

Kode Kelompok : OOP

Nama Kelompok : MarhabanTiba

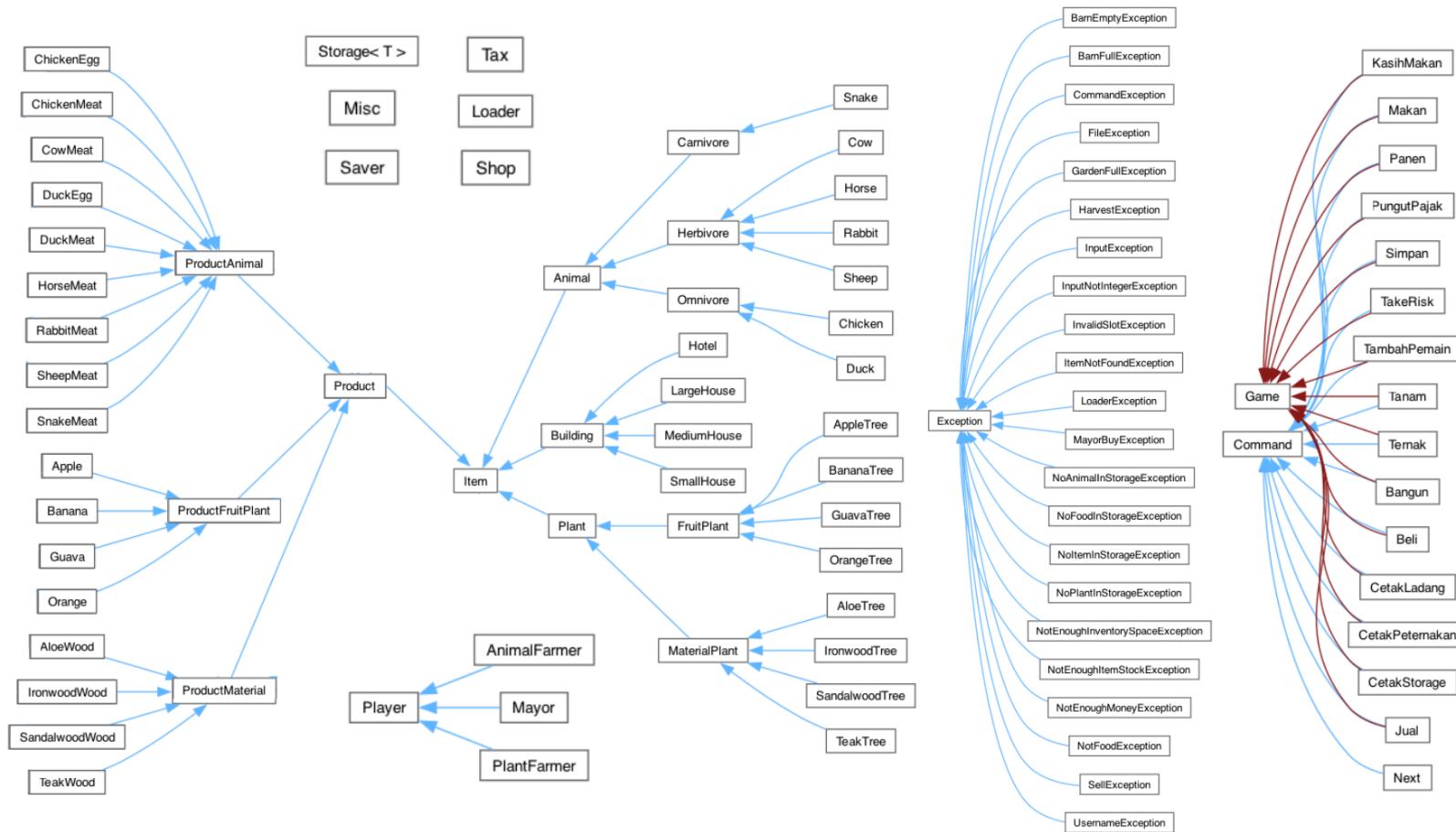
1. 13522025 / Debrina Veisha Rashika W
2. 13522035 / Melati Anggraini
3. 13522060 / Andhita Naura Hariyanto
4. 13522069 / Nabila Shikoofa Muida
5. 13522087 / Shulha

Asisten Pembimbing : Marcellus Michael Herman Kahari

Github Repository : <https://github.com/shulhajws/MarhabanTiba>

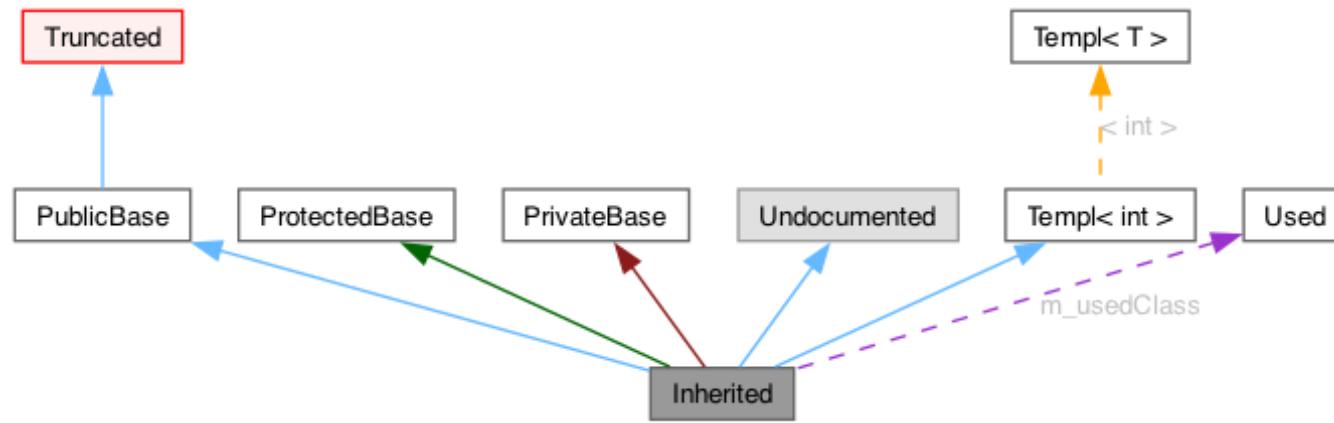
1. Diagram Kelas

1.1. *Class Hierarchy*



1.2. Class Diagram

Graph Legend



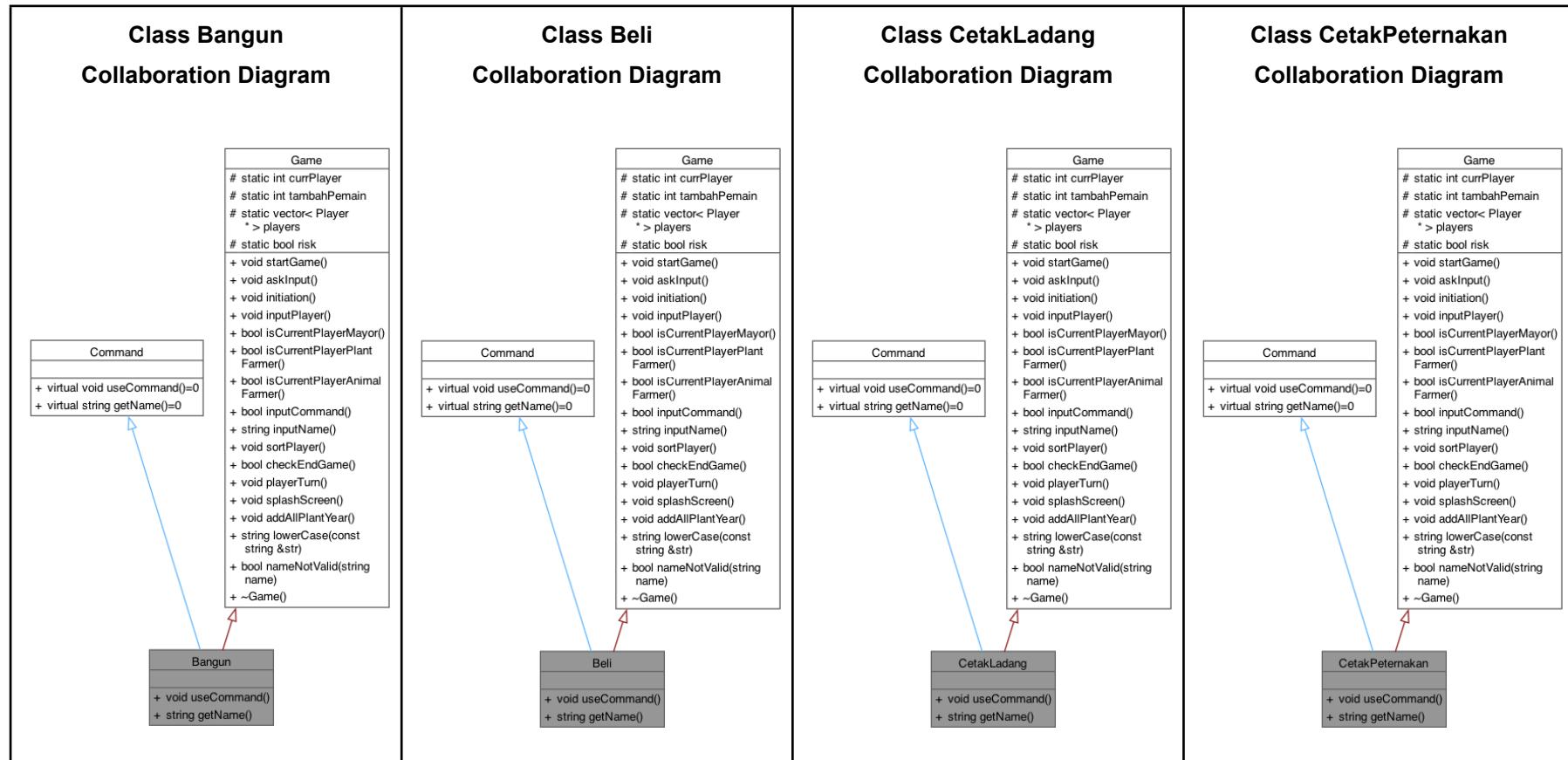
Kotak dalam grafik di atas memiliki arti sebagai berikut:

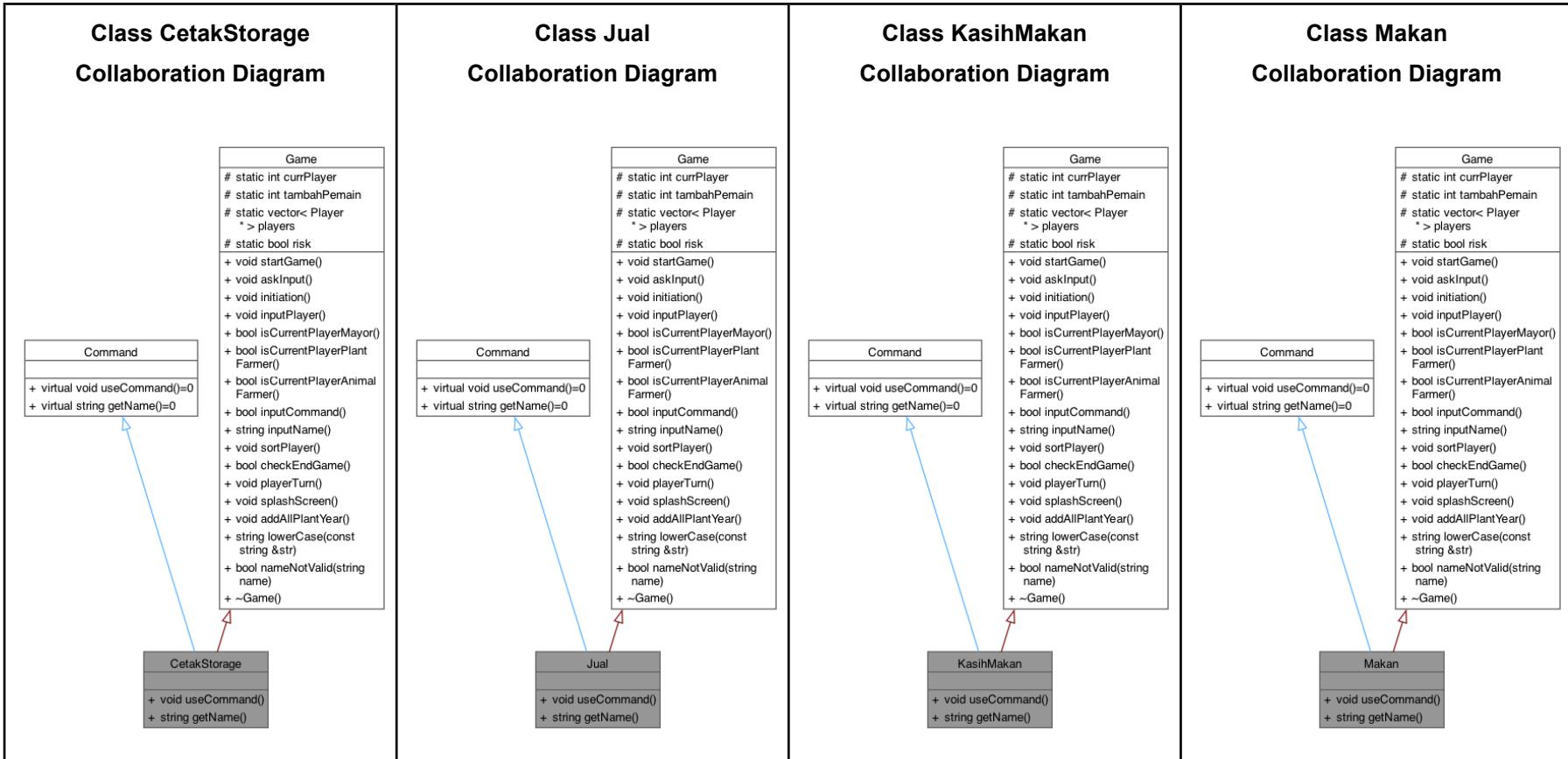
- Kotak abu-abu menggambarkan struct atau class untuk yang grafik tersebut dibuat.
- Kotak dengan batas hitam menunjukkan struct atau class yang didokumentasikan.
- Kotak dengan batas abu-abu menunjukkan struct atau class yang tidak didokumentasikan.
- Kotak dengan batas merah menunjukkan struct atau class yang didokumentasikan tetapi tidak semua relasi warisan/konten ditunjukkan.

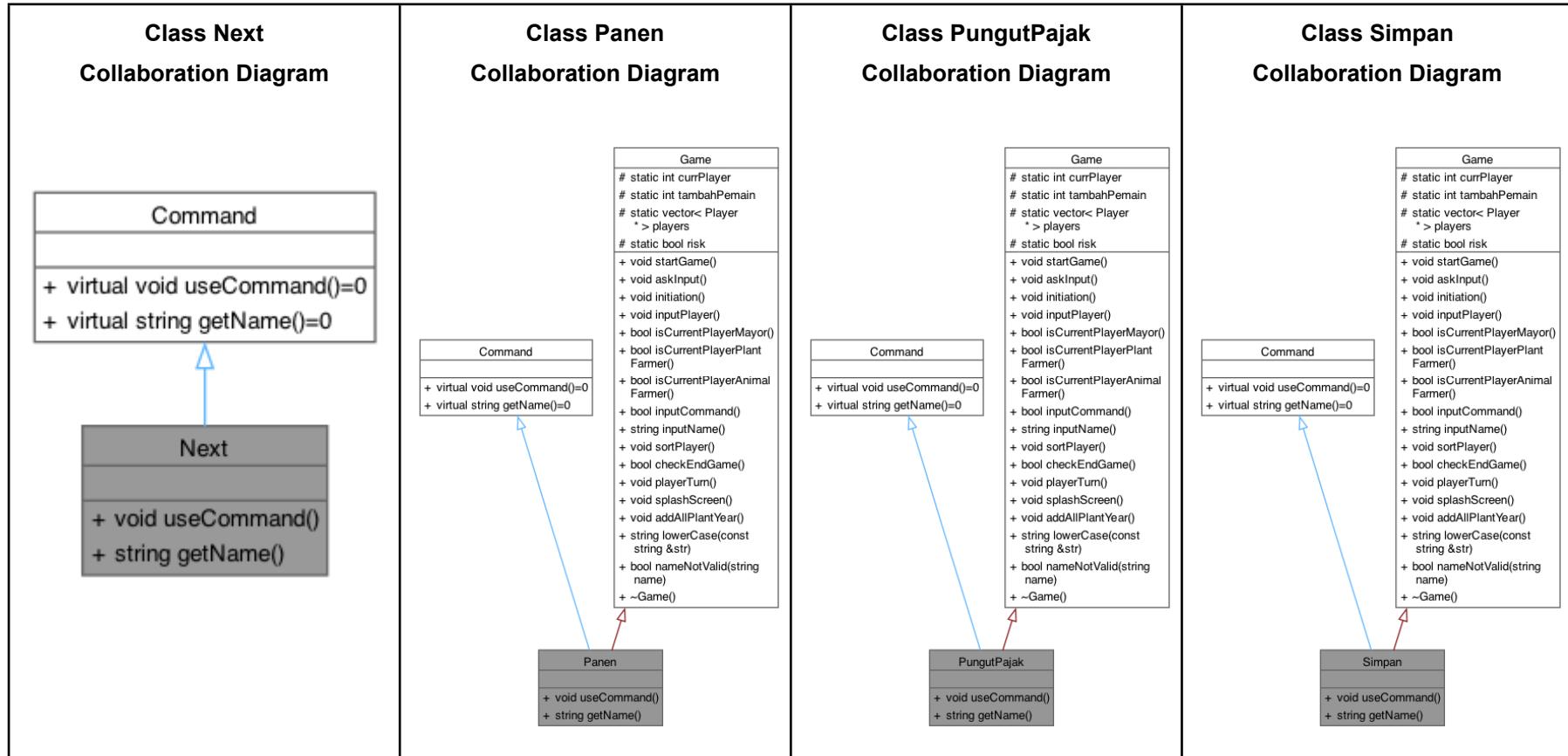
Panah-panah memiliki arti sebagai berikut:

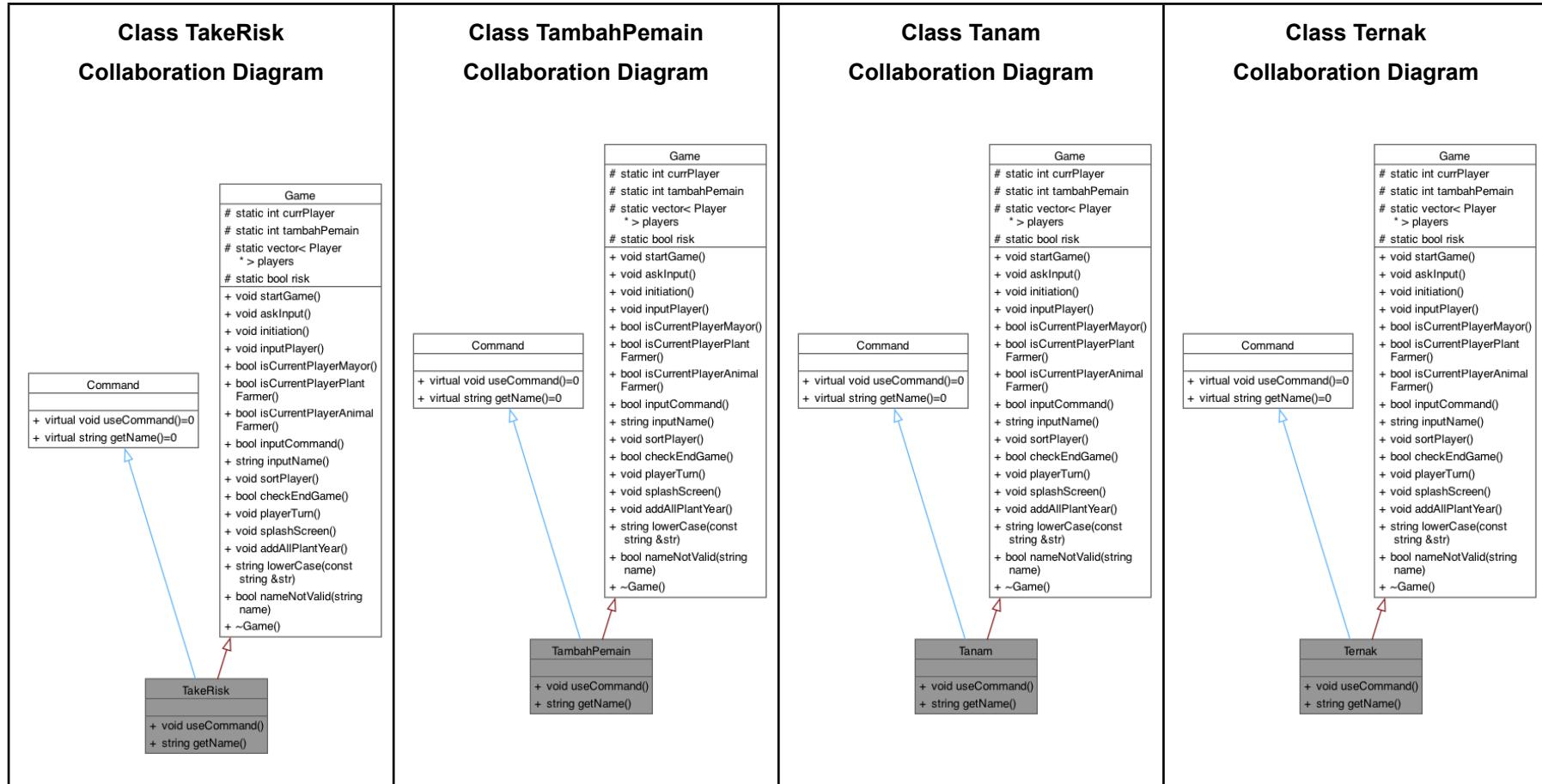
- Panah biru digunakan untuk memvisualisasikan relasi pewarisan publik antara dua kelas.
- Panah hijau gelap digunakan untuk protected inheritance.

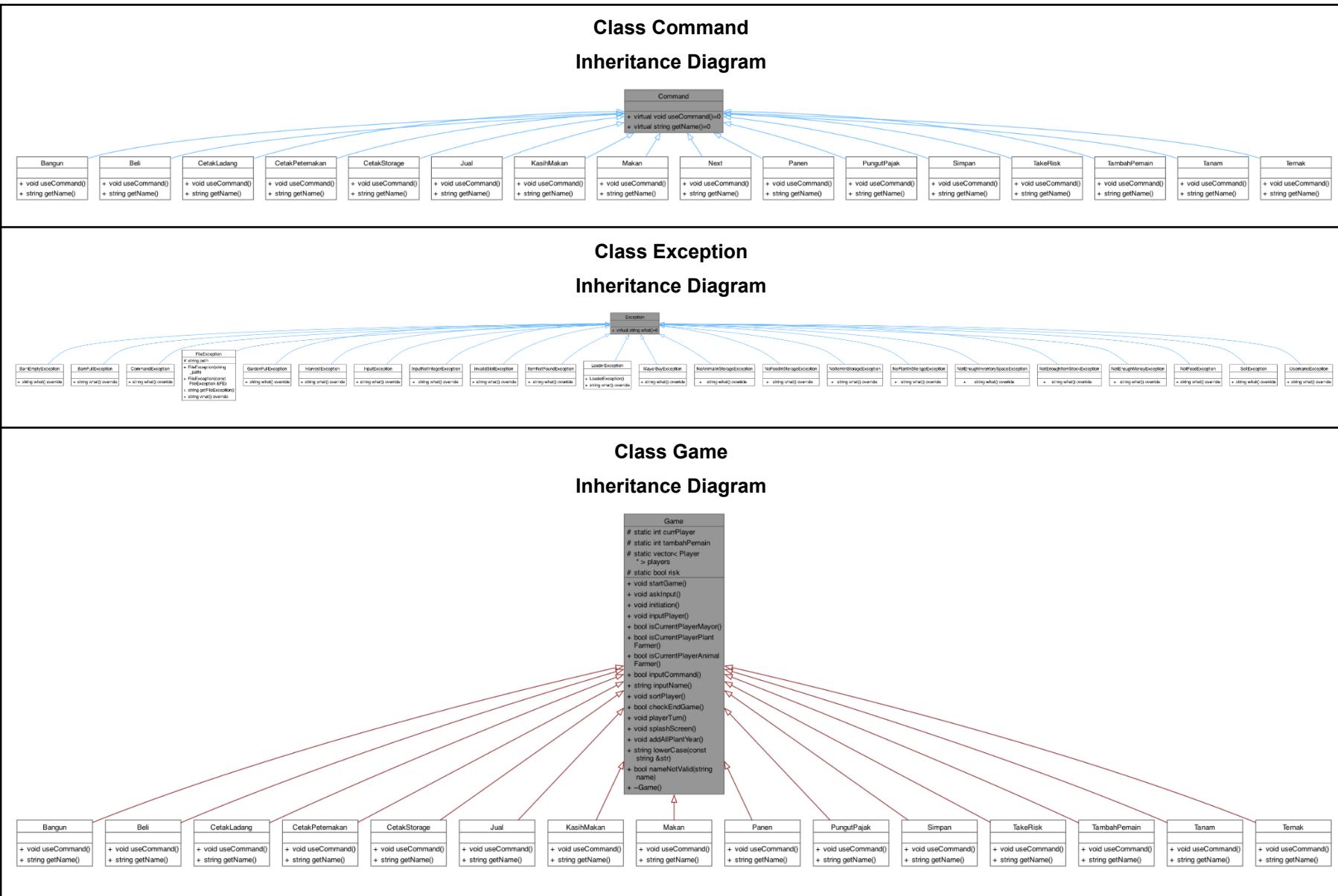
- Panah merah gelap digunakan untuk private inheritance.
- Panah putus-putus ungu digunakan jika sebuah kelas terkandung atau digunakan oleh kelas lain
- Panah putus-putus kuning menunjukkan relasi antara sebuah instansi template dan kelas template.

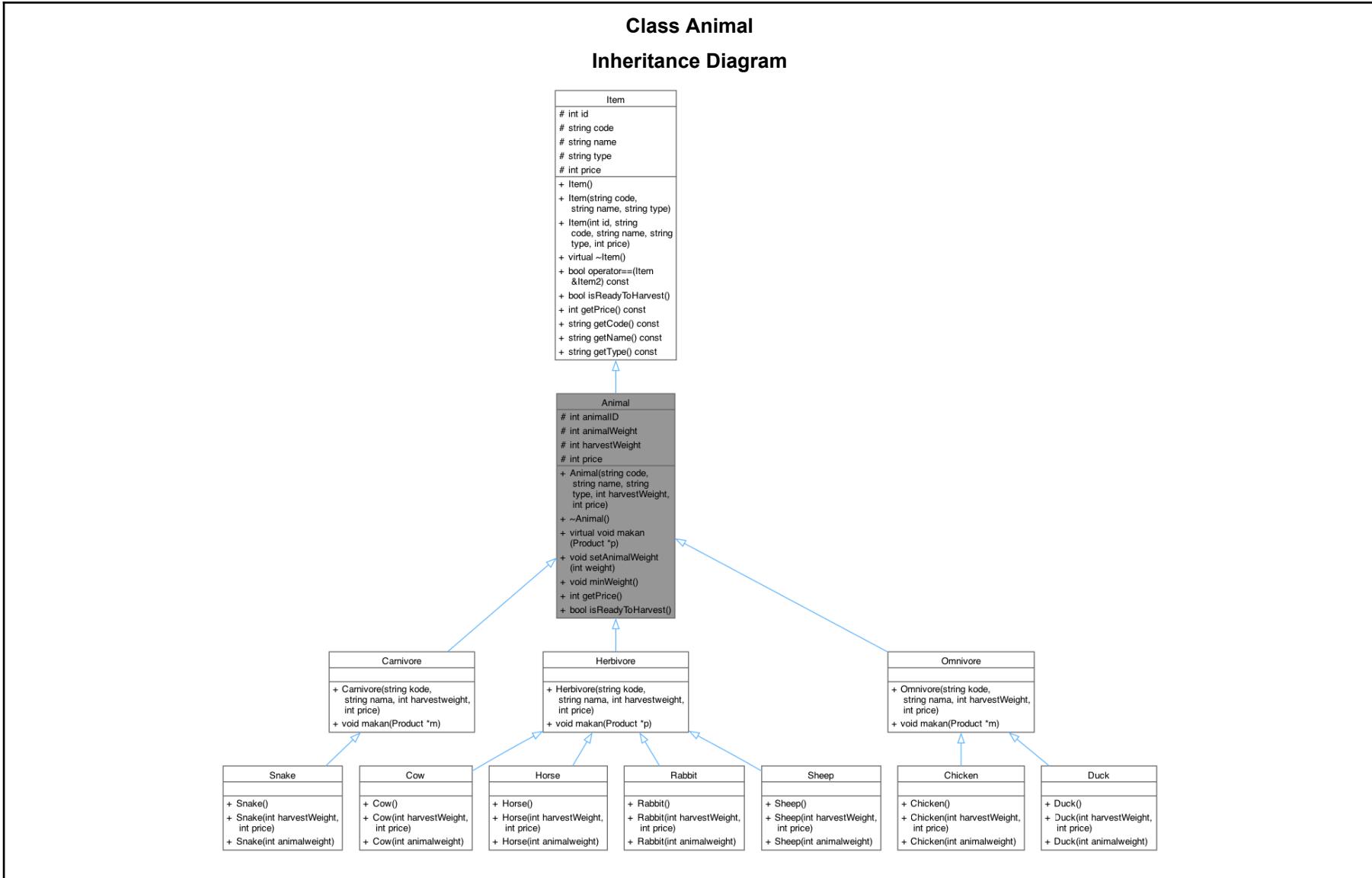


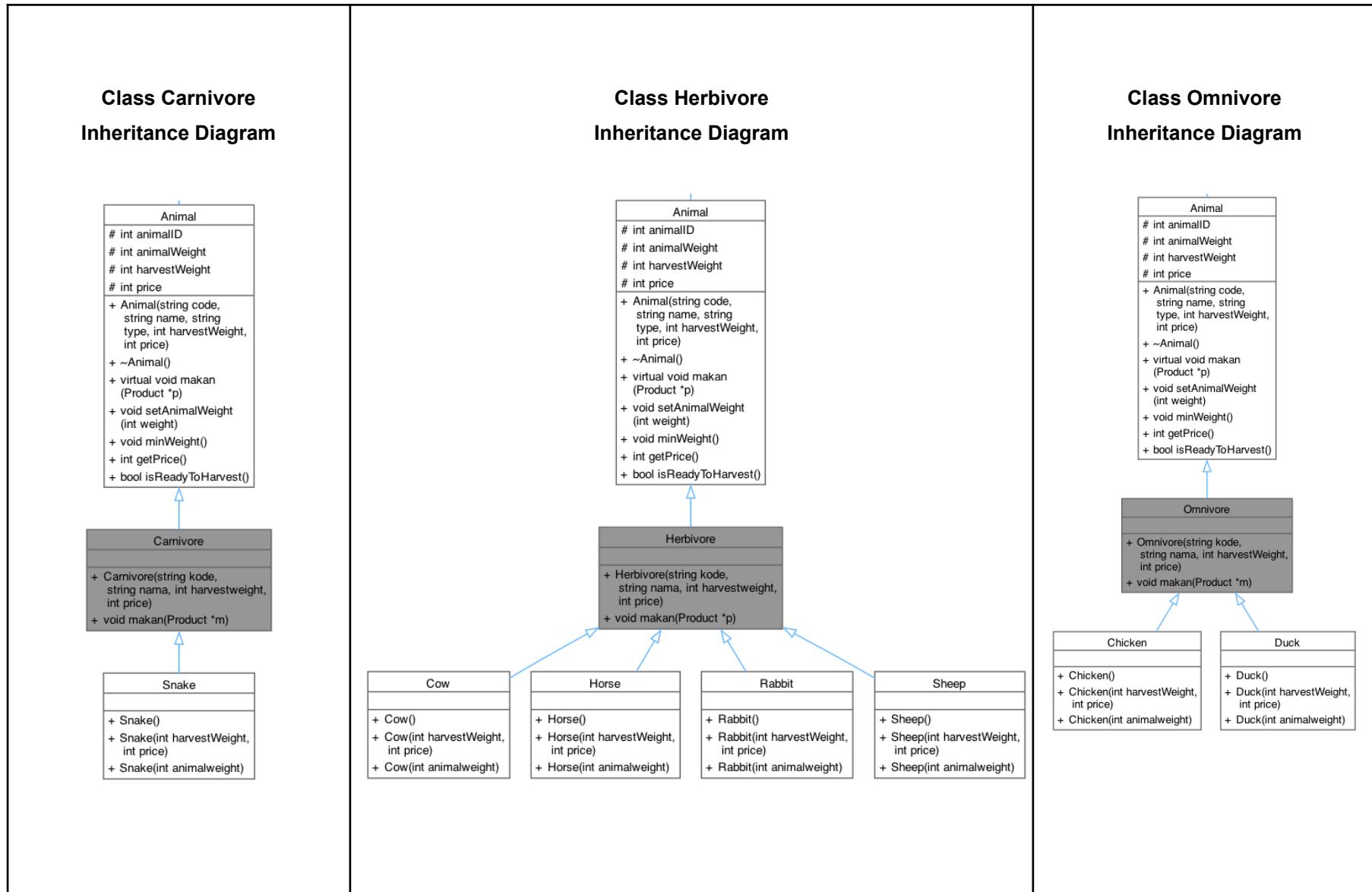


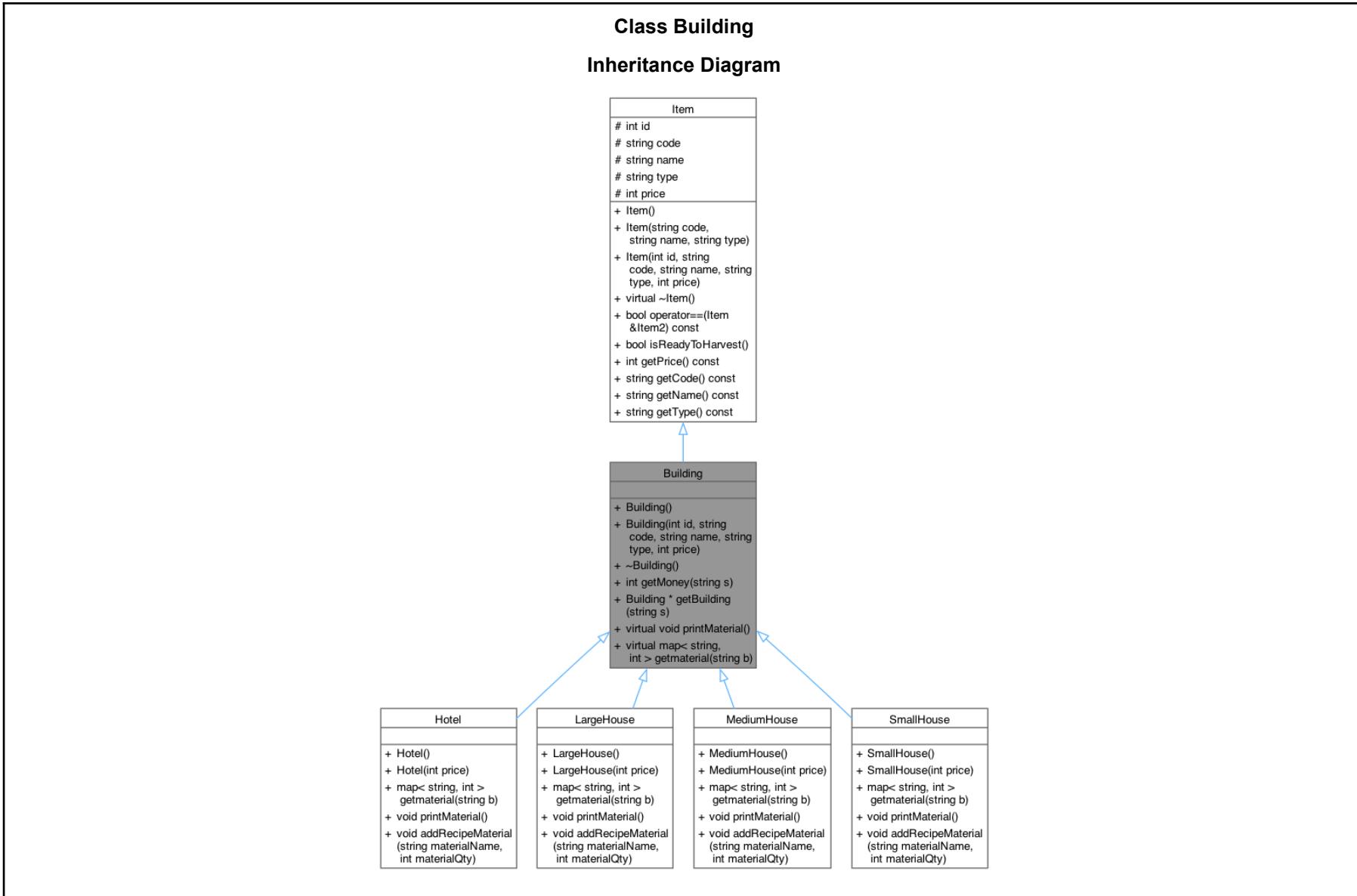


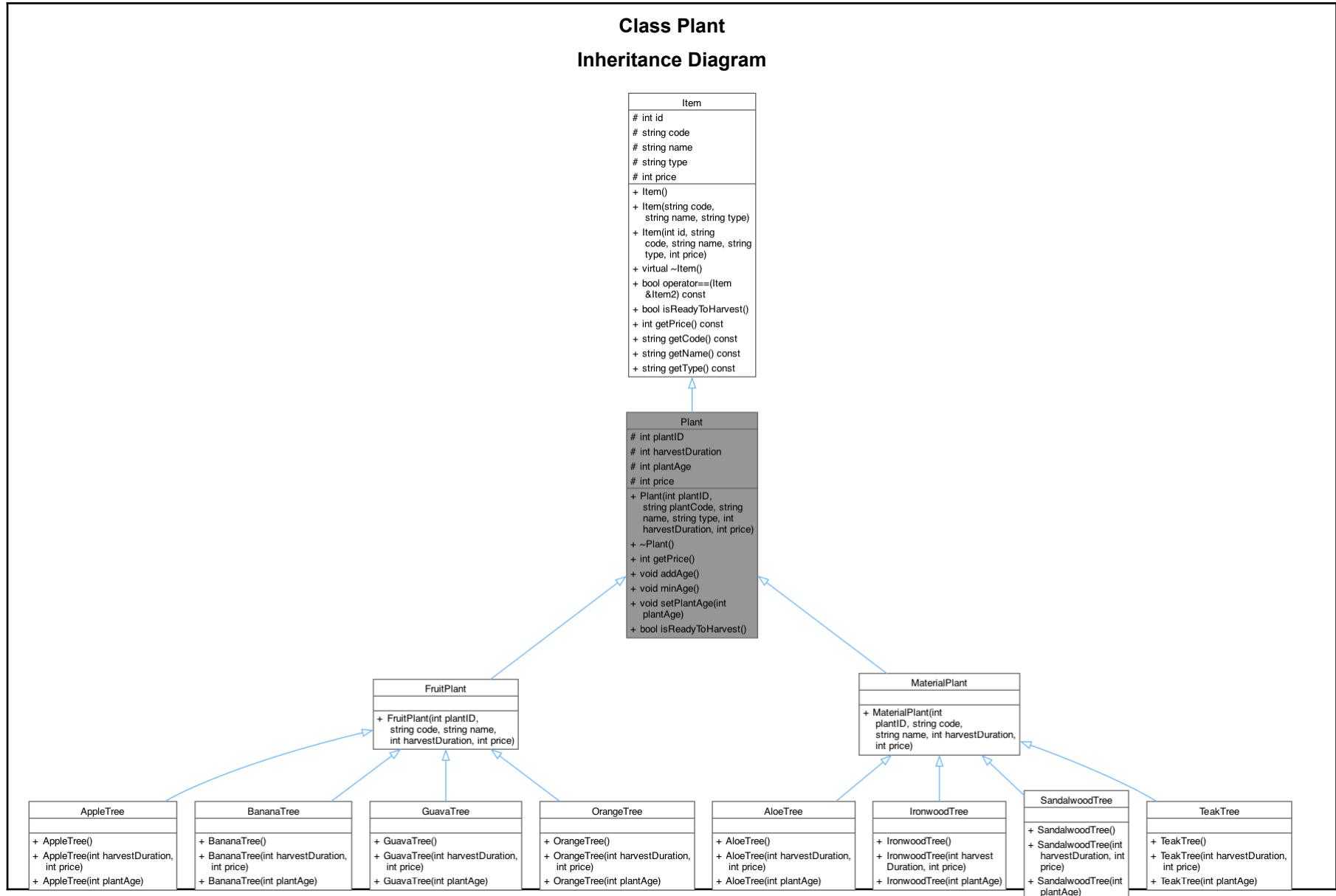


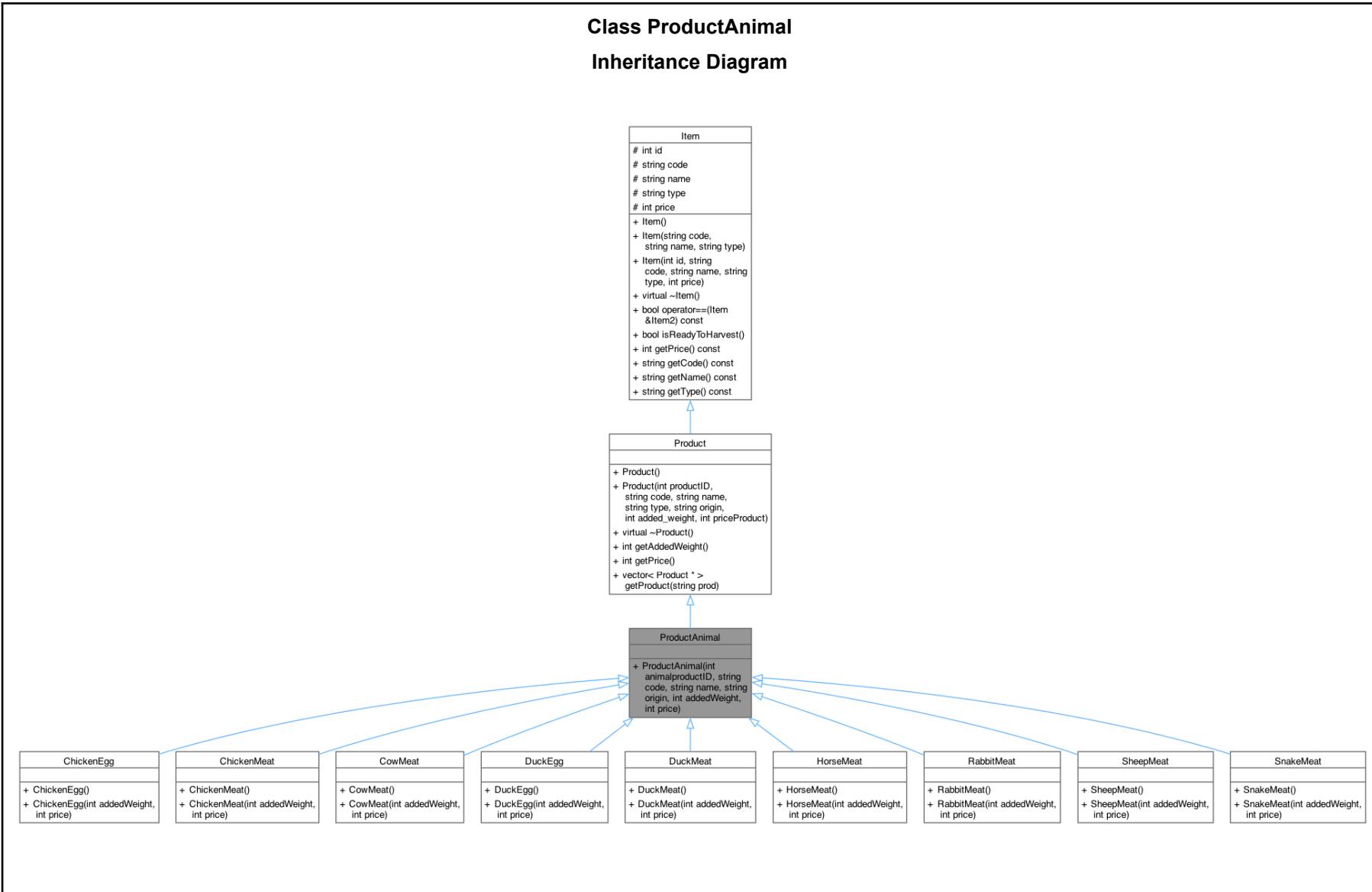


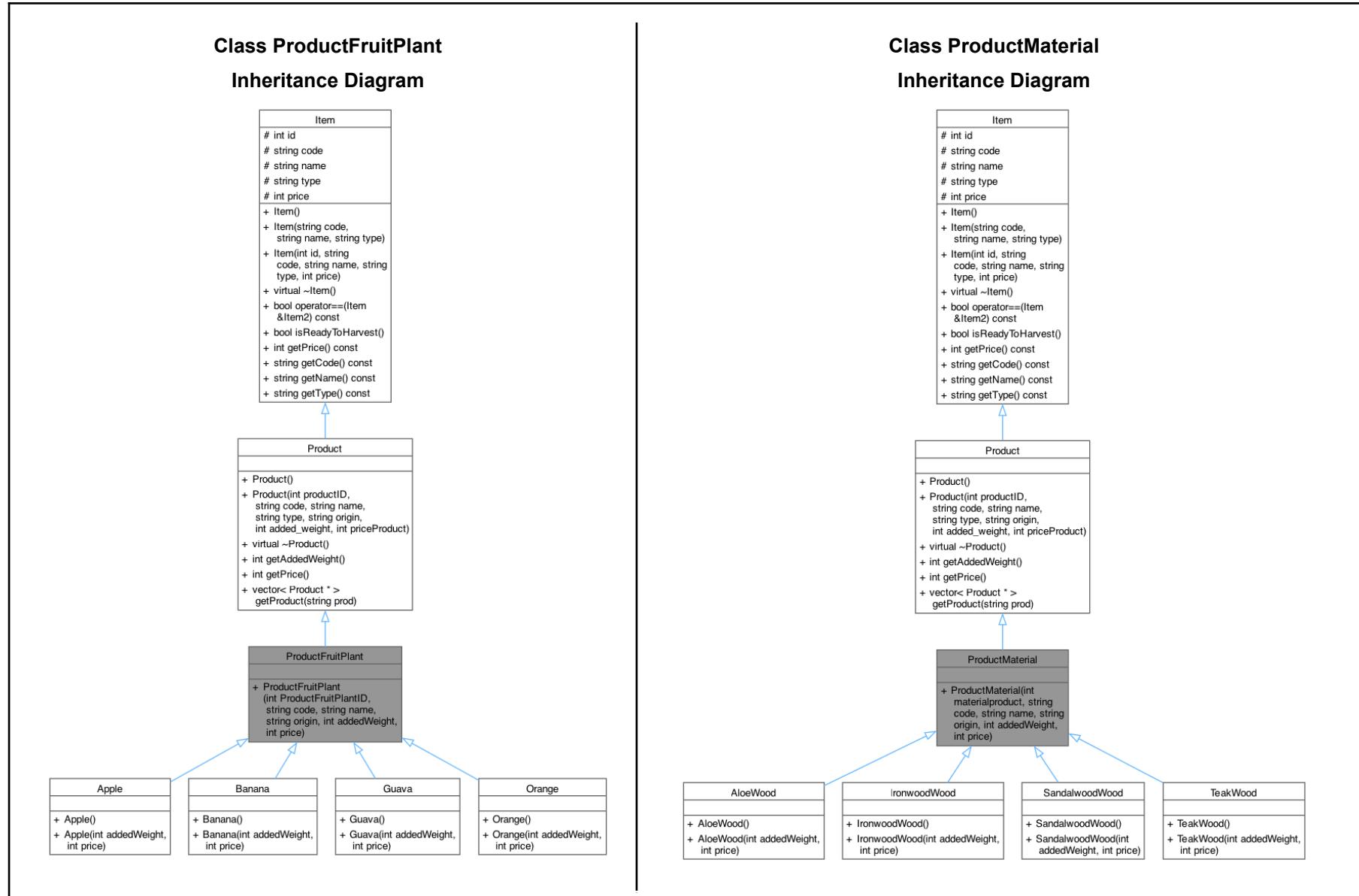


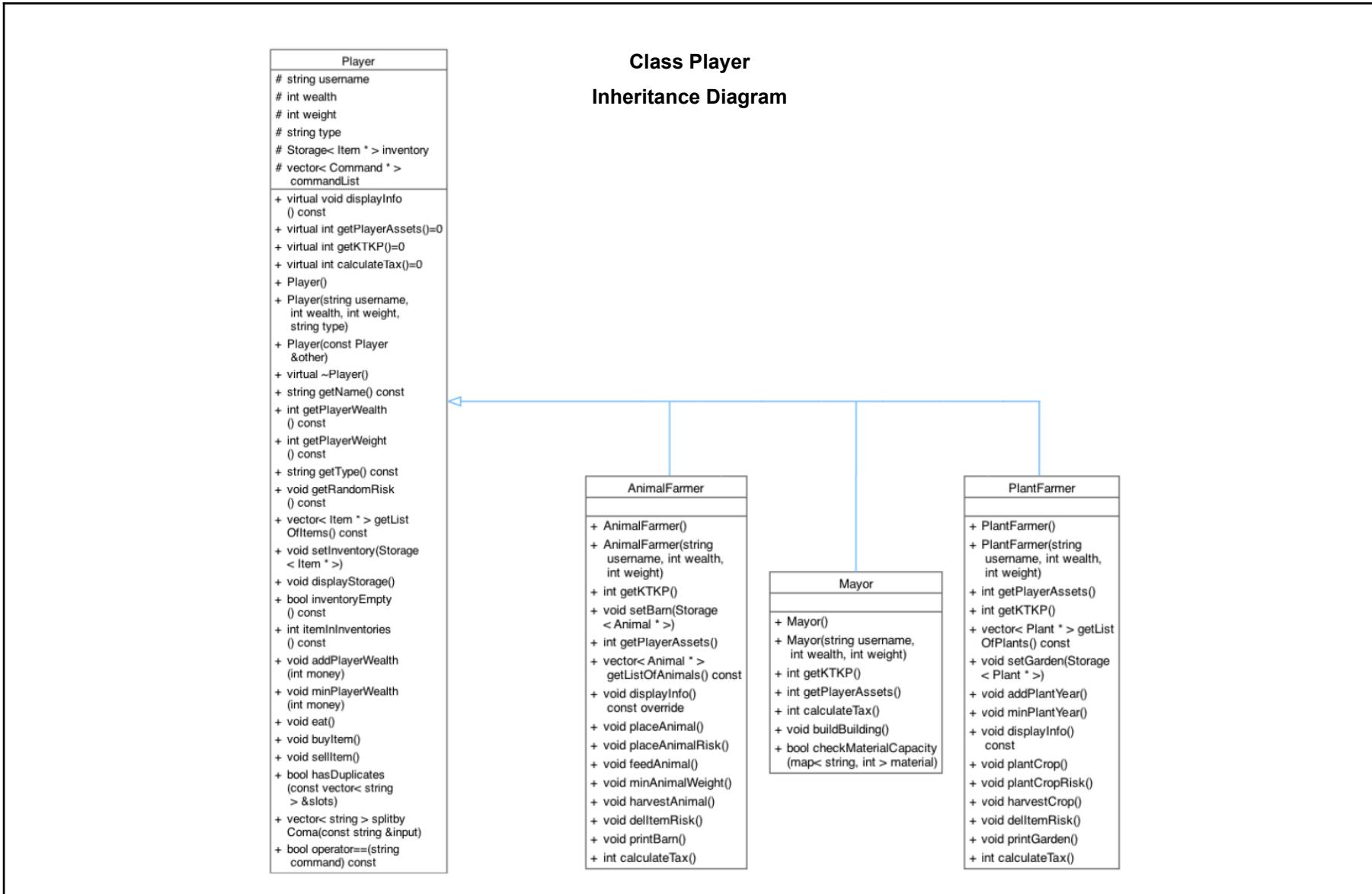






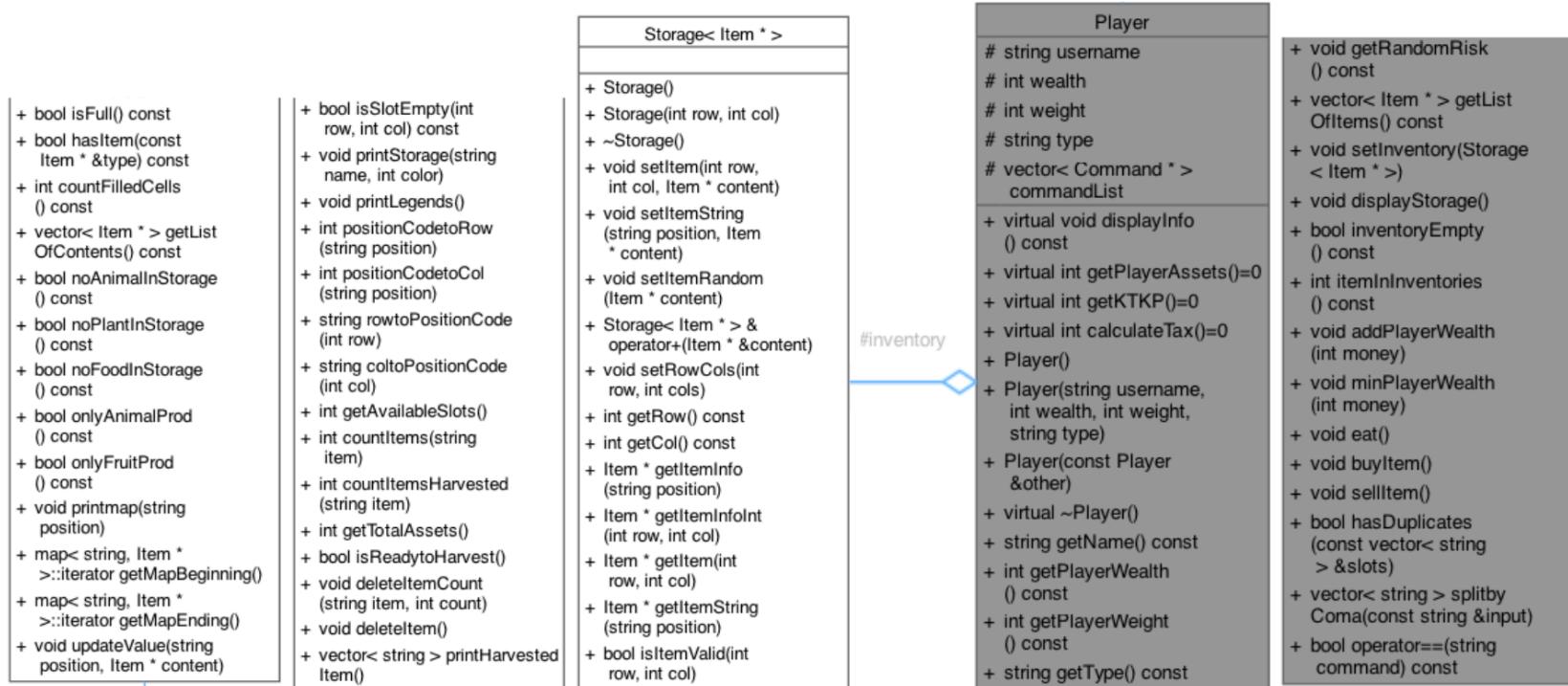


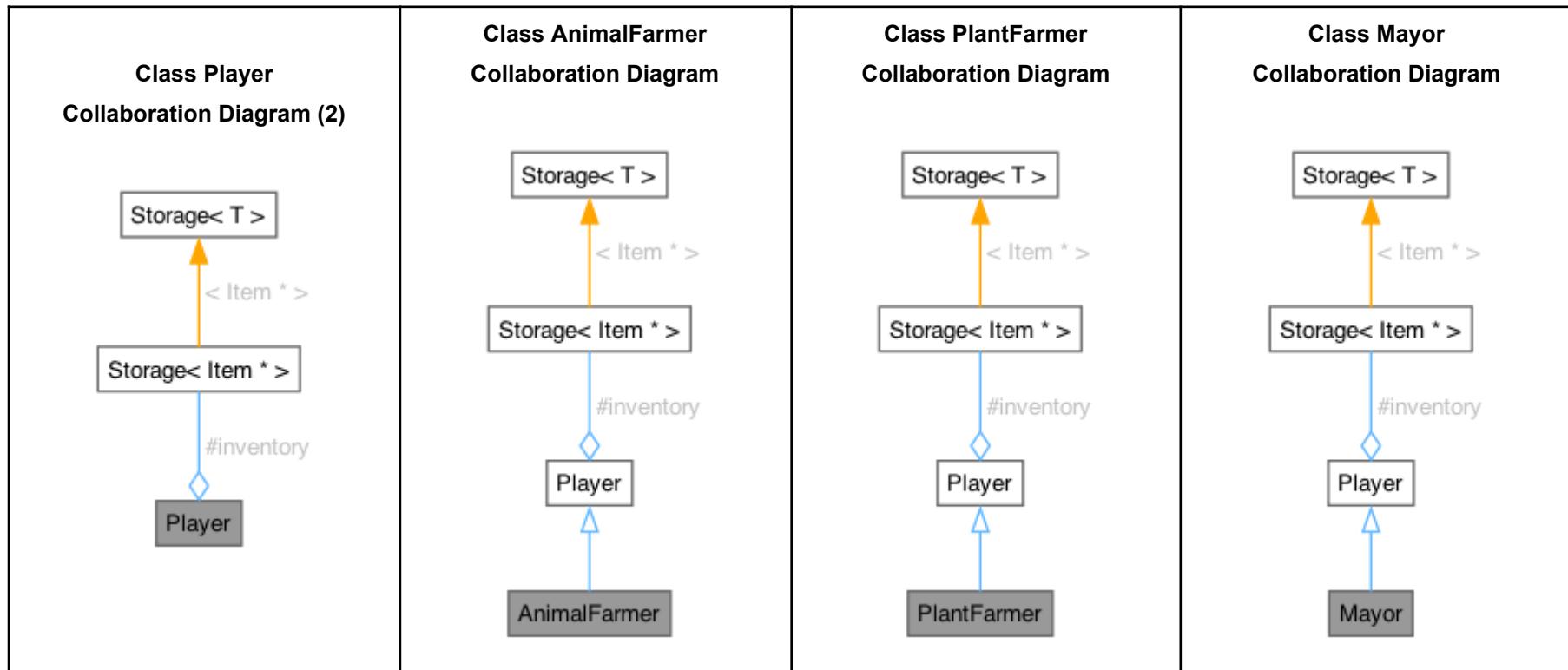




Class Player

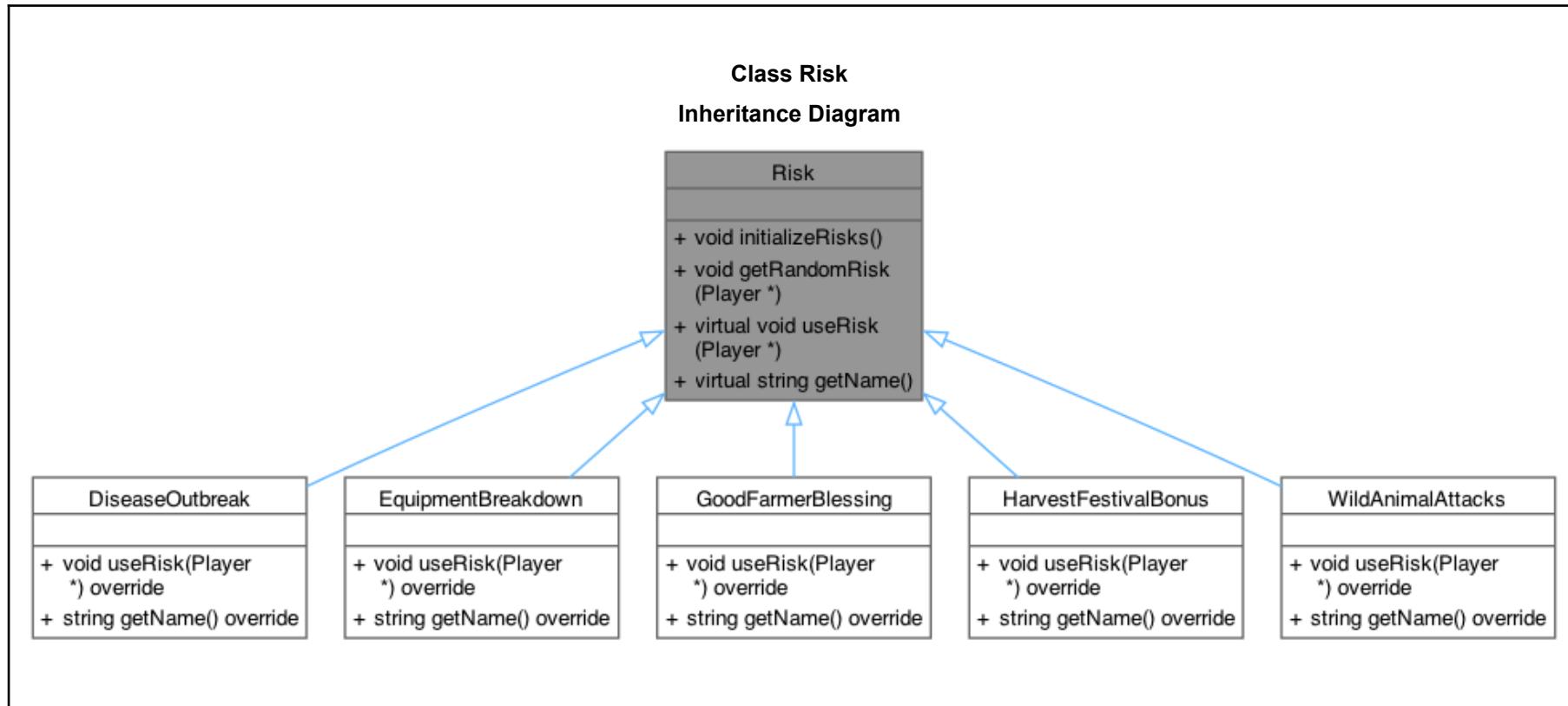
Collaboration Diagram





Class Storage
Collaboration Diagram

Storage< T >	
<ul style="list-style-type: none"> + Storage() + Storage(int row, int col) + ~Storage() + void setItem(int row, int col, T content) + void setItemString(string position, T content) + void setItemRandom(T content) + Storage< T > & operator +(T &content) + void setRowCols(int row, int cols) + int getRow() const + int getCol() const + T getItemInfo(string position) + T getItemInfoInt(int row, int col) + T getItem(int row, int col) + T getItemString(string position) 	<ul style="list-style-type: none"> + bool isItemValid(int row, int col) + bool isSlotEmpty(int row, int col) const + void printStorage(string name, int color) + void printLegends() + int positionCodetoRow(string position) + int positionCodetoCol(string position) + string rowtoPositionCode(int row) + string coltoPositionCode(int col) + int getAvailableSlots() + int countItems(string item) + int countItemsHarvested(string item) + int getTotalAssets() + bool isReadytoHarvest() + void deleteItemCount(string item, int count) + void deleteItem() + vector< string > printHarvestedItem() + bool isEmpty() const + bool isFull() const + bool hasItem(const T &type) const + int countFilledCells() const + vector< T > getListOfContents() const + bool noAnimalInStorage() const + bool noPlantInStorage() const + bool noFoodInStorage() const + bool onlyAnimalProd() const + bool onlyFruitProd() const + void printmap(string position) + map< string, T >::iterator getMapBeginning() + map< string, T >::iterator getMapEnding() + void updateValue(string position, T content)



<p>Class Tax</p> <p>Collaboration Diagram</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; background-color: #f0f0f0;">Tax</td> </tr> <tr> <td style="padding: 5px;"> # int ktkp # int uang # int harta + Tax(int ktkp, int uang, int harta) + int totalWealth() + int countKKP() + double countTax() + int getTax() + void payTax(Player &p, Player &walkot) </td> </tr> </table>	Tax	# int ktkp # int uang # int harta + Tax(int ktkp, int uang, int harta) + int totalWealth() + int countKKP() + double countTax() + int getTax() + void payTax(Player &p, Player &walkot)	<p>Class Shop</p> <p>Collaboration Diagram</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; background-color: #f0f0f0;">Shop</td> </tr> <tr> <td style="padding: 5px;"> + Shop() + Shop(vector< tuple< Building *, int > < building, vector< tuple< Product *, int > > product) + Shop & operator+(Item *item) + vector< tuple< Building *, int > > getItemsBuilding () const + vector< tuple< Product *, int > > getItemsProduct () const + int totalItem() + void addBuilding(Building *b) + void minBuilding(Building b, int num) + void addProduct(Product *p) + void minProduct(Product p, int num) + void minItems(Item &item, int num) + int getCapacity(Item &item) + Item * getItem(int i) + Item * getNewItem(int i) + void printShop() + bool isBuilding(Item &item) </td> </tr> </table>	Shop	+ Shop() + Shop(vector< tuple< Building *, int > < building, vector< tuple< Product *, int > > product) + Shop & operator+(Item *item) + vector< tuple< Building *, int > > getItemsBuilding () const + vector< tuple< Product *, int > > getItemsProduct () const + int totalItem() + void addBuilding(Building *b) + void minBuilding(Building b, int num) + void addProduct(Product *p) + void minProduct(Product p, int num) + void minItems(Item &item, int num) + int getCapacity(Item &item) + Item * getItem(int i) + Item * getNewItem(int i) + void printShop() + bool isBuilding(Item &item)	<p>Class Misc</p> <p>Collaboration Diagram</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; background-color: #f0f0f0;">Misc</td> </tr> <tr> <td style="padding: 5px;"> + Misc() + Misc(int minMoney, int minWeight, int storageRow, int storageCols, int fieldRow, int fieldCols, int barnRow, int barnCols) + int getminMoney() + int getminWeight() + int getStorageRow() const + int getStorageCols () const + int getFieldRow() const + int getFieldCels() const + int getBarnRow() const + int getBarnCols() const </td> </tr> </table>	Misc	+ Misc() + Misc(int minMoney, int minWeight, int storageRow, int storageCols, int fieldRow, int fieldCols, int barnRow, int barnCols) + int getminMoney() + int getminWeight() + int getStorageRow() const + int getStorageCols () const + int getFieldRow() const + int getFieldCels() const + int getBarnRow() const + int getBarnCols() const
Tax								
# int ktkp # int uang # int harta + Tax(int ktkp, int uang, int harta) + int totalWealth() + int countKKP() + double countTax() + int getTax() + void payTax(Player &p, Player &walkot)								
Shop								
+ Shop() + Shop(vector< tuple< Building *, int > < building, vector< tuple< Product *, int > > product) + Shop & operator+(Item *item) + vector< tuple< Building *, int > > getItemsBuilding () const + vector< tuple< Product *, int > > getItemsProduct () const + int totalItem() + void addBuilding(Building *b) + void minBuilding(Building b, int num) + void addProduct(Product *p) + void minProduct(Product p, int num) + void minItems(Item &item, int num) + int getCapacity(Item &item) + Item * getItem(int i) + Item * getNewItem(int i) + void printShop() + bool isBuilding(Item &item)								
Misc								
+ Misc() + Misc(int minMoney, int minWeight, int storageRow, int storageCols, int fieldRow, int fieldCols, int barnRow, int barnCols) + int getminMoney() + int getminWeight() + int getStorageRow() const + int getStorageCols () const + int getFieldRow() const + int getFieldCels() const + int getBarnRow() const + int getBarnCols() const								
<p>Class Saver</p> <p>Collaboration Diagram</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; background-color: #f0f0f0;">Saver</td> </tr> <tr> <td style="padding: 5px;"> + void saveGameState (string filepath, const vector< Player *> &, const Shop &) + static string getTestsPath() </td> </tr> </table>	Saver	+ void saveGameState (string filepath, const vector< Player *> &, const Shop &) + static string getTestsPath()						
Saver								
+ void saveGameState (string filepath, const vector< Player *> &, const Shop &) + static string getTestsPath()								

1.3. Class Design

1.3.1. Item

Kelas ini adalah kelas yang memiliki *child classes* yang juga memiliki *child classes*, menjadikan kelas ini menjadi salah satu bagian besar dan penting dalam program ini. Kelas ini utamanya menerapkan konsep inheritance, polymorphism, dan juga operator overloading. Class Item memiliki *child classes* yaitu Animal, Plant, Product, dan Building, yang masing-masing meng-*inherit* beberapa *child classes* lagi, dan seterusnya hingga kelas tak bisa menurunkan suatu kelas lagi.

Kelebihan kelas ini adalah memberikan abstraksi yang jelas dan modular dalam pengembangan tugas program berorientasi objek ini. Keberadaan kelas Item dan seluruh kelas turunannya memudahkan pendefinisian elemen-elemen dalam permainan kelola kerajaan ke dalam suatu kelas-kelas dan objek. Dengan kelas Item dan turunannya yang didesain hingga menurunkan kelas yang spesifik, penggunaan kelas seperti instansiasi dan konstruksi objek dapat dilakukan dengan lebih mudah. Misalnya dengan kelas spesifik yang secara umum memiliki atribut statik, penggunaan objek tidak lagi memerlukan parameter detail atribut dari kelas atau objek tersebut dan dapat dipanggil dengan lebih mudah. Kekurangan dari kelas ini dan turunannya, adalah memerlukan pembuatan kelas yang banyak, bahkan kelas-kelas spesifik dalam suatu kelas (misal Herbivore dari Animal) memiliki atribut yang serupa satu sama lain dan hanya dibedakan konstruktor saja.

1.3.2. Tax

Kelas Tax menangani kalkulasi pajak dari Players, penghitungan pajak dibuat dengan kelas khusus agar seluruh elemen dalam permainan dimodelkan oleh suatu kelas dan objek tertentu (*we see everything as an object*). Sementara kekurangan kelas ini adalah bahwa metode dalam kelas ini dapat diimplementasikan di kelas Player saja. Dengan mempertimbangkan prinsip Single Responsibility, kami membentuk kelas pajak.

1.3.3. Misc

Kelas Misc merupakan kelas utilitas yang menyimpan informasi tambahan yang digunakan dalam permainan. Kelebihan kelas ini adalah menjadikan atribut-atribut pada kelas Misc lebih mudah dimuat dari file konfigurasi juga ter-*attach* langsung dengan Game. Adapun kekurangan kelas ini adalah tidak mencerminkan suatu objek nyata dalam konteks pemrograman berorientasi objek. Atribut-atribut pada kelas ini dapat secara terpisah-pisah dimuat pada kelas Game maupun Player dan kelas anaknya.

1.3.4. Player

Kelas Player merupakan kelas yang mewakili pemain dalam permainan. Kelas ini bertanggung jawab atas aktivitas yang melibatkan pemain dalam permainan, seperti mengelola inventaris, melakukan transaksi dengan item, dan menjalankan aksi tertentu yang berkaitan dengan peran atau profesi dalam permainan. Konsep *Object Oriented Programming* banyak dipakai dalam Kelas Player seperti inheritance dan polymorphism, STL, dan *Abstract Base Class* (ABC). Kelas Player memiliki 3 *child class*, yaitu Kelas Mayor, Kelas PlantFarmer, dan Kelas AnimalFarmer yang memiliki karakter dalam permainan dengan cara yang sesuai dengan karakteristik unik dari masing-masing profesi.

Kelebihan kelas ini adalah menerapkan dengan baik paradigma berorientasi objek dengan abstraksi yang jelas dan modular. Sementara pada praktiknya, penerapan kelas ini kadang memerlukan *dynamic_cast* untuk menggunakan method dari *child class* Player.

1.3.5. Game

Kelas Game bertanggung jawab terhadap logika utama permainan. Kelas ini menerapkan konsep inheritance dan juga STL. Kelas Game menjadi *parent class* bagi berbagai kelas *commands*. Kelebihan kelas ini adalah menyimpan seluruh logika utama permainan sehingga tidak tercerer dan dengan optimal memanfaatkan kelas dan objek yang dibuat dalam program. Hal yang mungkin dapat diperbaiki adalah menjadikan syarat *end game* seperti berat dan uang minimum menjadi atribut statik pada kelas ini, sehingga tidak perlu melakukan import dan instansiasi kelas Misc.

1.3.6. Command

Kelas Command bertanggung jawab terhadap seluruh aksi yang dapat dilakukan pemain dalam permainan ini melalui input yang diberikan di CLI. Kelas ini menerapkan *inheritance* dan *Abstract Base Class*. Kelas ini mengibaratkan pemanggilan *command* sebagai suatu objek serta menjadikan program lebih modular.

1.3.7. Exception

Kelas Exception bertanggung jawab untuk menangani kesalahan eksekusi program dengan membuat Exception Object sesuai dengan tipe error yang diterima. Implementasi kelas Exception menggunakan konsep inheritance dan polymorphism untuk menggeneralisir tipe objek pada child class yang diinstansiasi. Penggunaan kelas ini sangatlah baik untuk mencegah program tidak berjalan sesuai yang diinginkan karena alasan-alasan tertentu. Kelas ini tidak dibuat dengan file .cpp untuk alasan hanya ada satu method untuk masing-masing kelas Exception.

1.3.8. FileIO

FileIO terdiri dari kelas Loader yang bertanggung jawab untuk aktivitas muat konfigurasi dan state permainan dari file .txt dalam folder serta kelas Saver untuk menyimpan state permainan terbaru ke suatu file .txt. Kelas-kelas ini menerapkan STL dan mengimpor berbagai kelas dalam program ini. Kegalauan dalam memilih desain kelas FileIO ini adalah seperti pemisahan kelas Load dan Save, bentuk *return type* dari setiap load konfigurasi maupun state, dan apakah perlu adanya metode-metode yang dikhususkan membuat instansiasi objek-objek (*item constructor*). Oleh karena alasan kesesuaian dengan kelas-kelas lain pada program ini, kami membuat pemisalan untuk kelas Load dan Save (S dalam SOLID), dan return type berupa vector of Item, void, dan lainnya yang sesuai.

1.3.9. Shop

Kelas Shop menggunakan konsep STL Vector, STL Tuple, serta operator overloading dalam implementasinya. Penggunaan STL Vector pada kelas Shop bertujuan untuk menyimpan Hewan dan Tumbuhan yang jumlah itemnya tak terbatas. Penggunaan STL Tuple pada kelas Shop bertujuan untuk mengelompokkan pasangan nilai terkait seperti Building dan jumlahnya dalam `vector<tuple<Building*, int>>`, serta Product dan jumlahnya dalam `vector<tuple<Product*, int>>`. Hal ini memudahkan penyimpanan dan pengaksesan informasi terstruktur tentang bangunan, produk, serta jumlahnya dalam toko secara efisien dan mudah dibaca. Kelas Shop memiliki atribut-atribut yang statik karena sifatnya yang dalam permainan ini hanya terdapat satu Shop (tidak memerlukan banyak instansiasi objek). Hal tersebut juga memudahkan proses muat konfigurasi maupun *updating* barang-barang di dalam toko.

1.3.10. Storage

Kelas Storage mewakili seluruh bentuk penyimpanan yang ada di permainan, yaitu Ladang, Peternakan, dan Penyimpanan. Kelas Storage menggunakan konsep template, kelas generik, STL Vector, STL Map, serta operator overloading dalam implementasinya. Penggunaan konsep kelas generik bertujuan untuk mengabstraksi tipe data yang diagregasi pada kelas Storage ini. Penginstansiasian kelas Storage dengan template class digantikan dengan pointer to Animal akan menghasilkan objek Peternakan yang diagregasi oleh AnimalFarmer(Peternak). Penginstansiasian kelas Storage dengan template class digantikan dengan pointer to Plant akan menghasilkan objek Ladang yang diagregasi oleh PlantFarmer(Petani). Penginstansiasian kelas Storage dengan template class digantikan dengan pointer to Item akan menghasilkan objek Ladang yang diagregasi oleh seluruh tipe pemain.

Penggunaan STL Vector pada Storage mendukung pembuatan sistem penyimpanan yang bersifat grid. Penggunaan STL Map pada Storage memungkinkan pemanggilan objek-objek yang tersimpan di Storage berdasarkan kode slot lokasi tersimpannya objek. Keuntungan utama dari penggunaan STL Vector dan Map ini adalah fleksibilitas dalam menambah dan mengakses elemen-elemen yang digunakan, serta kemampuan untuk mengelola informasi secara efisien dan terstruktur. Kelas Storage menggunakan operator overloading operator + untuk menambahkan objek ke dalam inventory dengan kondisi tidak membutuhkan informasi lokasi objek akan diletakkan. Operator ini akan meletakkan objek di slot kosong yang tersedia. Slot yang dipilih menjadi lokasi peletakan objek adalah slot kosong yang pertama kali ditemukan program dari pengiterasian seluruh slot inventory.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep Inheritance dan Polymorphism digunakan di beberapa kelas pada program. Inheritance digunakan untuk mengabstraksikan kelas yang terbentuk dan memungkinkan polymorphism dalam pemrograman berorientasi objek ini. Polymorphism digunakan untuk mempermudah pengolahan suatu objek dengan cukup menggunakan tipe kelas dasarnya saja. Penerapan Inheritance dan Polymorphism digunakan di banyak kelas seperti Player, Item dan kelas-kelas turunannya, dan juga kelas Game dan Command beserta kelas-kelas turunannya.

2.1.1. Class Player

```

1  class Player{
2      protected:
3          string username;
4          int wealth;
5          int weight;
6          string type;
7          StorageItem* inventory;
8          vector<Command*> commandList;
9
10     public:
11         // VIRTUAL METHOD
12         virtual void displayInfo() const;
13         virtual int getPlayerAssets()=0;
14         virtual int getKTP()=0;
15         virtual int calculateTax()=0;
16
17         // CLASS CONSTRUCTOR AND DESTRUCTOR
18         Player();
19         Player(string username, int wealth, int weight, string type);
20         Player(const Player& other);
21         virtual ~Player();
22
23         // GETTER AND SETTER
24         string getName() const;
25         int getPlayerWealth() const;
26         int getPlayerWeight() const;
27         string getType() const;
28         vector<Item*> getListOfItems() const;
29         void setInventory(Storage<Item*>* );
30
31         // CLASS METHOD
32         // Player Storage Method
33         void displayStorage();
}

```

```

class Mayor : public Player{
    public:
        // CONSTRUCTOR
        Mayor();
        Mayor(string username, int wealth, int weight);

        // GETTER
        int getKTP();
        int getPlayerAssets();

        // CLASS METHOD
        int calculateTax();
        void buildBuilding();
        bool checkMaterialCapacity(map<string, int> material);
};

```

```

● ● ●

1 class PlantFarmer : public Player{
2     private:
3         Storage<Plant*> Garden;
4     public:
5         // CONSTRUCTOR
6         PlantFarmer();
7         PlantFarmer(string username, int wealth, int weight);
8
9         // GETTER AND SETTER
10        int getPlayerAssets();
11        int getTKP();
12        vector<Plant*> getListOfPlants() const;
13        void setGarden(Storage<Plant*>);
14
15        // CLASS METHOD
16        void addPlantYear();
17        void displayInfo() const;
18        void plantCrop();
19        void harvestCrop();
20        void printGarden();
21        int calculateTax();
22    };

```



```

● ● ●

1 class AnimalFarmer : public Player{
2     private:
3         Storage<Animal*> Barn;
4     public:
5         // CONSTRUCTOR
6         AnimalFarmer();
7         AnimalFarmer(string username, int wealth, int weight);
8
9         // GETTER
10        int getTKP() override;
11        void setBarn(Storage<Animal*>);
12        int getPlayerAssets();
13        vector<Animal*> getListOfAnimals() const;
14
15        // CLASS METHOD
16        void displayInfo() const override;
17        void placeAnimal();
18        void feedAnimal();
19        void harvestAnimal();
20        void printBarn();
21        int calculateTax();
22    };

```

Pada class Player, terdapat tiga *child* yang di-*inherit* dari Player yaitu class Mayor, PlantFarmer, dan AnimalFarmer. Ketiga *child class* memiliki sifat seperti *base class* atau *parent class* nya yaitu Player yang dapat melakukan makan, membeli, dan menjual dan memiliki *method* dan *attribute* spesifik yang dimiliki dirinya.

2.1.2. Class Item

```
● ● ●
class Item{
protected:
    int id;
    string code;
    string name;
    string type;
    int price;
public:
    Item();
    Item(string code, string name, string type);
    Item(int id, string code, string name, string type, int price);
    virtual ~Item();
    bool operator==(Item& Item2) const;
    bool isReadyToHarvest();
    int getPrice() const;
    string getCode() const;
    string getName() const;
    string getType() const;
};
```

Pada class Item, terdapat empat *child* yang di-*inherit* dari Item yaitu class Animal, Plant, Product, dan Building. Seluruh *child class* memiliki atribut *base class* atau *parent class* nya yaitu Item seperti id, kode, nama, tipe, dan juga harga.

```
● ● ●
class Animal : public Item{
protected:
    int animalID;
    int animalWeight;
    int harvestWeight;
    int price;

public:
    Animal(string code, string name, string type, int harvestWeight, int price);
    ~Animal();
    virtual void makan(Product* p);
    void setAnimalWeight(int weight);
    int getPrice();
    bool isReadyToHarvest();
};
```

```
● ● ●
class Plant : public Item{
protected:
    int plantID;
    int harvestDuration;
    int plantAge;
    int price;

public:
    Plant(int plantID, string plantCode, string name, string type, int harvestDuration, int price);
    ~Plant();
    int getPrice();
    void addAge();
    void setPlantAge(int plantAge);
    bool isReadyToHarvest();
};
```

```
● ● ●
class Building : public Item {
private:
    int price;
    map<string, int> materials;
    vector<Building*> houses;
public:
    Building();
    Building(int id, string code, string name, string type, int price);
    ~Building();
    int getMoney(string s);
    Building* getBuilding (string s);
    virtual void printMaterial();
    virtual map<string, int> getmaterial(string b);
};
```

```
● ● ●
class Product : public Item {
private:
    int productID;
    int addedWeight;
    int priceProduct;
    string origin;
    vector<Product*> product;

public:
    Product();
    Product(int productID, string code, string name, string type, string origin, int added_weight, int priceProduct);
    virtual ~Product();
    int getAddedWeight();
    int getPrice();
    vector<Product*> getProduct(string prod);
};
```

2.2. Exception dan Exception Handling

2.2.1. Class Exception



```
● ● ●
1 class Exception {
2 public:
3     virtual string what() = 0;
4 };
5
6 class InputException : public Exception {
7 public:
8     string what() override {
9         return "\u033[1;31mInput is not valid.\nPlease try another input.\n\u033[0m";
10    }
11 };
12
13 class InputNotIntegerException : public Exception {
14 public:
15     string what() override {
16         return "\u033[1;31mInput is not valid.\nPlease enter only an integer.\n\u033[0m";
17    }
18 };
19
20 class CommandException : public Exception {
21 public:
22     string what() override {
23         return "\u033[1;31mWhoops! That command seems to have lost its way. \nHow about a different one?\n\u033[0m";
24    }
25 };
```

Implementasi kelas exception digunakan untuk mengelola dan menangani kondisi-kondisi tertentu yang abnormal atau tidak diinginkan yang mungkin terjadi selama eksekusi program. Gambar di atas adalah beberapa exception yang diimplementasikan pada program. Ketika suatu kondisi yang memerlukan validasi ditemukan, seperti saat mencoba untuk memasukkan input yang salah, nantinya instance dari InputException akan dilempar untuk menunjukkan kesalahan tersebut kepada pemanggil kode. Dengan demikian, menggunakan kelas exception kustom memungkinkan untuk memberikan pesan kesalahan yang spesifik pada kode, sehingga memudahkan dalam menangani dan memahami kondisi yang terjadi.

2.2.2. Exception Handling

Penggunaan penanganan exception dengan blok try-catch banyak digunakan dalam pembuatan program, terutama untuk melakukan validasi input dari pengguna. Pendekatan ini lebih efisien dibandingkan dengan penggunaan struktur kondisional (if-else) secara manual untuk validasi. Contoh penggunaan exception adalah sebagai berikut:



```
● ● ●
1 string Game::inputName(){
2     string name;
3     while (true){
4         cout << "Enter username: ";
5         cout << "\033[1;92m";
6         getline(cin, name);
7         cout << "\033[0m";
8         try{
9             if(!nameNotValid(name)){
10                 return name;
11                 break;
12             }
13         }
14         catch(UsernameException e){
15             cout<<e.what()<<endl;
16         }
17     }
18 }
```

Dalam metode `inputName()` yang terdapat di kelas `Game`, blok try-catch digunakan untuk mengatasi situasi di mana input nama yang diberikan tidak valid. Jika terjadi kesalahan, sebuah `UsernameException` akan dilemparkan, kemudian akan ditangkap dalam blok `catch`, dan pesan kesalahan akan ditampilkan kepada pengguna untuk memberitahu bahwa nama yang dimasukkan tidak sesuai dengan persyaratan yang ditentukan.



```
1 void Loader::stateOfShop(string statepath, int ctr){
2     vector<vector<string>> vectorOfWords;
3     vector<tuple<Building*, int>> building;
4     vector<tuple<Product*, int>> product;
5     try {
6         vectorOfWords = Loader::getWordsFromFile(statepath);
7
8         int numOfShopItems = stoi(vectorOfWords[ctr][0]); ctr++;
9         for(int i = 0; i < numOfShopItems; i++){
10             string itemName = trim(vectorOfWords[ctr][0]);
11             int itemQty = stoi(vectorOfWords[ctr][1]);
12             ctr++;
13             if(isBuilding(itemName)){
14                 building.push_back(make_tuple(buildingConstructor(itemName), itemQty));
15             }
16             else{
17                 product.push_back(make_tuple(productConstructor(itemName), itemQty));
18             }
19         }
20         Shop(building, product);
21     } catch (FileNotFoundException &FE){
22         cout << FE.what();
23     }
24 }
```

Selain digunakan dalam method utama untuk validasi input dalam fitur utama game, exception juga penting dalam pengaturan konfigurasi dan pemrosesan berkas pada program. Dalam contoh program yang disebutkan, penggunaan exception terkait dengan validasi lokasi path file memungkinkan program untuk mengidentifikasi dan menangani kondisi di mana path yang dimasukkan tidak valid atau file tidak ditemukan. Ketika exception seperti FileNotFoundException dilemparkan, blok catch akan menangkapnya dan mengeluarkan pesan kesalahan yang informatif kepada pengguna, memungkinkan program untuk memberikan respons yang sesuai terhadap masalah yang terjadi selama eksekusi.

2.3. Method/Operator Overloading

2.3.1. Operator ==

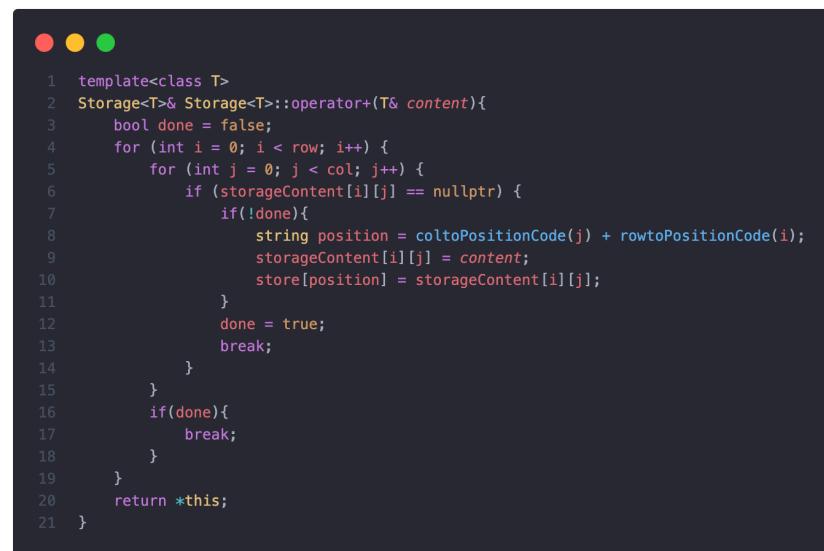
```
● ● ●  
1 bool Player::operator==(string command) const{  
2     bool ada = false; // cari command punya player atau bukan  
3     for (auto &c : commandList){  
4         if (c->getName() == command){  
5             ada = true; // command ada  
6             c->useCommand();  
7         }  
8     }  
9     return ada;  
10 }
```

Di dalam metode Player::operator==, digunakan operator == untuk membandingkan string command dengan nama command yang terdaftar dalam commandList milik Player. Apabila ada command yang cocok dengan string command, maka command tersebut akan digunakan melalui pemanggilan c->useCommand().

```
● ● ●  
1 bool Item::operator==(Item& Item2) const{  
2     if(this->code == Item2.getCode() && this->name == Item2.getName() && this->type==Item2.getType()){  
3         return true;  
4     }  
5     return false;  
6 }
```

Penggunaan operator overloading == dalam kelas Player membandingkan objek Player dengan string memudahkan dalam memeriksa apakah suatu command terdapat dalam daftar command yang dimiliki oleh player. Hal ini memungkinkan penggunaan yang lebih sesuai dengan konvensi umum dalam pemrograman, di mana operator == digunakan untuk membandingkan objek dengan tipe data lain. Penggunaan operator == juga digunakan pada kelas Item untuk membandingkan dua objek item.

2.3.2. Operator +



```
● ● ●
1 template<class T>
2 Storage<T>& Storage<T>::operator+(T& content){
3     bool done = false;
4     for (int i = 0; i < row; i++) {
5         for (int j = 0; j < col; j++) {
6             if (storageContent[i][j] == nullptr) {
7                 if(!done){
8                     string position = coltoPositionCode(j) + rowtoPositionCode(i);
9                     storageContent[i][j] = content;
10                    store[position] = storageContent[i][j];
11                }
12                done = true;
13                break;
14            }
15        }
16        if(done){
17            break;
18        }
19    }
20    return *this;
21 }
```

Di dalam metode Storage::operator+, digunakan operator + untuk menambahkan objek ke dalam inventory. Operator ini dapat digunakan untuk menambahkan objek Item ke ‘Penyimpanan’, menambahkan objek Animal ke

'Peternakan', dan menambahkan objek Plant ke 'Ladang' dengan kondisi peletakkan yang tidak membutuhkan informasi lokasi spesifik objek akan diletakkan. Operator ini akan meletakkan objek di slot kosong yang tersedia. Slot yang dipilih menjadi lokasi peletakkan objek adalah slot kosong yang pertama kali ditemukan program dari pengiterasian seluruh slot inventory.



```
Shop& Shop::operator+(Item* item) {
    if(item->getType() == "CARNIVORE" || item->getType() == "OMNIVORE" ||
       item->getType() == "HERBIVORE" || item->getType() == "MATERIAL_PLANT" ||
       item->getType() == "FRUIT_PLANT"){
    }
    else if(item->getType() == "BUILDING"){
        Building* building = dynamic_cast<Building*>(item);
        addBuilding(building);
    }
    else{
        Product* prod = dynamic_cast<Product*>(item);
        addProduct(prod);
    }
    return *this;
}
```

Operator + dalam kelas Shop digunakan untuk penambahan objek Item ke dalam stok toko. Dengan menggunakan operator ini, dapat dengan mudah menambahkan jumlah stok item yang tersedia dalam toko. Terutama pada item Building dan Product. Penggunaan operator + memungkinkan penanganan operasi penambahan stok secara lebih efisien.

2.4. Template & Generic Classes

```
1 template<class T>
2 class Storage {
3     private :
4         int row;
5         int col;
6         vector<vector<T> > storageContent;
7         map<string, T> store;
8     public :
9         Storage();
10        Storage(int row, int col);
11        ~Storage();
12        void setItem(int row, int col, T content);
13        void setItemString(string position, T content);
14        void setItemRandom(T content);
15        Storage<T>& operator+(T& content);
16 }
```

Peternakan yang dimiliki Peternak, Ladang yang dimiliki Petani, dan Penyimpanan yang dimiliki oleh semua tipe pemain memiliki kesamaan fungsi, yaitu untuk menyimpan objek-objek terkait dan tentunya menampilkan penyimpanan melalui perintah CETAK_PENYIMPANAN, CETAK_LADANG, dan CETAK_PETERNAKAN dengan format yang sama. Kesamaan fungsi dari Peternakan, Ladang, dan Penyimpanan membuat penerapan kelas generik menjadi relevan untuk efektivitas instansiasi dari setiap kelas terkait.

Kelas Storage dibuat menggunakan *prefix* “template<class T>”. Kelas T yang dapat digunakan untuk menginstansiasi kelas ini adalah Animal* untuk menginstansiasi Peternakan, Plant* untuk menginstansiasi Ladang, dan Item* untuk menginstansiasi Penyimpanan.

2.5. C++ Standard Template Library

2.5.1. Vector dan Tuple pada class Shop

```
● ● ●  
1 class Shop {  
2 private:  
3     static vector<tuple<Building*, int>> itemsBuilding;  
4     static vector<tuple<Product*, int>> product;  
5     vector<Plant*> itemsPlants;  
6     vector<Animal*> itemsAnimals;  
7 };
```

Penggunaan STL vector dan tuple salah satunya digunakan dalam kelas Shop memberikan pendekatan yang efisien untuk menyimpan dan mengelola data toko. vector digunakan untuk menyimpan Hewan dan Tumbuhan yang jumlah itemnya tak terbatas. Sementara itu, tuple digunakan untuk mengelompokkan pasangan nilai terkait seperti Building dan jumlahnya dalam vector<tuple<Building*, int>>, serta Product dan jumlahnya dalam vector<tuple<Product*, int>>. Hal ini memudahkan penyimpanan dan pengaksesan informasi terstruktur tentang bangunan, produk, serta jumlahnya dalam toko secara efisien dan mudah dibaca.

2.5.2. Vector dan Map pada class Building

```
1 class Building : public Item {  
2     private:  
3         int price;  
4         map<string, int> materials;  
5         vector<Building*> houses;
```

Penggunaan STL map dan vector dalam kelas Building memberikan pendekatan yang fleksibel dan efisien dalam manajemen data bangunan. Penggunaan `map<string, int> materials` memungkinkan menyimpan bahan-bahan yang dibutuhkan untuk membangun suatu bangunan berdasarkan nama bahan dan jumlahnya. Selain itu, juga digunakan `vector<Building*> houses` memungkinkan Building untuk menyimpan koleksi bangunan yang ada. Keuntungan utama dari STL ini adalah fleksibilitas dalam menambah dan mengakses elemen-elemen yang digunakan, serta kemampuan untuk efisien mengelola informasi terstruktur seperti bahan-bahan yang diperlukan untuk membangun bangunan. Hal ini mempermudah pengembangan sistem yang terorganisir dan dapat juga diperluas.

2.5.3. Vector dan Map pada class Storage

```
● ● ●  
1 template<class T>  
2 class Storage {  
3     private :  
4         int row;  
5         int col;  
6         vector<vector<T> > storageContent;  
7         map<string, T> store;
```

Penggunaan STL vector dan map dalam kelas Storage memberikan pendekatan yang fleksibel dan efisien dalam manajemen penyimpanan objek terkait pada seluruh media penyimpanan baik itu pada Peternakan, Ladang, maupun Penyimpanan. Penggunaan `vector<vector<T> >` pada setiap media penyimpanan memungkinkan pembuatan sistem penyimpanan yang bersifat grid. Penggunaan `map<string, T>` pada setiap media penyimpanan memungkinkan pemanggilan objek-objek yang tersimpan berdasarkan kode slot lokasi tersimpannya objek. Keuntungan utama dari STL ini adalah fleksibilitas dalam menambah dan mengakses elemen-elemen yang digunakan, serta kemampuan untuk mengelola informasi secara efisien dan terstruktur. Hal ini mempermudah pengembangan sistem yang terorganisir dan dapat juga dapat diperluas.

2.6. Konsep OOP lain

2.6.1. Abstract Base Class

2.6.1.1. Class Player

```
● ● ●
1 class Player{
2     protected:
3         string username;
4         int wealth;
5         int weight;
6         string type;
7         Storage<Item*> inventory;
8         vector<Command*> commandList;
9
10    public:
11        // VIRTUAL METHOD
12        virtual void displayInfo() const;
13        virtual int getPlayerAssets()=0;
14        virtual int getKTP()=0;
15        virtual int calculateTax()=0;
```

Penggunaan Abstract Base Class (ABC) dalam kelas Player membantu menyederhanakan program dan memisahkan bagian umum dari bagian khusus dari berbagai jenis pemain. Dengan menetapkan beberapa metode *pure virtual* seperti getPlayerAssets(), getKTP(), dan calculateTax(), dapat memberi informasi spesifik yang harus diberikan oleh setiap jenis pemain. Ini memungkinkan penggunaan polimorfisme untuk memproses pemain dengan cara yang sama, tanpa harus peduli dengan cara spesifik masing-masing pemain di dalamnya. Menggunakan ABC juga membuat kode lebih mudah diperluas, sehingga dapat menambahkan jenis pemain baru di masa depan tanpa mengganggu bagian kode yang sudah ada.

2.6.1.2. Class Command

```
1 class Command{
2 public:
3     virtual void useCommand() = 0;
4     virtual string getName() = 0;
5 };
```

Penggunaan Abstract Base Class (ABC) dalam kelas Command berguna untuk menyediakan method umum yang mendefinisikan method dasar yang harus diimplementasikan oleh setiap jenis command. Dengan menetapkan metode *pure virtual* seperti useCommand() dan getName(), ABC memastikan setiap turunan Command harus mengimplementasikan fungsionalitas ini sesuai dengan kebutuhan spesifiknya. Keuntungan utamanya adalah meningkatkan modularitas kode dan mudah menambahkan jenis command baru tanpa mengubah struktur yang sudah ada.

2.6.1.3. Class Exception

```
1 class Exception {  
2     public:  
3         virtual string what() = 0;  
4 };
```

Penggunaan Abstract Base Class (ABC) dalam kelas Exception berguna untuk menangani berbagai jenis pengecualian, seperti validasi input, command, file, dan loader. Dalam kelas ini, parent class Exception tidak dapat diinstansiasi. Kelas ini memiliki satu metode pure virtual, yaitu string what(), yang memastikan setiap kelas turunan Exception telah mengimplementasikan fungsionalitasnya.

2.6.2. Aggregation

2.6.2.1. Class Player

```
● ● ●  
1 class Player{  
2     protected:  
3         string username;  
4         int wealth;  
5         int weight;  
6         string type;  
7         Storage<Item*> inventory;  
8         vector<Command*> commandList;
```

Penggunaan konsep aggregation dalam kelas Player, terutama dengan `Storage<Item*> inventory` dan `vector<Command*> commandList`, bermanfaat karena Player dapat memiliki hubungan dengan objek Storage dan Command tanpa harus memiliki kepemilikan penuh terhadap kelas tersebut. Hal ini memisahkan method yang ada pada Player dan Storage, memungkinkan fleksibilitas dalam pengelolaan inventory dan method yang terkait dengan pemain. Aggregation juga mendukung desain yang lebih modular dan memungkinkan pengembangan yang lebih mudah diperluas serta disesuaikan dengan kebutuhan, karena perubahan pada inventory atau command tidak mempengaruhi struktur inti dari kelas Player secara langsung.

2.6.2.2. Class Game

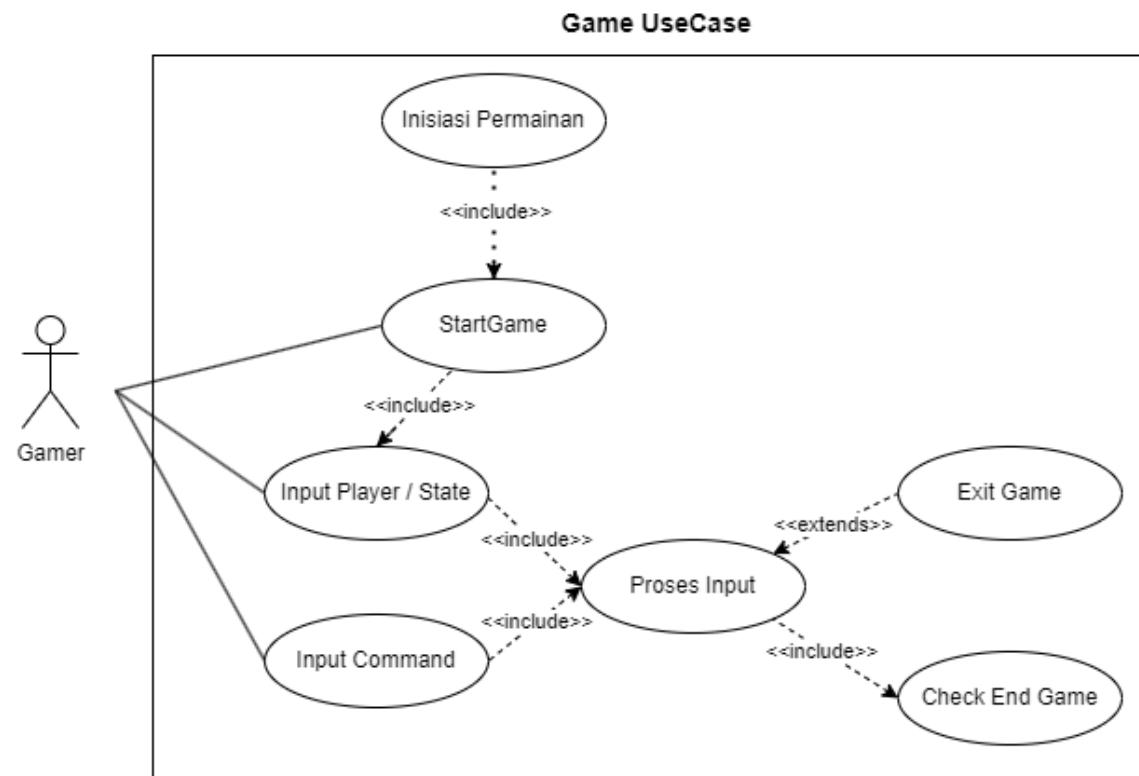
```
● ● ●  
1 class Game{  
2     protected:  
3         static int currPlayer;  
4         static int tambahPemain;  
5         static vector<Player*> players;  
6     };
```

Dalam kelas Game, penggunaan konsep aggregation dengan atribut `vector<Player*> players` memungkinkan Game untuk memiliki dan mengelola kumpulan objek Player sebagai bagian dari permainan. Keuntungan penggunaan aggregation di sini adalah fleksibilitas dalam mengelola daftar pemain dalam permainan, kemudahan dalam manipulasi dan akses terhadap objek Player, serta pemisahan method antara Game dengan Player. Dengan pendekatan ini, Game dapat memiliki struktur yang modular dan mudah diperluas sesuai kebutuhan permainan yang berkembang tanpa mempengaruhi inti dari kelas Game maupun kelas Player.

3. Bonus Yang Dikerjakan

3.1. Bonus yang Diusulkan oleh Spek

3.1.1. Diagram Use Case



3.2. Bonus Kreasi Mandiri

```
● ● ●  
1 class Risk{  
2     private:  
3         vector<Risk*> risk;  
4     public:  
5         void initializeRisks();  
6         void getRandomRisk(Player*);  
7         virtual void useRisk(Player*);  
8         virtual string getName();  
9     };
```

Kami menambahkan fitur ‘RISK’ pada permainan untuk memudahkan petani dan peternak dalam memenangkan permainan. Seperti yang diketahui kesempatan walikota untuk mendapat uang cukup besar karena dengan memungut pajak uang dan menjual bangunan uang yang didapat cepat bertambah banyak. Sedangkan, pada petani dan penjual uang dapat berkurang dengan mudah jika walikota terus melakukan pungut pajak. Padahal syarat untuk menang adalah uang dan berat dari pemain tercukupi. Pada risk ini pemain bisa mendapatkan keuntungan maupun kerugian, sehingga harus berhati-hati dalam pemakaiannya. Dan tiap gilirannya hanya bisa digunakan satu kali saja. Berikut beberapa risk yang dibuat:

3.2.1. Disease Outbreak

```
You take a risk!
You got Disease Outbreak
Your garden have been infected by a disease!
Your animal is getting sick, all of your animal weights decreased by 2
```

```
You take a risk!
You got Disease Outbreak
Your garden have been infected by a disease!
Your plant is getting sick, all of your plant years decreased by 1
```

Jika player mendapat risk Disease Outbreak, ladang atau peternakan pemain akan terinfeksi virus. Jika pemain adalah petani maka semua tanaman di ladang umurnya akan berkurang 1 dan jika pemain adalah perternak maka semua hewan pada peternakan akan berkurang 2. Semua berat hewan dan umur tumbuhan akan berkurang secara otomatis saat risk dipanggil.

3.2.2. Harvest Festival Bonus

```
You take a risk!
You got Harvest Festival Bonus
You've received a bonus from the harvest festival!
You got 50 extra gulden
```

Jika player mendapat risk Harvest Festival Bonus, player akan mendapatkan uang sebesar 50 gulden. Uang player akan bertambah secara otomatis saat risk dipanggil.

3.2.3. GoodFarmerBlessing

You take a risk!
You got Good Weather Blessing
Your farm is blessed because you are a good farmer!
As an plant farmer, you received a Apple Tree and 15 gulden as a reward!

You take a risk!
You got Good Weather Blessing
Your farm is blessed because you are a good farmer!
As an animal farmer, you received a Cow and 15 gulden as a reward!

Jika player mendapat risiko GoodFarmer Blessing, player akan mendapatkan uang sebesar 15 gulden dan 1 pohon apel jika merupakan seorang petani dan mendapat 1 sapi jika merupakan seorang peternak. Item yang didapatkan akan ditambahkan pada ladang atau peternakan secara otomatis. Namun, jika ladang / peternakan yang dimiliki penuh akan diberikan kompensasi berupa uang sebesar 20 gulden.

3.2.4. Equipment Breakdown

You take a risk!
You got Equipment Breakdown
Your farming equipment has broken down!
You have to pay 20 gulden for maintenance.
Deducting 20 gulden from your wealth.

Jika player mendapat risk Equipment Breakdown, player harus membayar biaya perbaikan sebesar 20 gulden. Jika uang player tersisa kurang dari 20 gulden maka uang yang tersisa akan dibayarkan sepenuhnya. Uang player akan dikurangi secara otomatis saat risk dipanggil.

3.2.5. Wild Animal Attacks

You take a risk!
You got Wild Animal Attacks
Wild animals are attacking your farm!
One of your plant was eaten by wild animal

Jika player mendapat risk Wild Animal Attacks, Peternakan / Ladang milik pemain akan diserang dan salah satu hewan / tumbuhan yang ada akan dimakan oleh hewan tersebut. Sehingga satu hewan pada peternakan bagi peternak atau satu tumbuhan pada petani akan hilang secara otomatis saat command risk dipanggil.

3.2.6. No Scam MLM

You take a risk!
You got MLM Anti Scam
You invested some money to join a multi-level marketing. Dreaming your farm to be as great as those in Nuu Zee Land.
You join an MLM and suddenly your wealth is double dripper.
Your before now is 1000 gulden.
Your wealth now is 2000 gulden. Thank God

Player mungkin mendapat risiko melakukan investasi bisnis di suatu MLM dan mendapatkan keuntungan tanpa *scam* dan penipuan. Jika player mendapat risk No Scam MLM, player petani / peternak akan mendapat kekayaan sejumlah kekayaannya saat ini sehingga memiliki kekayaan dua kali lipat.

3.2.7. Iftar Time

```
Enter command > RISK

You take a risk!
You got Iftar Time
It's iftar time in MarhabanTiba kingdom. The kingdom is giving you 3 chicken eggs and 3 duck eggs for your iftar.
===== [ Storage ] =====
 A   B   C   D   E   F   G   H   I   J
+---+---+---+---+---+---+---+---+---+
01 | CHE | ALW | CHE | DCE | CHE | DCE | CHE | DCE |
+---+---+---+---+---+---+---+---+---+
```

Kerajaan MarhabanTiba suka makan telur untuk berbuka puasa. Jika player mendapat risk Iftar Time, player petani / peternak akan mendapatkan tiga telur ayam dan tiga telur bebek.

3.2.8. Bedah Rumah Show

```
Enter command > RISK

You take a risk!
You got Bedah Rumah Show
Congrats! You got some surprise from MarhabanTiba Ent. They are renovating your houses to hotels.
===== [ Storage ] =====
 A   B   C   D   E   F   G   H   I   J
+---+---+---+---+---+---+---+---+---+
01 | COW | COM | COW | ALT | HTL |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
```

Bedah Rumah adalah salah satu tayangan favorit Marhaban Tiba Entertainment. Jika player mendapat risk Bedah Rumah Show, permainan akan menyulap seluruh rumah yang dimiliki player menjadi hotel.

3.2.9. Ramadan Diet

```
Enter command > RISK
```

You take a risk!

You got Ramadan Diet

It's Ramadan. You can't help but loss some of your weight due to fasting.

Your weight before was 20 kg. You loss 1 kg weight.

Your weight now is 19 kg. No worries, you may gain weight after eid!

Kerajaan MarhabanTiba berpuasa saat bulan Ramadan. Jika player mendapat risk Ramadan Diet, player petani / peternak akan kehilangan berat badan sebanyak 7.5% dari berat badan saat ini.

3.2.10. Gain Weight in Eid

```
You take a risk!
```

You got Gain Weight in Eid

It's Eid Mubarak. You can't handle your appetite. You accidentally gain weight.

Your weight before was 20 kg. You gain 1 kg weight.

Your weight now is 21 kg. Sure you want to diet but you need to win this game!

Kerajaan MarhabanTiba menyelenggarakan Idul Fitri. Pada perayaan yang disebut lebaran itu, petani dan peternak menjadi sejahtera akibat zakat fitrah. Jika player mendapat risk Gain Weight in Eid, player petani / peternak mengalami kenaikan berat badan sebanyak 5%.

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Next	13522025	13522035, 13522060, 13522069
Cetak Penyimpanan	13522060	13522025, 13522035, 13522069
Pungut Pajak	13522025	13522035, 13522069, 13522087
Cetak Ladang	13522060	13522025, 13522035, 13522069
Cetak Peternakan	13522060	13522025, 13522035, 13522069
Tanam	13522069, 13522060	13522025, 13522035, 13522087
Ternak	13522069, 13522060	13522025, 13522035, 13522087
Bangun Bangunan	13522025	13522035, 13522060, 13522069
Makan	13522069, 13522035	13522025, 13522060, 13522087
Memberi Pangan	13522025, 13522035	13522035, 13522060, 13522087
Membeli	13522035	13522025, 13522060, 13522069
Menjual	13522035	13522025, 13522060, 13522069
Memanen	13522069, 13522060	13522025, 13522035, 13522087

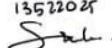
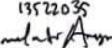
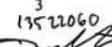
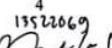
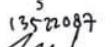
Muat	13522087	13522035, 13522060, 13522069
Simpan	13522087	13522035, 13522060, 13522069
Tambah Pemain	13522025	13522035, 13522060, 13522069
Exit (Additional)	13522060	13522035, 13522060, 13522087
Risk (Additional)	13522025, 13522087	13522025, 13522035, 13522087, 13522060, 13522069

LAMPIRAN

**Form Asistensi Tugas Besar
IF2210/Pemrograman Berorientasi Objek
Sem. I 2023/2024**

No. Kelompok/Kelas	: ODP / KIFK2 Ganeshia
Nama Kelompok	: Mahasiswa Tiba
Anggota Kelompok (Nama/NIM)	: <ul style="list-style-type: none"> 1. Andhitia Naura Hariyanto (13522060) 2. Debrina Veisha Rushika W (13522025) 3. Melati Anggraini (13522035) 4. Nabila Shikoota Mirda (13522069) 5. Shulha (13522087) 6.

Asisten Pembimbing: Morally Michael H L 1352052

Tanggal: 1 April 2024	Catatan Asistensi:
Tempat: Lab Pogramming - ITB Ganeshia	<ul style="list-style-type: none"> Storage lebih baik diimplementasi dengan kousen genetik dibandingkan dijadikan class. Buat print title storage sendiri saja. coba pertimbangkan opsi jadiin tax sebagai virtual function Saran dari Kak Michael adalah pakai operator + dan operator - untuk buat dan mengurangi size di class shop. Pertimbangkan mau pakai method isEatable() atau engga. Untuk menyimpan resep bangunan bisa coba buat variabel static final di game manager. - sebaiknya testing dengan config diasumsikan selalu valid. - Apabila ada penambahan pemain, urutan tetap dilakukan secara leksikografi. $ABCDE + 0 \Rightarrow ABCDE$ - Common mistake: deadliner, tidak mencatat laporan udah testing program.
Kehadiran Anggota Kelompok:	
No NIM Tanda tangan	1 13522025  2 13522035  3 13522060  4 13522069  5 13522087  6
	Tanda Tangan Asisten: