelas DoubleAct, HalfAct, NextAct



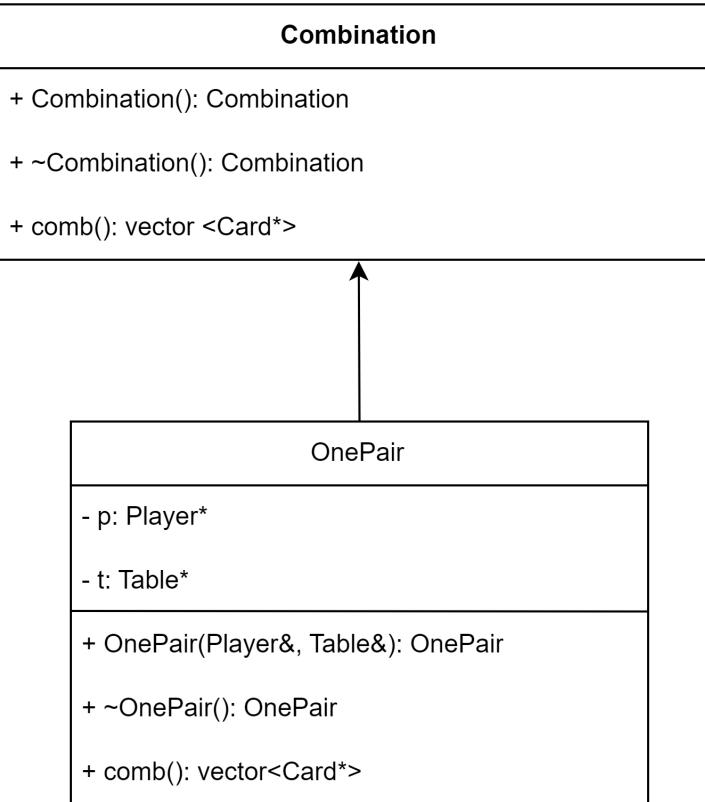
1.33 Kelas Flush dengan Parent Combination



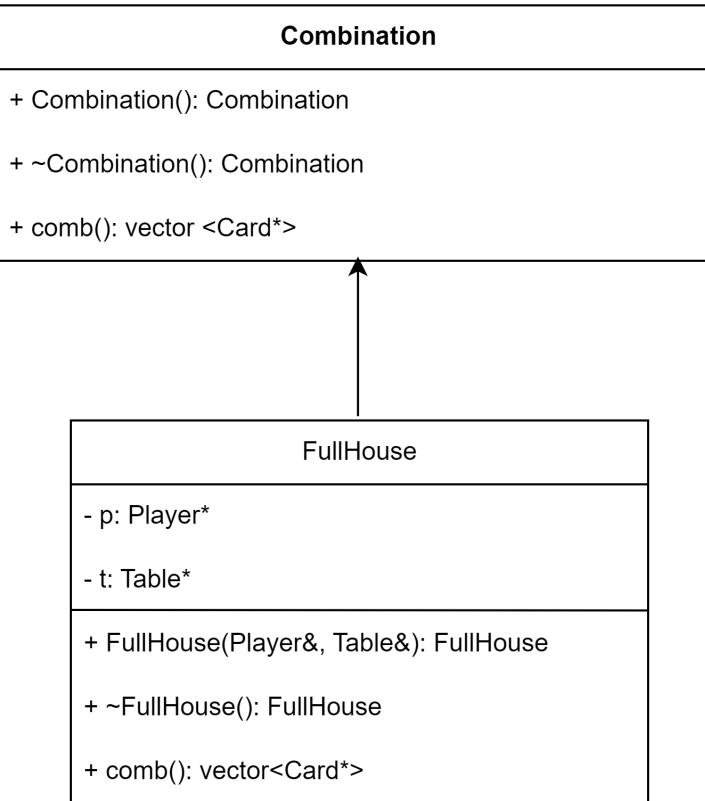
1.34 Kelas FourOfAKind dengan Parent Combination



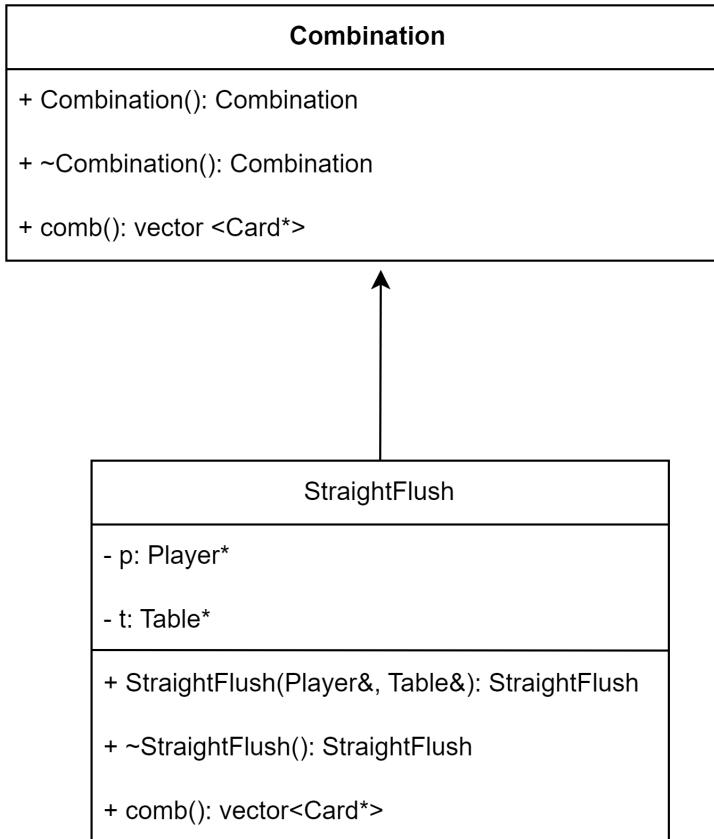
1.35 Kelas OnePair dengan Parent Combination



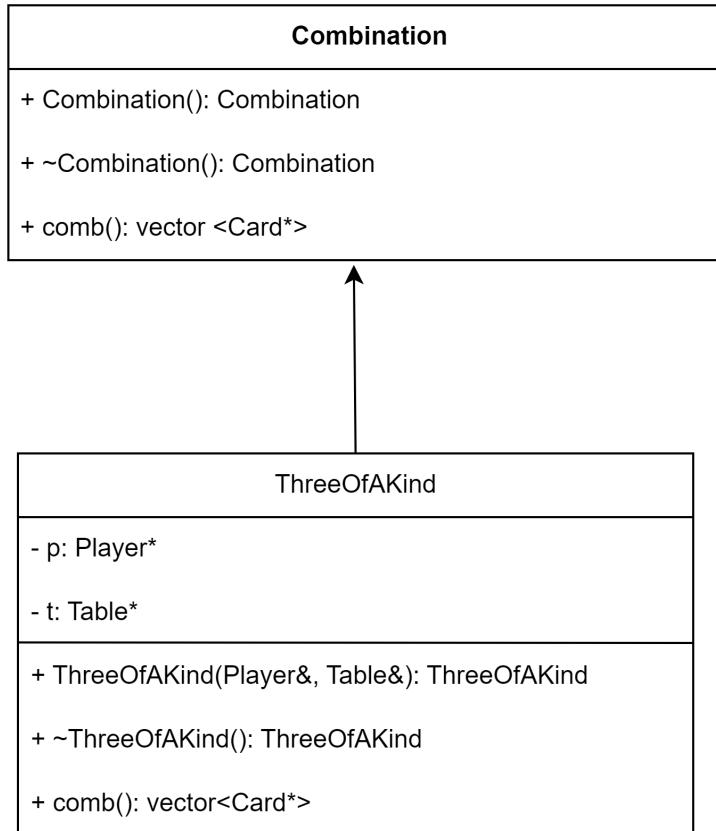
1.36 Kelas FullHouse dengan Parent Combination



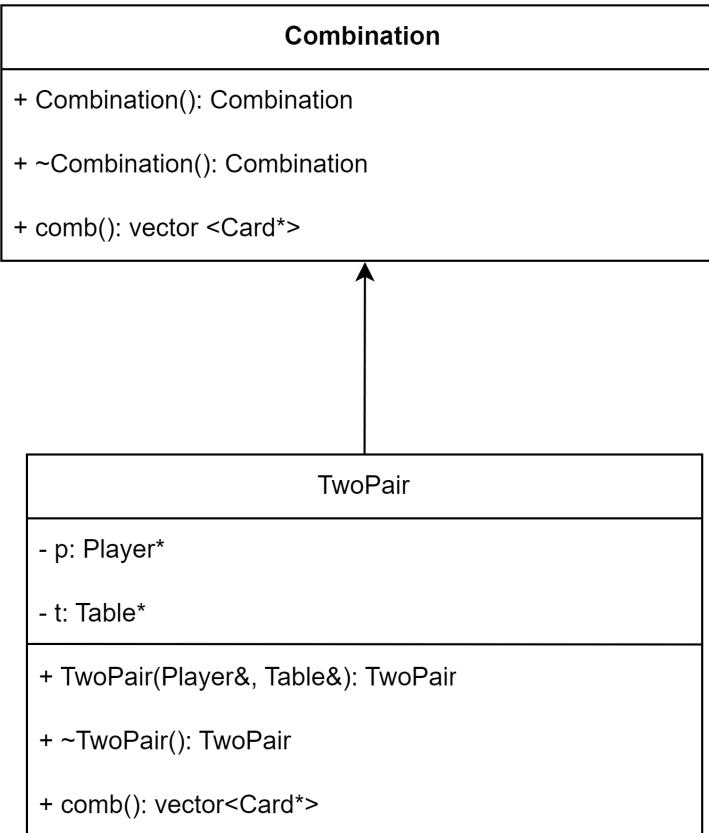
1.37 Kelas StraightFlush dengan Parent Combination



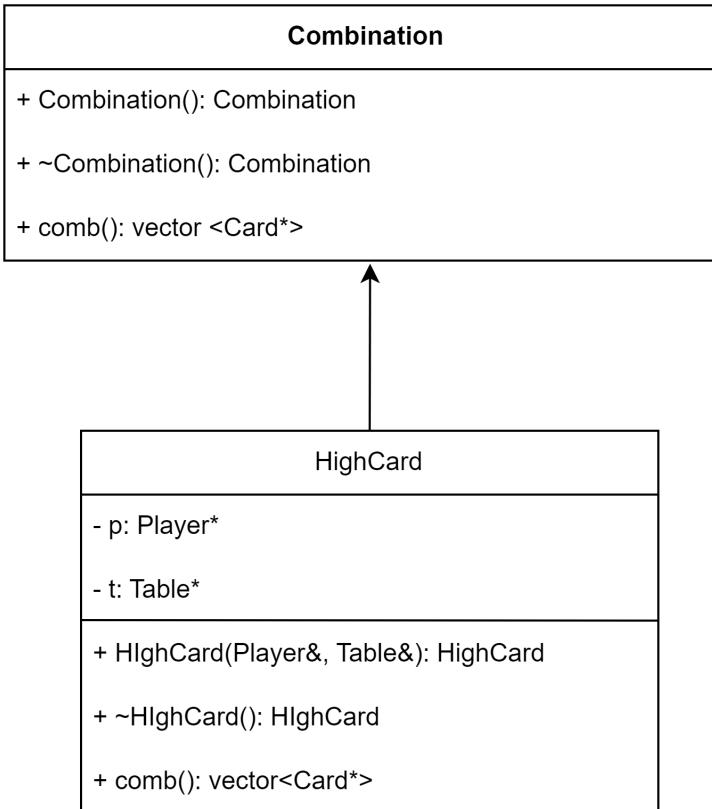
1.38 Kelas ThreeOfAKind dengan Parent Combination



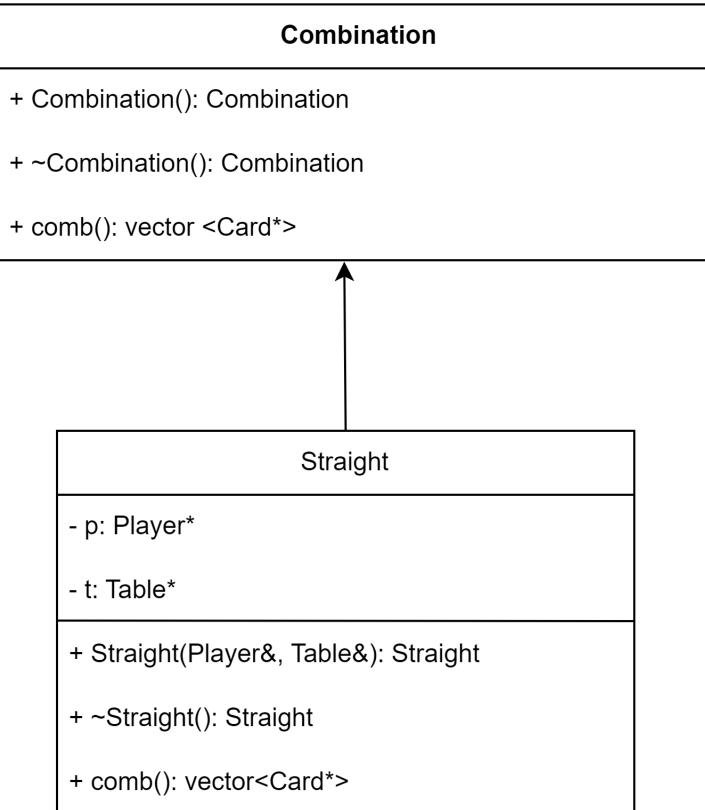
1.39 Kelas TwoPair dengan Parent Combination



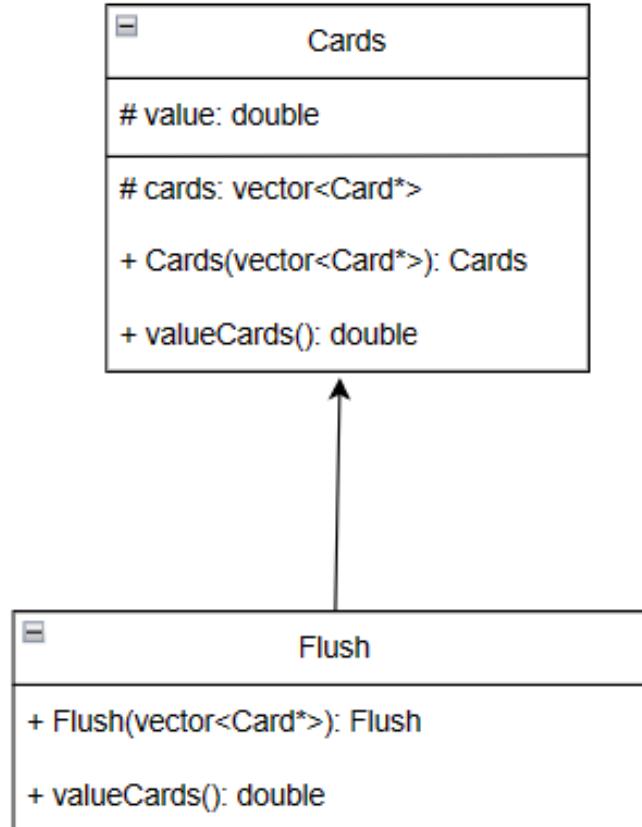
1.40 Kelas HighCard dengan Parent Combination



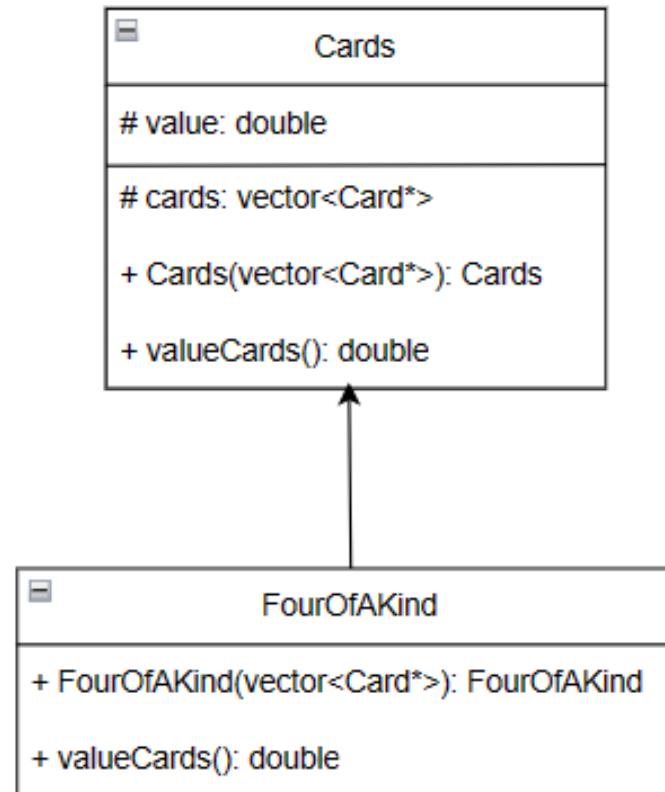
1.41 Kelas Straight dengan Parent Combination



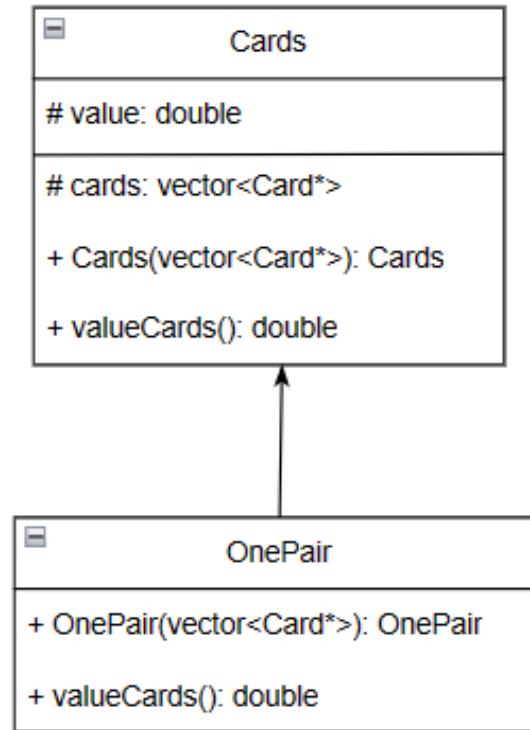
1.42 Kelas Flush dengan Parent Cards



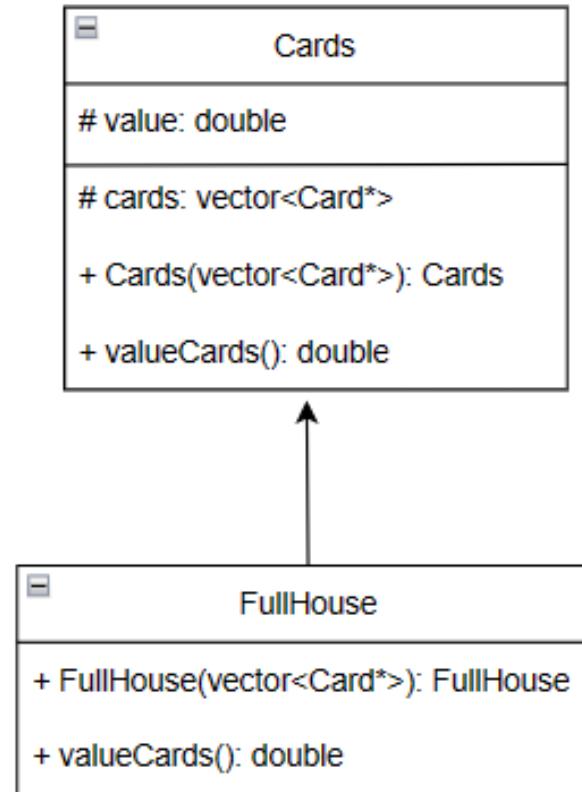
1.43 Kelas FourOfAKind dengan Parent Cards



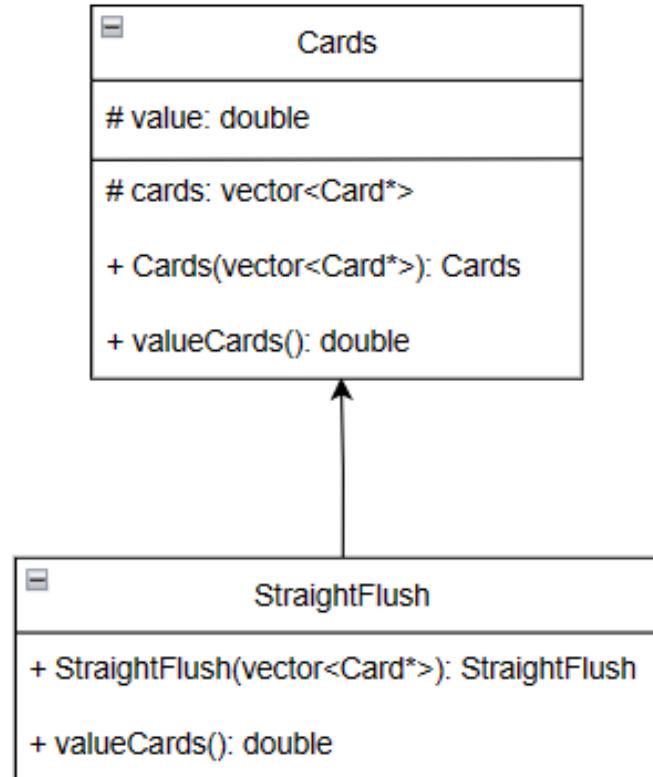
1.44 Kelas OnePair dengan Parent Cards



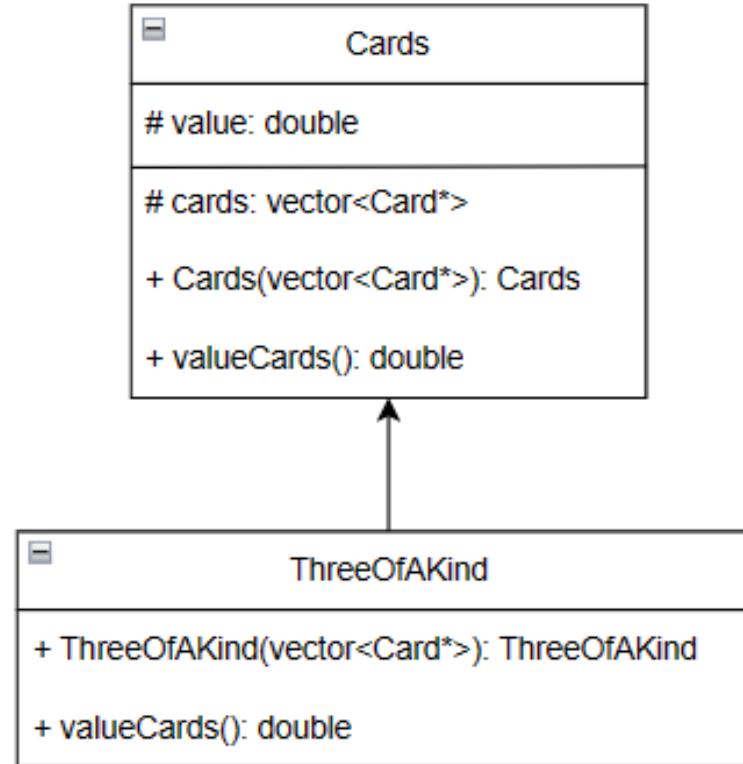
1.45 Kelas FullHouse dengan Parent Cards



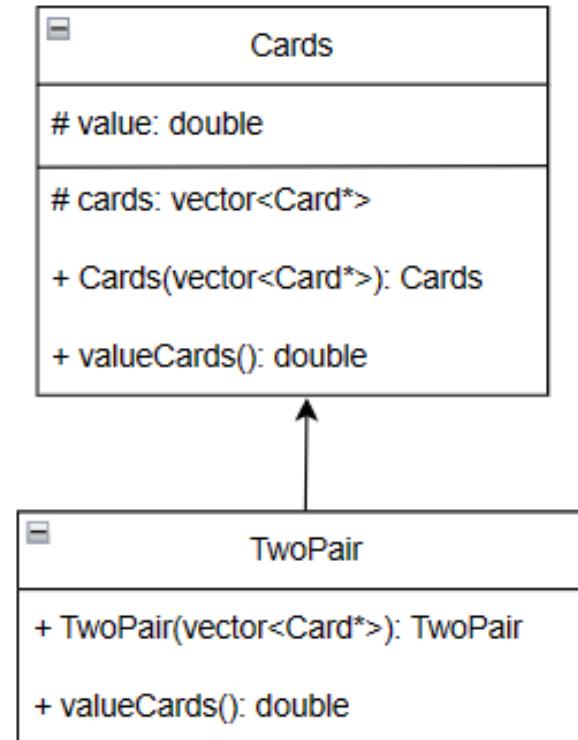
1.46 Kelas StraightFlush dengan Parent Cards



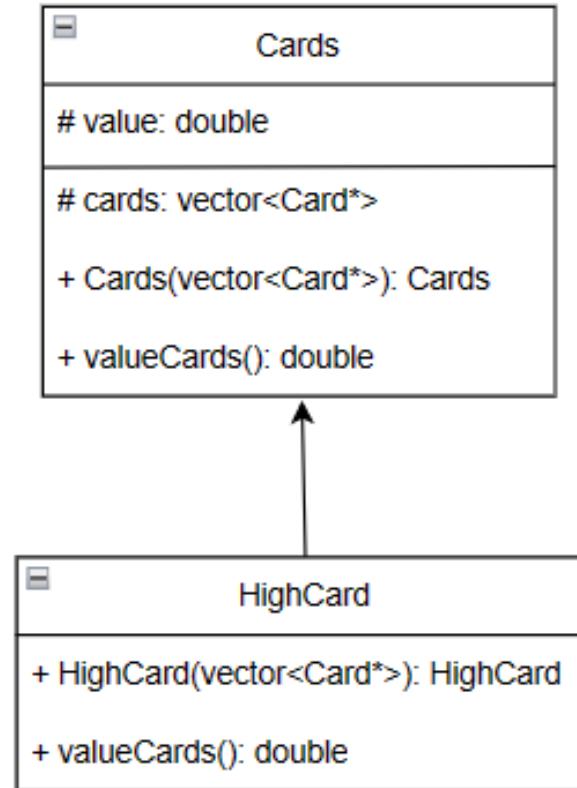
1.47 Kelas ThreeOfAKind dengan Parent Cards



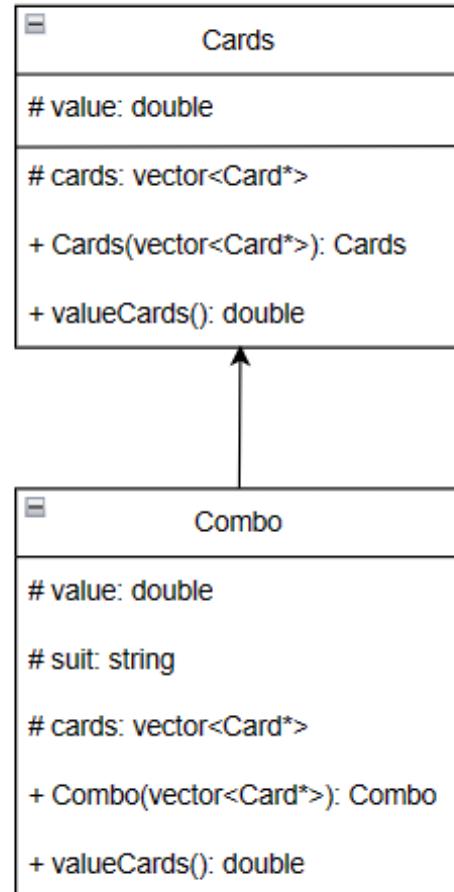
1.48 Kelas TwoPair dengan Parent Cards



1.49 Kelas HighCard dengan Parent Cards



1.50 Kelas Combo dengan Parent Cards



2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Inheritance adalah konsep dimana sebuah kelas dapat mewarisi atribut dan metode dari kelas lain yang disebut *superclass* (kelas induk). Kelas yang mewarisi atribut dan metode dari *superclass* disebut *subclass* (kelas turunan). Dalam inheritance, *subclass* memiliki semua atribut dan metode dari *superclass* serta bisa menambahkan atribut atau metode baru. Konsep inheritance memungkinkan untuk membangun hierarki kelas yang terstruktur dan memungkinkan penggunaan kembali kode yang sudah ada sehingga menghemat waktu dan usaha dalam pengembangan perangkat lunak. Polymorphism adalah konsep dimana suatu objek dapat memiliki banyak bentuk. Polymorphism memungkinkan sebuah objek dari sebuah kelas dapat dianggap sebagai objek dari *superclassnya*, sehingga memungkinkan kita untuk memperlakukan objek yang berbeda-beda sebagai objek yang sama ketika mengakses metodenya. Ada dua bentuk polymorphism, yaitu runtime polymorphism dan compile-time polymorphism. Compile-time polymorphism adalah teknik pemrograman dimana keputusan untuk memanggil sebuah fungsi atau metode dilakukan pada saat kompilasi. Sedangkan, runtime polymorphism adalah teknik pemrograman dimana keputusan untuk memanggil sebuah fungsi atau metode dilakukan pada saat runtime (saat program sedang berjalan).

Berdasarkan kedua pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut ke dalam AbilityCard. Kelas AbilityCard sebagai *subclass* dari kelas Card. Hal tersebut disebabkan adanya penurunan sifat dari *superclass* Card ke *subclass* AbilityCard dan digunakan sebagai kelas yang sama (polymorphism). Berikut adalah contoh implementasi kode kami.

Berbagai macam kelas dalam efek kartu seperti reroll, quadruple, quarter, reverse, swap, switch, dan abilityless yang meng- <i>inherit</i> kelas Ability	
--	--

	 <pre>1 class AbilityLessAct: public Action,public Abilityless{ 2 private: 3 Player* p; 4 SetGame* s; 5 public: 6 AbilityLessAct(Player&,SetGame&); 7 void Act(); 8 }; 8 </pre>
Beberapa macam subclass dalam kartu aksi seperti abilityless, reroll, switch, dan lain-lain memiliki superclass yaitu action dan abilityless.	

2.2. Method/Operator Overloading

Operator overloading adalah konsep dalam pemrograman berorientasi objek (OOP) dimana sebuah operator didefinisikan ulang untuk memiliki arti baru ketika digunakan pada suatu tipe data kustom. Dalam OOP, operator seperti +, -, *, /, dan lain-lain, digunakan untuk melakukan operasi matematika pada tipe data numerik, namun dengan operator overloading, operator-operator tersebut dapat digunakan untuk melakukan operasi pada tipe data kustom. Dalam operator overloading, sebuah operator didefinisikan ulang untuk menerima tipe data kustom dan memungkinkan operasi yang telah didefinisikan pada tipe data kustom tersebut. Misalnya, operator + dapat didefinisikan ulang pada kelas yang merepresentasikan vektor, sehingga operator + dapat digunakan untuk menjumlahkan dua objek vektor.

Berdasarkan pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut ke dalam kelas Player untuk mengecek kondisi kesamaan antara pemain satu dengan pemain yang lainnya. Berikut adalah contoh implementasi kode kami.

Operator untuk menentukan apakah pemain sama atau tidak

```
● ● ●  
1 bool Player::operator==(const Player &p) {  
2     return this->getTurn() == p.getTurn();  
3 }  
4 bool Player::operator!=(const Player &p) {  
5     return this->getTurn() != p.getTurn();  
6 }
```

2.3. Template & Generic Classes

Template dan generic classes adalah konsep dalam pemrograman berorientasi objek yang memungkinkan pembuatan kode yang lebih fleksibel dan dapat digunakan kembali. Template adalah fitur dalam C++ yang memungkinkan pembuatan fungsi dan kelas generik. Fungsi atau kelas template didefinisikan menggunakan tipe atau nilai parameter yang dapat ditentukan saat fungsi atau kelas di-instansiasi.

Berdasarkan pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut kedalam fungsi randomIndex.

Template yang digunakan untuk merandom indeks secara unik dari sebuah vector yang nantinya akan dibagikan ke setiap player

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <cstdlib>
 4 #include <ctime>
 5 using namespace std;
 6
 7 template<class T>
 8 void randomIndex(const vector<T>& vec, vector<int>& idx) {
 9     srand(time(nullptr));
10     vector<int> randomIdx(vec.size());
11     for (int i = 0; i < vec.size(); i++) {
12         randomIdx[i] = i;
13     }
14     for (int i = 0; i < vec.size(); i++) {
15         int randIndex = i + (rand() % (vec.size() - i));
16         idx.push_back(randomIdx[randIndex]);
17         swap(randomIdx[i], randomIdx[randIndex]);
18     }
19 }
```

2.4. Exception

Exception adalah mekanisme dalam pemrograman yang memungkinkan suatu program untuk menangani kondisi abnormal atau kesalahan yang terjadi selama program berjalan. Ketika sebuah kondisi yang tidak terduga atau tidak diinginkan terjadi, seperti kesalahan input atau kegagalan operasi, program dapat melempar (throw) sebuah exception untuk memberitahu pengguna atau bagian lain dari program bahwa sesuatu telah salah. Exception biasanya diimplementasikan sebagai objek khusus yang berisi informasi tentang kesalahan atau kondisi yang tidak terduga, dan kemudian dilemparkan oleh program menggunakan kata kunci throw. Setelah exception dilempar, program akan mencari bagian dari kode yang dapat menangani (handle) exception tersebut menggunakan blok try-catch.

Berdasarkan pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut ke dalam kondisi ketika tidak memiliki kartu ability tertentu dan ketika kartu *ability* tersebut sudah dipakai. Implementasi dilakukan dengan melempar pesan ke dalam *struct* (throw exception) jika kondisi tidak punya kartu *ability* dan kartu *ability* sudah dipakai terjadi. Berikut adalah contoh implementasi kode kami.

Contoh implementasi exception untuk mengembalikan pesan tidak memiliki kartu ability

```

1 struct notHaveReRoll: public exception{
2     const char* const what() throw(){
3         return "Kamu tidak memiliki ReRollCard";
4     }
5 };
6 struct notHaveQuadrupleCard: public exception{
7     const char* const what() throw(){
8         return "Kamu tidak memiliki QuadrupleCard";
9     }
10};
11 struct notHaveQuarterCard: public exception{
12    const char* const what() throw(){
13        return "Kamu tidak memiliki QuarterCard";
14    }
15};
16 struct notHaveReverseCard: public exception{
17    const char* const what() throw(){
18        return "Kamu tidak memiliki ReverseCard";
19    }
20};
21 struct notHaveSwapCard: public exception{
22    const char* const what() throw(){
23        return "Kamu tidak memiliki SwapCard";
24    }
25};
26 struct notHaveSwitchCard: public exception{
27    const char* const what() throw(){
28        return "Kamu tidak memiliki SwitchCard";
29    }
30};
31 struct notHaveAbilityCard: public exception{
32    const char* const what() throw(){
33        return "Kamu tidak memiliki AbilityCard";
34    }
35};

```

Contoh penggunaan throw dalam reRoll	<pre> ● ● ● 1 void PlayerAction::reRoll(Player &p, SetGame &s){ 2 int idxPlayer=0; 3 int idxCard=0; 4 5 for(int i=0;i<s.getPlayers().size();i++){ 6 if(*s.getPlayers()[i]==p) idxPlayer=i; 7 } 8 9 10 for(int i=0;i<s.getPlayers()[idxPlayer]->getCardsPlayer().size();i++){ 11 bool abilityP1 = 12 s.getPlayers()[idxPlayer]->getCardsPlayer()[i]->getNameCard()!="M" 13 s.getPlayers()[idxPlayer]->getCardsPlayer()[i]->getNameCard()!="B" 14 s.getPlayers()[idxPlayer]->getCardsPlayer()[i]->getNameCard()!="K" 15 s.getPlayers()[idxPlayer]->getCardsPlayer()[i]->getNameCard()!="H"; 16 17 if(abilityP1){ 18 idxCard=i; 19 } 20 } 21 22 if(!(*s.getPlayers()[idxPlayer]->getCardsPlayer()[idxCard]->getNameCard())=="ReRoll") throw 23 notHaveReRoll(); 24 if(s.getPlayers()[idxPlayer]->getCardsPlayer()[idxCard]->getIsCardDeath()) throw 25 reRollIsDeath(); 26 27 }</pre>

2.5. C++ Standard Template Library

C++ Standard Template Library (STL) adalah kumpulan dari kelas dan fungsi template yang disediakan sebagai bagian dari C++ Standard Library. STL menyediakan algoritma, struktur data, dan fungsi bawaan lainnya yang dapat digunakan oleh programmer C++ untuk mempercepat dan mempermudah pembuatan program yang efisien. STL terdiri dari tiga komponen utama: container, iterator, dan algorithm. Container adalah kelas yang menyimpan dan mengelola data dalam berbagai struktur, seperti vector, deque, list, dan map.

Iterator adalah kelas yang memungkinkan akses ke elemen dalam container, mirip dengan pointer dalam C. Algorithm adalah kumpulan fungsi template yang diterapkan pada container untuk melakukan operasi seperti sorting, searching, dan manipulating.

Berdasarkan pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut ke dalam program kami yaitu *library vector* untuk set permainan dengan vektor yang ada memiliki tipe data vektor kartu dan vektor pemain. Selain itu, kelompok kami juga mengimplementasikan *library map* untuk proses penentuan skor pada setiap kemungkinan kombinasi mulai dari *High Card* hingga *Straight Flush*. Berikut adalah contoh implementasi kode kami.

Penggunaan STL vector untuk kelas SetGame

```
● ● ●  
1 #include <iostream>  
2 #include <vector>  
3 #include <algorithm>  
4  
5  
6 class SetGame {  
7 private:  
8     vector<Card*> cards;  
9     vector<Player*> players;  
10    vector<Card*> ability;  
11    static int turn;  
12    Long int pointGame;  
13
```

Penggunaan STL map untuk proses menentukan skor pada setiap kemungkinan kombinasi (dalam contoh ini kombinasi untuk High Card dan One Pair)

```
● ● ●  
1 #include "Combination.hpp"  
2 #include <algorithm>  
3 #include <map>
```

```
1 HighCard :: HighCard(vector<Card*> &v) : Cards(cards) {
2     map <string,int> mp;
3     mp["H"] = 0;
4     mp["B"] = 1;
5     mp["K"] = 2;
6     mp["M"] = 3;
7
8     this -> value = v[0]->getNumberCard() * 0.1 + mp[v[0]->getNameCard()] * 0.03;
9 }
10
11 double HighCard :: valueCards() {
12     return this -> value;
13 }
14
15 OnePair :: OnePair(vector<Card*> &v) : Cards(cards) {
16     map <string,int> mp;
17     mp["H"] = 0;
18     mp["B"] = 1;
19     mp["K"] = 2;
20     mp["M"] = 3;
21
22     double maxHigh = 1.39;
23     double ret = maxHigh;
24     ret += v[0]->getNumberCard() * 0.1;
25     if(mp[v[0]->getNameCard()] > mp[v[1]->getNameCard()]){
26         ret += mp[v[0]->getNameCard()] * 0.03 + mp[v[1]->getNameCard()] * 0.003;
27     }
28     else {
29         ret += mp[v[1]->getNameCard()] * 0.03 + mp[v[0]->getNameCard()] * 0.003;
30     }
31     this -> value = ret;
32 }
```

2.6. Virtual Function

Virtual function adalah fungsi yang dapat di-override oleh subclass atau turunan dari kelas yang mengandung fungsi tersebut. Dengan menggunakan virtual function, program dapat mengimplementasikan polymorphism, yaitu kemampuan sebuah objek untuk memiliki banyak bentuk atau perilaku. Jika sebuah fungsi dideklarasikan sebagai virtual dalam kelas dasar, kelas turunan dapat meng-override fungsi tersebut dengan implementasi yang berbeda. Ketika fungsi dipanggil pada objek yang ditunjuk oleh pointer atau referensi dari kelas dasar, maka fungsi yang dijalankan adalah versi yang di-override pada kelas turunan.

Berdasarkan pengertian tersebut, kelompok kami mengimplementasikan konsep tersebut ke dalam kelas Card pada fungsi getNameCard dengan tujuan subclass Card dapat meng-override superclass Card.

Contoh implementasi dari virtual function, yaitu getNameCard dari superclass ability.

```
1  /*  
2   mengambil nama dari kartu  
3   entah nama warna maupun nama dari ability  
4   */  
5   virtual string getNameCard() const;
```

```
1  class Action {  
2  public:  
3    Action();  
4    virtual void Act();  
5  };
```

```
● ● ●  
1  class Ability{  
2  public:  
3      // Konstruktor  
4      Ability(string,bool,bool);  
5      // Destruktor  
6      ~Ability();  
7      // Setter and Getter  
8      virtual void setIsCardUsed(bool);  
9      virtual void setIsCardDeath(bool);  
10     virtual bool getIsCardUsed()const;  
11     virtual bool getIsCardDeath()const;  
12     virtual string getNameCard() const;  
13     virtual void effect(Player &p,SetGame &s);  
14  
15 protected:  
16     string name;  
17     bool isDeath;  
18     bool isUsed;  
19 };
```

Contoh implementasi dari abstract class dan pure virtual function, yaitu double valueCards() yang nanti digunakan oleh kelas anakannya yaitu Cards dan berbagai kelas kombinasi. Kelas-kelas ini akan meng-override kelas abstrak Calculable.

```
● ○ ●  
1 // Pure Virtual Class  
2 class Calculable {  
3     public:  
4         virtual double valueCards() = 0;  
5 };  
6
```

```
7 class Cards : public Calculable {  
8     public:  
9         Cards (vector<Card*>);  
10        double valueCards() override;  
11        // int getSuit();  
12        // int getNumber();  
13    protected:  
14        double value;  
15        // int number;  
16        // int suit;  
17        vector<Card*> cards;  
18    };  
19  
20 class HighCard : public Cards {  
21     public :  
22         HighCard(vector<Card*> &v);  
23         double valueCards() override;  
24    };
```

3. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Ability	13521014, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Card dan turunannya	13521001, 13521008, 13521014, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Calculable dan turunannya	13521001, 13521008, 13521013, 13521014, 13521016	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Combination (StraightFlush, FullHouse, TwoPair, dll)	13521001, 13521008, 13521013, 13521014, 13521016	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Exception dan turunannya	13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Player	13521013, 13521016, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
SetGame	13521008, 13521014, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Splash	13521008	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Table	13521001, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031
Main Program	13521001, 13521008, 13521013, 13521014, 13521016, 13521031	13521001, 13521008, 13521013, 13521014, 13521016, 13521031



Kode Kelompok : SST

Nama Kelompok : firSST

1. 13521001 / Angger Ilham A

2. 13521008 / Jason Rivalino

3. 13521013 / Eunice Sarah Siregar

4. 13521014 / Muhamad Syauqi Jannatan

5. 13521016 / Laila Bilbina Khoiru Nisa

6. 13521031 / Fahrian Afdholi

Asisten Pembimbing : Muhammad Tito Prakasa

1. Konten Diskusi

Q: bingung bagian kombinasi, setiap player kan ada 2 kombinasi terus nanti di save player baru diambil kombinasi paling besar. Nah, masih ada yg keliru ga?

A: Disini kan kombinasinya masih return vector. Saran, returnnya langsung kombinasinya aja. Harusnya ada class turunan combination, kayak twopair, straight flush yang bakal nyimpen list of card yang baka di return. Misal, si player punya properti combination nanti tipe datanya sudah bernilai straight flush. Idenya si card sama combi bakal nurunin abstract class calculable yg punya value sehingga value card sama combo bisa diambil valuenya. Card sama combo masih parent class, bisa nge implement virtual function di childnya card sama combo. Class combo sama card punya list of card. Waktu nyari pemenang, tinggal manggil highest combo. Combination yg udh dibuat lebih masuk ke perhitungan combo. Bakal butuh class-class baru, soalnya perhitungannya beda-beda. Ketika menghitung list of card di player combo bakal beda.

Q: Ability yang sudah kami buat sudah benar atau belum? (menunjukan source code)

A: kurang tepat, soalnya ability sebuah value gapunya nomor, parentnya dibikin lebih general. Gaperlu isUsed isDead taronya di deck card. Atau, dipisah aja bukan turunan dari card jadi suatu kelas tersendiri.

2. Tindak Lanjut

- Mengganti program yang salah
- Menambah kelas baru
- Mengikuti saran yang diberikan dari Kakak Asisten pada saat asistensi kemarin

Kode Kelompok : SST

Nama Kelompok : firSST

1. 13521001 / Angger Ilham A

2. 13521008 / Jason Rivalino

3. 13521013 / Eunice Sarah Siregar

4. 13521014 / Muhamad Syauqi Jannatan

5. 13521016 / Laila Bilbina Khoiru Nisa

6. 13521031 / Fahrian Afholi

Asisten Pembimbing : Muhammad Tito Prakasa

1. Konten Diskusi

Konten diskusi berisi setiap pertanyaan dan jawaban yang dibahas saat asistensi. Dapat ditulis dalam format poin-per-poin atau paragraf.

-Q: Untuk setiap ronde, sistem pertambahan kartunya itu bagaimana ya?

A: Sistemnya untuk setiap ronde, akan ada pertambahan satu kartu, ronde pertama dibagikan dua kartu, ronde kedua dibagikan kartu ability, dari ronde 1-5, setiap akhir dari ronde, akan dibagikan satu kartu ke atas meja (*table card*). Pemain bisa membuat susunan kartu dari kartu ditangan dikombinasikan dengan kartu yang ada di meja, untuk ronde 6, tidak ada pembagian kartu di meja.

-Q: apakah poin 64 itu karena ada 6 ronde sehingga 2^6 ?

A: bukan, 64 itu poin awal dari setiap ronde, pertambahannya berdasarkan aksi yang pemain lakukan

-A: pemain bisa memilih next, kalo ga mau ngedoublein ato nge half

-Q: 1 kali permainan kan 6 ronde, kalo misal lanjut permainan poin pemainnya bakal kesimpul atau bagaimana

A: Poin hadiah di meja bakal 64 lagi, tapi poin pemainnya kesimpul

-Q: apakah permainan bisa selesai kalau sudah lewat 1 permainan tapi belum ada yang mencapai 2^{32} ?

A: tidak bisa, harus ada yang poinnya mencapai 2^{32} (tapi kalo mau bikin ekstra ditengah-tengah game bisa berhenti diperbolehkan)

-Q: sistem ngedapetin poinnya itu kalau misal sudah selesai 1 permainan brati yang dapet poin cuman 1 pemenang ya?

A: iya cuma 1 pemenang

-Q: poinnya itu bakal bisa berkurang atau cuma bertambah aja?

A: poinnya cuma bisa bertambah aja, tambahnya tergantung aksi dari pemain, mau double, half, quadruple, atau quarter (kalo misal poin di meja ada 2^1 , terus ada pemain yang pengen nge quarter, itu gabakal jadi 2^{-2} , tapi bakal mentok jdi 2^0 aja atau 1)

Tindak Lanjut

- Melanjutkan pembuatan program yang masih belum selesai
- Memperbaiki program yang masih terdapat kesalahan
- Mengikuti saran yang diberikan dari Kakak Asisten pada saat asistensi kemarin