

Kode Kelompok : B88

Nama Kelompok : Bahari88

1. 13522123 / Jimly Nur Arif

2. 13522127 / Maulana Muhamad Susetyo

3. 13522134 / Shabrina Maharani

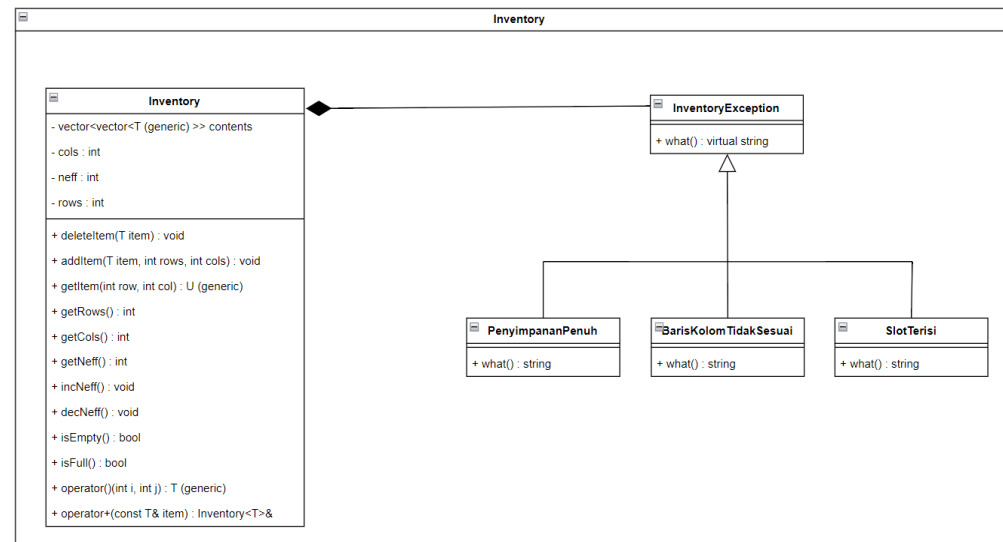
4. 13522151 / Samy Muhammad Haikal

5. 13522152 / Muhammad Roihan

Asisten Pembimbing : Primanda Adyatma Hafiz

## 1. Diagram Kelas

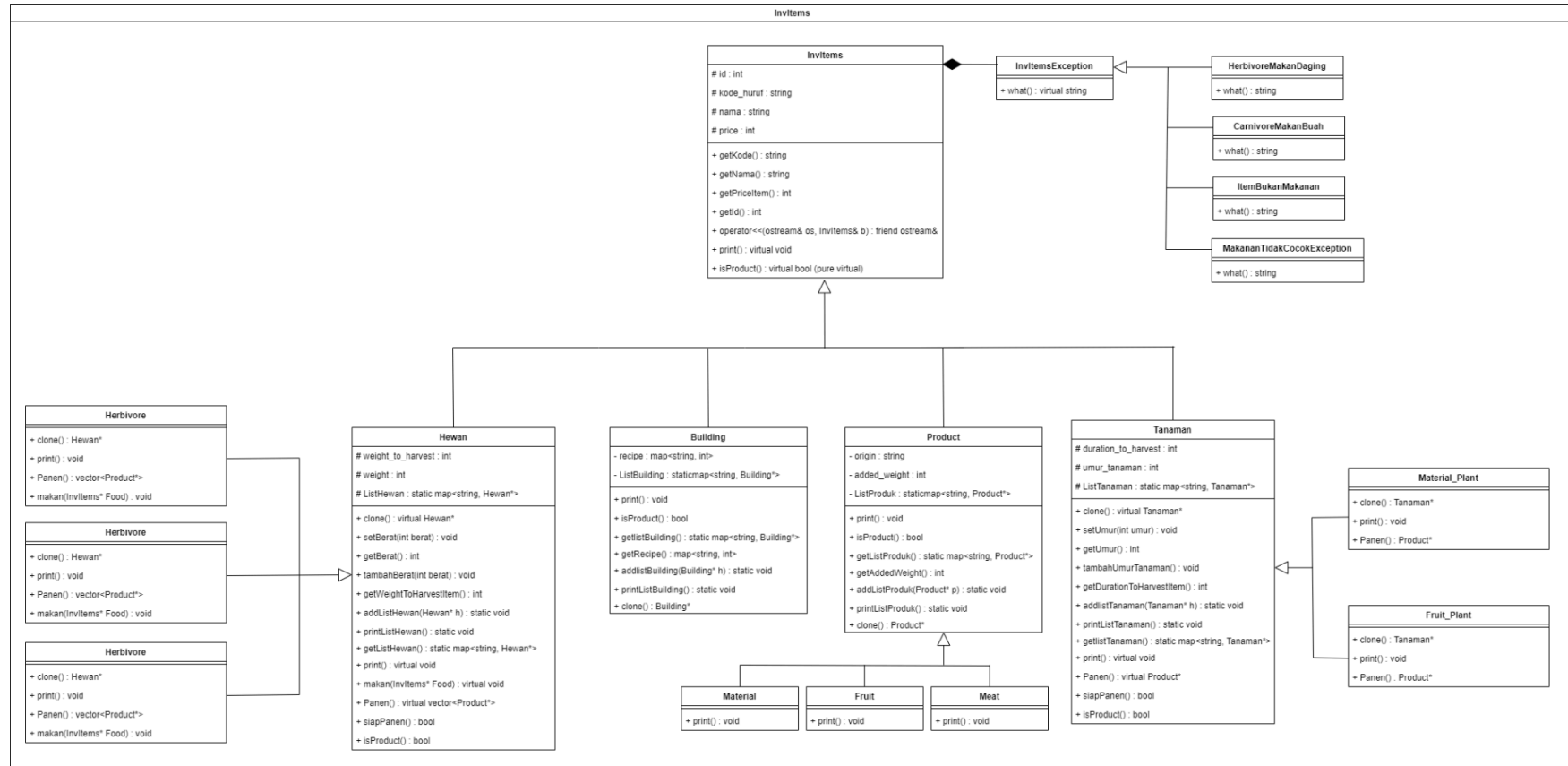
### 1.1 Inventory



Gambar 1.1 Diagram Kelas Inventory

Sumber : <https://app.diagrams.net/#G1eGHscqRWyWAOjddPk2gOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRYzhbsY4tBxP0kiT6n%22%7D> page All

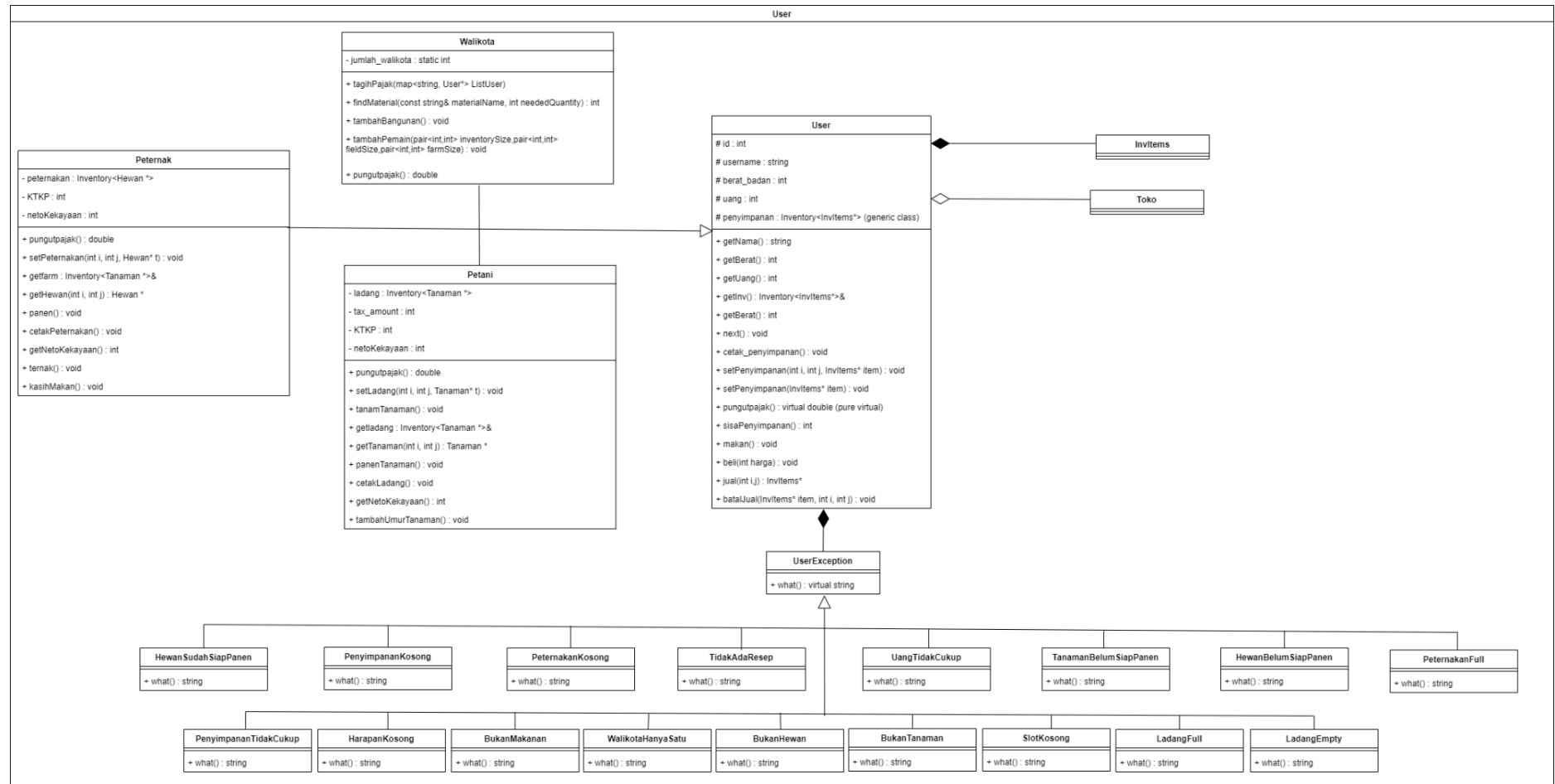
## 1.2 InvItems



Gambar 1.2 Diagram Kelas InvItems

Sumber : <https://app.diagrams.net/#G1eGHscqRWyWAOjddPk2gOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRYzhbsY4tBxP0kiT6n%22%7D> page InvItems

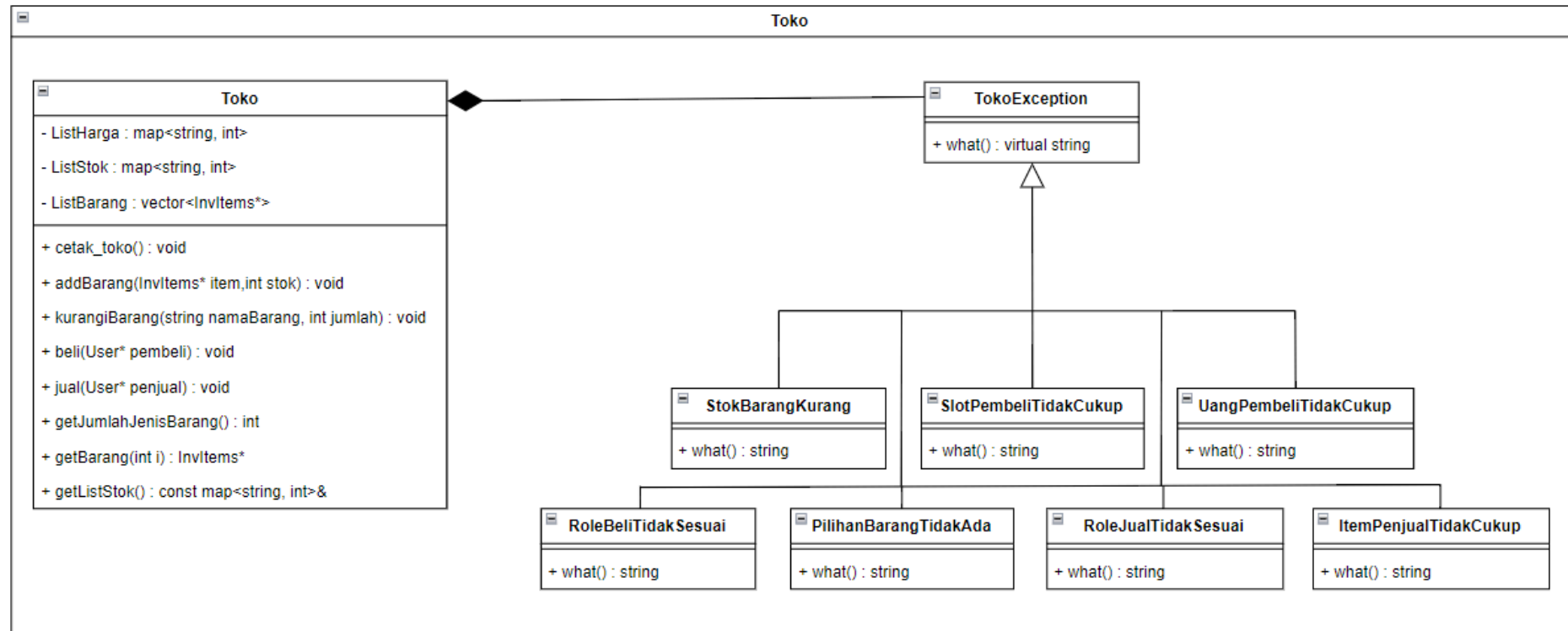
## 1.3 User



Gambar 1.3 Diagram Kelas User

Sumber : <https://app.diagrams.net/#G1cGHscqRWyWAOjddPk2gOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRVzhbsY4tBxP0kiT6n%22%7D> page User

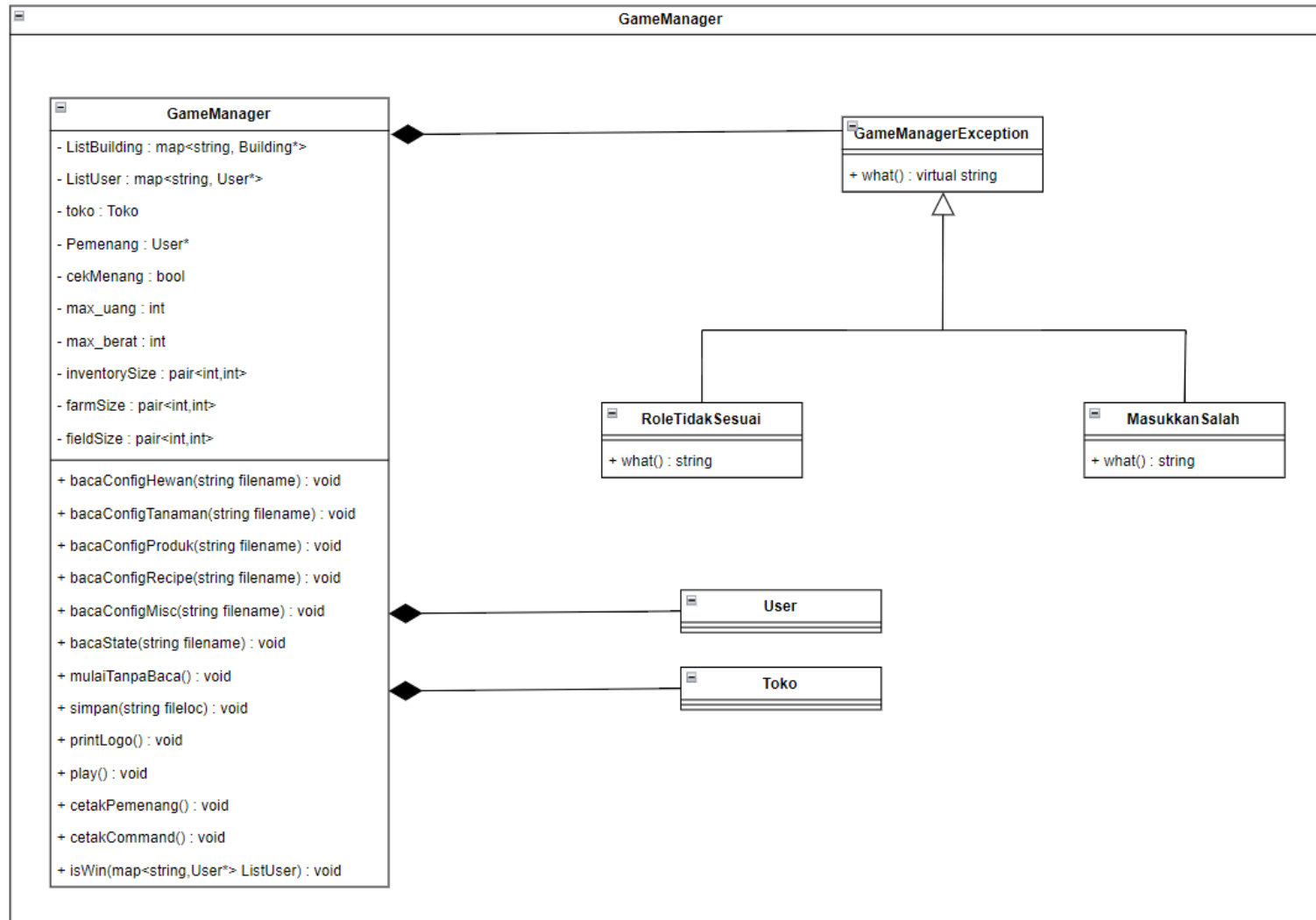
## 1.4 Toko



Gambar 1.4 Diagram Kelas Toko

Sumber : <https://app.diagrams.net/#G1eGHscqRWyWAOjddPk2eOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRYzhbsY4tBxP0kiT6n%22%7D> page All

## 1.5 GameManager



Gambar 1.5 Diagram Kelas GameManager

Sumber : <https://app.diagrams.net/#G1eGHscqRWvWAOjddPk2gOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRYzhbsY4tBxP0kiT6n%22%7D> page All

Desain kelas yang kami pilih telah dibuat sedemikian rupa dengan tujuan memastikan bahwa setiap kelas memiliki tanggung jawab yang jelas sesuai dengan perannya masing-masing dalam sistem. Kami memulai dengan kelas Inventory sebagai kelas generik yang bertanggung jawab atas penyimpanan yang dimiliki oleh masing-masing User. Kelas InvItems menjadi *parent* dari Hewan, Tanaman, Product, dan Building, menunjukkan hubungan *inheritance* yang memungkinkan untuk mengelompokkan jenis-jenis item yang berbeda dengan *behaviour* yang berbeda pula. Sementara itu, kelas User menjadi *parent* dari peran-peran dalam permainan, yakni walikota, peternak, dan petani, menunjukkan fleksibilitas dalam manajemen peran-peran yang terlibat. Toko bertanggung jawab sebagai tempat untuk bertransaksi, sementara GameManager mengatur alur jalannya permainan.

Dalam menangani exception, kami memutuskan untuk memisahkan penanganannya menjadi kelas-kelas *exception* terpisah seperti UserException, InvItemsException, InventoryException, GameManagerException, dan TokoException. Hal ini dilakukan untuk memastikan tanggung jawab setiap kelas *exception* dalam menangani kesalahan pada masing-masing kelas agar lebih terfokus dan terstruktur. Pemilihan desain untuk membuat tiap jenis-jenis hewan, tanaman, dan produk menjadi *child* dari kelas Hewan, Tanaman, dan Product secara berurutan adalah untuk mengakomodasi perbedaan-perbedaan *behaviour* yang dimiliki oleh masing-masing jenis yang berbeda. Penggunaan konsep agregasi dan komposisi juga digunakan untuk menghubungkan elemen-elemen dalam desain secara lebih terstruktur.

Kelebihan dari desain ini adalah setiap kelas telah disesuaikan dengan tanggung jawabnya masing-masing yang memungkinkan keterfokusan dalam melaksanakan perannya masing-masing. Namun, kekurangannya terletak pada implementasinya yang memerlukan lebih banyak waktu dan usaha karena kompleksitas desainnya.

Dalam proses pemilihan desain, kami menghadapi beberapa kendala seperti banyaknya perubahan dan diskusi yang harus dilakukan untuk menyatukan pandangan dari semua anggota tim. Selain itu, memastikan tidak adanya konflik tanggung jawab dari tiap-tiap kelas juga menjadi tantangan tersendiri dalam pembuatan desain ini. Dengan memperhatikan semua aspek ini, kami yakin desain kelas yang dipilih dapat menjadi landasan yang kokoh untuk pengembangan program ini.

## 2. Penerapan Konsep OOP

### 2.1. Inheritance & Polymorphism

```
1  class Material: public Product{
2      public:
3          Material();
4          Material(int id, string kode_huruf, string nama, string origin, int ad
ded_weight, int price);
5          void print();
6  };
7
8  //Type : PRODUCT_FRUIT_PLANT
9  class Fruit: public Product{
10     public:
11         Fruit();
12         Fruit(int id, string kode_huruf, string nama, string origin, int added
_weight, int price);
13         void print();
14 };
15
16 //Type : PRODUCT_ANIMAL
17 class Meat: public Product{
18     public:
19         Meat();
20         Meat(int id, string kode_huruf, string nama, string origin, int added_
weight, int price);
21         void print();
22 };
```

Salah satu contoh polymorphism dan inheritance adalah pada penyimpanan item pada inventory (selanjutnya disebut InvItem). InvItem merupakan parent dari Tanaman, Hewan, Product. Contoh *sceenshoot* kode di atas adalah menampilkan bagaimana file header yang menjelaskan turunan dari kelas Product, yakni Fruit, Meat, dan Material.

Polymorphism dan inheritance kami gunakan untuk menghindari redundansi dan menerapkan DRY (Don't Repeat Yourself) dikarenakan ada kelas - kelas yang memiliki fungsi yang mirip sehingga hanya perlu menuliskan mayoritas kode pada parent dan melakukan sedikit ubahan pada kelas child.

## 2.2. Method/Operator Overloading

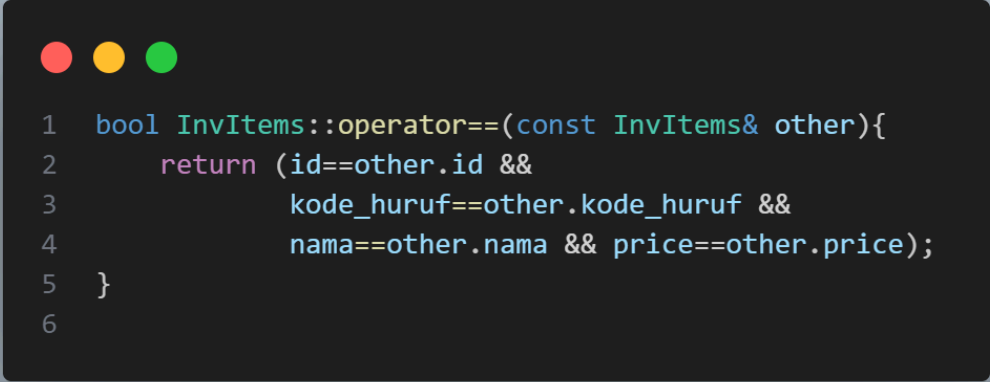
### 1. Operator+

```
1  template<class T>
2  Inventory<T>& Inventory<T>::operator+(const T& item) {
3      for(int i=0; i<rows; i++){
4          for(int j=0; j<cols; j++){
5              if(contents[i][j]==nullptr){
6                  contents[i][j]=item;
7                  neff++;
8                  return *this;
9              }
10         }
11     }
12     throw PenyimpananPenuh();
13 }
```



Operator+ digunakan dalam kelas inventory untuk menambahkan elemen pada satu slot inventory kosong paling mendekat awal. Fungsi ini tidak mengembalikan suatu instance inventory baru melainkan memodifikasi inventory awal. Operator+ dipilih sebagai cara mudah untuk menambahkan item kedalam inventory

## 2. Operator==



```
1  bool InvItems::operator==(const InvItems& other){
2      return (id==other.id &&
3              kode_huruf==other.kode_huruf &&
4              nama==other.nama && price==other.price);
5  }
6
```

Operator== digunakan dalam kelas InvItems untuk membandingkan dua invitems dan mengembalikan true jika semua atribut invitems tersebut sama. Operator== memudahkan perbandingan dua objek invitems tanpa harus mengakses semua atribut.

## 3. Operator()

```
template<class T>
T& Inventory<T>::operator()(int i, int j) {
    if(i<0 || j<0 || i>rows-1 || j>cols-1){
        throw BarisKolomTidakSesuai();
    }
    return contents[i][j];
}
```

Operator() digunakan dalam kelas Inventory untuk mengakses suatu 'petak' tertentu. Dilakukan juga pengecekan terhadap parameter operator sehingga hanya bisa mengembalikan isi Inventory pada indeks yang valid. Operator() Memudahkan akses (set and get) Inventory jika diminta petak yang ingin diakses.

### 2.3. Template & Generic Classes

```

1  template<class T>
2  class Inventory {
3  private:
4      vector<vector<T>> contents;
5      int rows, cols;
6      int neff;
7
8  public:
9      Inventory(int rows, int cols);
10
11     ~Inventory();
12
13     void addItem(T item, int row, int col);
14
15     void deleteItem(T item);
16

```

```


17     template<typename U>
18     U* getItem(int row, int col);
19
20     int getRows() const;
21
22     int getCols() const;
23
24     int getNeff() const;
25
26     void incNeff();
27
28     void decNeff();
29
30     bool isEmpty();
31
32     bool isFull();
33
34     T& operator()(int i, int j);
35
36     Inventory<T>& operator+(const T& item);
37 };

```

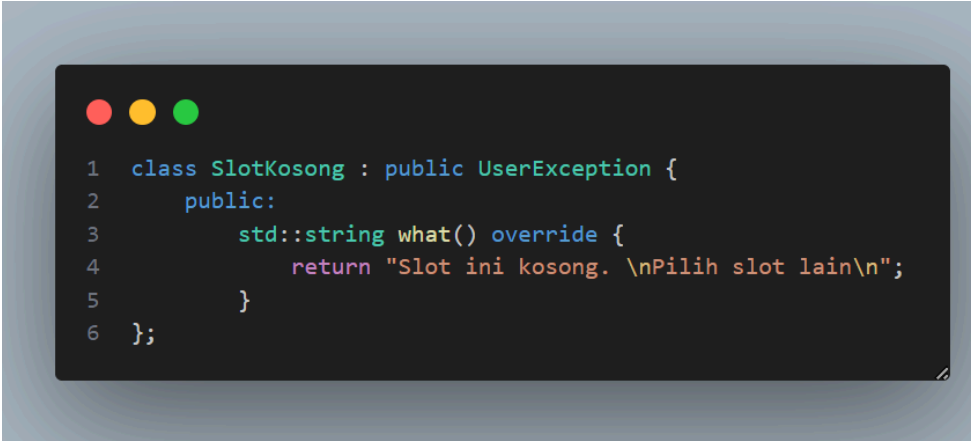
Kami menggunakan konsep template dan generic classes untuk mengimplementasikan inventory, kelas inventory adalah kelas yang memiliki atribut yaitu vector dari vector dari suatu kelas T. Dengan menggunakan template, kelas ini bisa menyimpan berbagai jenis kelas. Dalam pengimplementasiannya kami menggunakan kelas ini untuk membuat objek penyimpanan, ladang, dan juga peternakan. Penyimpanan adalah Instansiasi Inventory dengan tipe elemen InvItems\*, ladang adalah instansiasi inventory dengan tipe elemen Tanaman\*, dan peternakan adalah instansiasi Inventory dengan tipe elemen Hewan\*.

## 2.4. Exception & Exception Handling

### 2.4.1. Validasi masukan slot penyimpanan



```
1 class HarapanKosong : public UserException {  
2     public:  
3         std::string what() override {  
4             return "Kamu mengambil harapan kosong dari penyimpanan";  
5         }  
6 };  
7
```



```
1 class SlotKosong : public UserException {  
2     public:  
3         std::string what() override {  
4             return "Slot ini kosong. \nPilih slot lain\n";  
5         }  
6 };
```

```
1 class LadangFull : public UserException {
2     public:
3         std::string what() override {
4             return "Ladang Sudah Penuh. \n";
5         }
6 };
7
8 class LadangEmpty : public UserException {
9     public:
10        std::string what() override {
11            return "Ladang Masih Kosong. \n";
12        }
13 };
14
15 class PeternakanFull : public UserException {
16     public:
17        std::string what() override {
18            return "Peternakan Sudah Penuh. \n";
19        }
20 };
21
22 class PenyimpananKosong : public UserException {
23     public:
24        std::string what() override {
25            return "Penyimpanan saat ini Kosong. \n";
26        }
27 };
```

### 2.4.2. Validasi Kesiapan Panen

```
1 class TanamanBelumSiapPanen : public UserException {
2     public:
3         std::string what() override {
4             return "Belum ada tanaman yang siap untuk dipanen! \n";
5         }
6 };
7 class HewanBelumSiapPanen : public UserException {
8     public:
9         std::string what() override {
10            return "Belum ada hewan yang siap untuk dipanen! \n";
11        }
12 };
```

### 2.4.3. Validasi masukan

```
1 class BarisKolomTidakSesuai : public InventoryException {
2     public :
3         std::string what() override{
4             return "Input baris dan kolom tidak sesuai dengan ukuran matriks penyimpanan\n";
5         }
6 };
```

```
1 class HerbivoreMakanDaging : public InvItemsException {
2     public:
3     string what() {
4         return "Herbivore tidak bisa makan daging, berikan makanan lain!";
5     }
6 };
7 class CarnivoreMakanBuah : public InvItemsException {
8     public:
9     string what() {
10        return "Carnivore tidak bisa makan buah, berikan makanan lain!";
11    }
12 };
13 class ItemBukanMakanan : public InvItemsException {
14     public:
15     string what() {
16        return "Item ini bukan Makanan!";
17    }
18 };
19 class MakananTidakCocokException : public InvItemsException {
20     public:
21     string what() {
22        return "Makanan tidak cocok, berikan jenis makanan lain!";
23    }
24 };
25
```



```
1 class GameManagerException{
2     public :
3         virtual string what()=0; //pure virtual
4 };
5
6 class RoleTidakSesuai : public GameManagerException {
7     public :
8         string what() override{
9             return "Tidak dapat menjalankan command dengan role ini\n";
10        }
11 };
12
13 class MasukkanSalah : public GameManagerException {
14     public :
15         string what() override{
16             return "Masukkan tidak sesuai periksa lagi!\n";
17        }
18 };
```

#### 2.4.4. Penanganan Exception tidak terduga

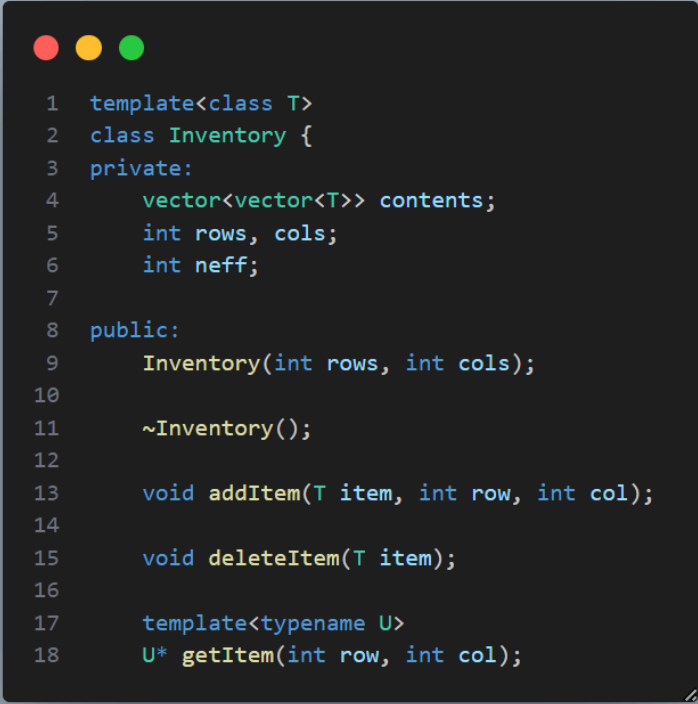
```
1 } catch (exception& e) {
2     cout << "Kesalahan di luar ketentuan: " << e.what() << endl;
3 }
```

Keuntungan menggunakan exception adalah:

1. Mempermudah proses debugging  
Pesan kesalahan yang diprediksi akan terjadi dapat tampil secara spesifik.
2. Mempermudah pengguna program mengetahui kesalahan yang terjadi dan mempelajari agar tidak mengulang kesalahan yang sama
3. Menghindari program berhenti total ketika pengguna memberi masukan yang salah
4. Membuat kode mudah dipelajari oleh developer lain

## **2.5. C++ Standard Template Library**

## 1. Vector

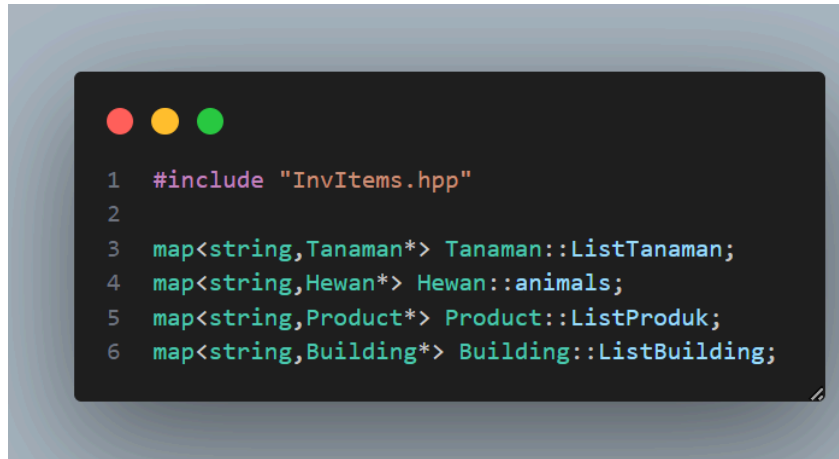


```
1  template<class T>
2  class Inventory {
3  private:
4      vector<vector<T>> contents;
5      int rows, cols;
6      int neff;
7
8  public:
9      Inventory(int rows, int cols);
10
11     ~Inventory();
12
13     void addItem(T item, int row, int col);
14
15     void deleteItem(T item);
16
17     template<typename U>
18     U* getItem(int row, int col);
```

Justifikasi dan keuntungan:

Vector mempermudah menyimpan elemen visualisasi seperti matriks (lihat -> bagian penyimpanan)

## 2. Map

A code editor window with a dark background and light-colored text. It contains six lines of C++ code. Line 1: #include "InvItems.hpp". Line 2: (empty). Line 3: map<string,Tanaman\*> Tanaman::ListTanaman;. Line 4: map<string,Hewan\*> Hewan::animals;. Line 5: map<string,Product\*> Product::ListProduk;. Line 6: map<string,Building\*> Building::ListBuilding;. The code is color-coded: #include is orange, string is green, Tanaman\* is blue, Hewan\* is blue, Product\* is blue, and Building\* is blue. The ::ListTanaman, ::animals, ::ListProduk, and ::ListBuilding are in green. The code is enclosed in a light gray border with three colored circles (red, yellow, green) in the top left corner.

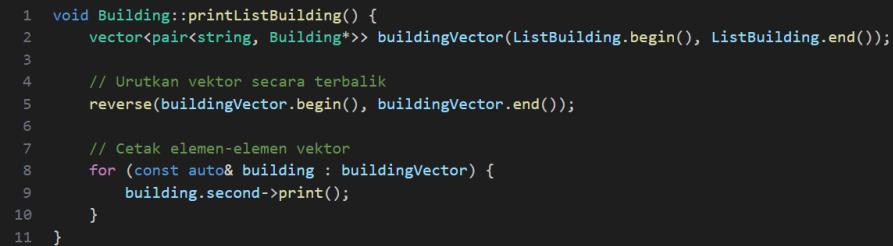
```
1  #include "InvItems.hpp"
2
3  map<string,Tanaman*> Tanaman::ListTanaman;
4  map<string,Hewan*> Hewan::animals;
5  map<string,Product*> Product::ListProduk;
6  map<string,Building*> Building::ListBuilding;
```

Justifikasi dan keuntungan:

Map mempermudah untuk mengakses suatu nilai dari sebuah kunci (Key akses Value), misal penggunaan listTanaman, bila ingin menyimpan alamat dan kemudian suatu saat mengakses alamat dari suatu tanaman berdasarkan string kode, tentu saja map adalah solusi yang tepat.

Map juga digunakan untuk listUser karena Urutan giliran permainan ditentukan berdasarkan urutan leksikografis dari username pemain. Sementara key value dalam map akan otomatis terurut secara leksikografis. Oleh karena itu kami memutuskan untuk menggunakan map dengan key username sebagai list user.

## 3. Algorithm



```
1 void Building::printListBuilding() {
2     vector<pair<string, Building*>> buildingVector(ListBuilding.begin(), ListBuilding.end());
3
4     // Urutkan vektor secara terbalik
5     reverse(buildingVector.begin(), buildingVector.end());
6
7     // Cetak elemen-elemen vektor
8     for (const auto& building : buildingVector) {
9         building.second->print();
10    }
11 }
```

Justifikasi dan keuntungan:

Algorithm mempermudah pengeluaran output yang sesuai dengan yang diinstruksikan seperti penggunaan reverse pada printListBuilding().

## 2.6. Konsep OOP lain

### 1. Abstract Base Class

Abstract Base Class (ABC) adalah kelas dalam pemrograman berorientasi objek yang dirancang untuk digunakan sebagai kerangka dasar untuk kelas-kelas turunannya. Hal ini ditandai dengan: Mengandung Metode Virtual Murni (Metode virtual murni ditandai dengan menggunakan sintaks virtual diikuti oleh = 0), Tidak Dapat Diinstansiasi, dan digunakan untuk Polimorfisme.

```
1  class InvItems
2  {
3  protected:
4      int id;
5      string kode_huruf;
6      string nama;
7      int price;
8
9  public:
10     InvItems();
11     InvItems(int id, string kode_huruf, string nama, int price);
12     string getKode();
13     string getNama();
14     int getPriceItem();
15     int getId();
16
17     // Operator overloading (akan memprint nama item)
18     friend ostream &operator<<(ostream &os, InvItems &b);
19     // memprint semua atribut objek
20     virtual void print();
21     virtual bool isProduct() const = 0;
22 };
23
```

```
1  class InvItems{
2      protected:
3          int id;
4          string kode_huruf;
5          string nama;
6          int price;
7      public:
8          InvItems();
9          InvItems(int id, string kode_huruf, string nama, int price);
10         string getKode();
11         string getNama();
12         int getPriceItem();
13         int getId();
14         bool operator==(const InvItems& other);
15
16         //Operator overloading (akan memprint nama item)
17         friend ostream& operator<<(ostream& os, InvItems& b);
18         //memprint semua atribut objek
19         virtual void print();
20         virtual bool isProduct() const = 0;
21         virtual bool isMakanan() = 0;
22         virtual int getAddedWeight() = 0;
23     };
```

## 2. Composition

Composition adalah konsep OOP yang menjelaskan suatu kelas itu terdiri atas beberapa komponen, yang tiap komponen tersebut tidak mempengaruhi komponen lainnya misalnya sebuah mobil yang terdiri dari sasis dan roda. Bila roda pecah maka sasisnya akan tetap baik-baik saja. Konsep ini baik untuk program TuBes ini karena kode yang dikerjakan sangat banyak, akan sangat konyol kalau satu komponen salah maka keseluruhan program menjadi rusak.

```
1  class User{
2      protected:
3          // static int n_player;
4          int id;
5          std::string username;
6          int berat_badan;
7          int uang;
8          Inventory<InvItems*> penyimpanan;
9      public:
10         User();
11         User(std::string username, pair<int,int> invSize);
12         User(std::string username, int berat,int uang, pair<int,int> invSize);
13         virtual ~User();
14
```

Salah satu konsep *composition* adalah antara InventoryItems (disingkat InvItems) dengan Inventory, yang mana kelas InvItems merupakan komponen dari Inventory. Misal ada seseorang yang merusak logika InvItems maka secara praktis kelas Inventory tidak akan menjadi rusak.



```
1 void User::makan() {
2     cetak_penyimpanan();
3     std::string slot;
4     std::string subslot;
5     cout << "\nSlot: ";
6     cin >> slot;
7     subslot = slot.substr(1, 2);
8     int i = stoi(subslot) - 1;
9     int j = slot[0] - 'A';
10
11
12     try {
13         // Melakukan penanganan exception
14         if (i >= penyimpanan.getRows() || j >= penyimpanan.getCols() || i < 0 || j < 0) {
15             throw BarisKolomTidakSesuai();
16         }
17
18         if (penyimpanan(i, j) == nullptr) {
19             throw HarapanKosong();
20         }
21     }
```


```
1  InvItems* item = penyimpanan(i, j);
2  string tipeIdItem = typeid(*item).name();
3  string tipeItem = tipeIdItem.substr(1);
4  cout << "Anda memilih " << tipeItem << " untuk diberikan kepada " << tipe << endl;
5
6  if(tipe == "Carnivore" && tipeItem != "Meat"){
7      throw MakananTidakCocokException();
8  }
9  if(tipe == "Herbivore" && tipeItem != "Fruit"){
10     throw MakananTidakCocokException();
11 }
12 if(tipe == "Omnivore" && !(tipeItem == "Meat" || tipeItem == "Fruit")){
13     throw MakananTidakCocokException();
14 }
```

Demikian juga antara kelas User Exception dan Inventory Exception dengan kelas yang menggunakannya, yakni user dan inventory. Exception merupakan komponen dari kelas User dan Inventory.

### 3. Aggregation

Aggregation adalah konsep di mana sebuah objek yang kompleks terdiri dari objek-objek yang lebih sederhana atau komponen-komponen. Dalam agregasi, objek-objek tersebut tetap independen satu sama lain, yang berarti jika objek induknya dihapus,

objek-objek komponennya tetap ada. Ditandai bagian pointer menuju kelas User yang berada di kelas GameManager

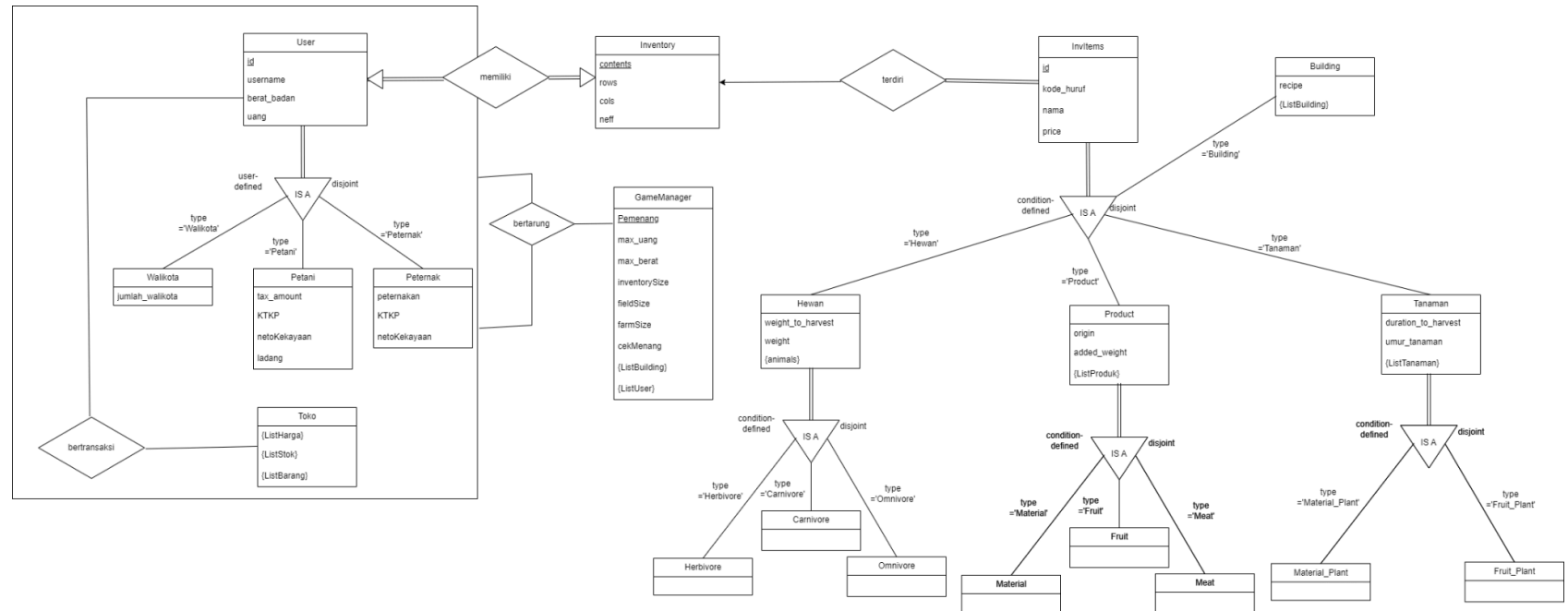


```
1 class GameManager{
2     private:
3         map<string,User*> ListUser;
4         Toko toko;
5         User *Pemenang;
```

### 3. Bonus Yang dikerjakan

#### 3.1. Bonus yang diusulkan oleh spek

##### 3.1.1. Diagram Sistem



Gambar 3.1.1. Diagram Sistem Model E-R

Sumber : <https://app.diagrams.net/#G1eGHscqRWyWAOjddPk2gOM4Gb6dK-cvGB#%7B%22pageId%22%3A%22aCRYzhbsY4tBxP0kiT6n%22%7D> page Diagram Sistem

## 4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Kelas Utama -> InvItems	13522123 Jimly 13522151 Samy	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad

		13522152 Muhammad Roihan
Kelas Utama -> Inventory	13522123 Jimly 13522127 Maulana MS 13522151 Samy	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Next	13522152 Muhammad Roihan	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Cetak Penyimpanan	13522127 Maulana MS	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Pungut Pajak	13522123 Jimly 13522152 Muhammad Roihan	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Cetak ladang	13522123 Jimly 13522127 Maulana MS	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Cetak Peternakan	13522127 Maulana MS	13522123 Jimly Nur Arif

		13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Tanam	13522127 Maulana MS	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Ternak	13522127 Maulana MS	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Bangun bangunan	13522134 Shabrina Maharani	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Makan	13522134 Shabrina Maharani	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Memberi Pangan	13522123 Jimly Nur Arif	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan

Perintah -> Membeli	13522152 Muhammad Roihan	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah->Menjual	13522152 Muhammad Roihan	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Panen	13522151 Samy Muhammad Haikal	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Muat	13522151 Samy Muhammad Haikal	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Simpan	13522151 Samy Muhammad Haikal	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan
Perintah -> Tambah Pemain	13522127 Maulana MS 13522152 Muhammad Roihan	13522123 Jimly Nur Arif 13522127 Maulana MS 13522134 Shabrina Maharani 13522151 Samy Muhammad 13522152 Muhammad Roihan

**Lampiran Form Asistensi Tugas Besar  
IF2210/Pemrograman Berorientasi Objek  
Sem. 1 2023/2024**



No. Kelompok/Kode Kelompok : B88  
 Nama Kelompok : Bahari88  
 Anggota Kelompok (Nama/NIM) :  
     1. Jimly Nur Arif/13522123  
     2. Maulana Muhamad Susetyo/13522127  
     3. Shabrina Maharani/13522134  
     4. Samy Muhammad Haikal/13522151  
     5. Muhammad Roihan/13522152

Asisten Pembimbing : Primanda Adyatma Hafiz / 13520022

<b>Tanggal : 05-04-2024</b>	<b>Catatan Asistensi:</b>
<b>Tempat : Daring</b>	<ol style="list-style-type: none"> <li>1. Typename sama class dicoba aja tapi kalau bisa pakainya class</li> <li>2. Hewan itu ada beberapa macam, kalau dimasukin ke type jadi ga single, jadi nyimpennya terlalu banyak. Jadi, kalau bisa buat abstraksi untuk hewan. (Berlaku pula untuk tumbuhan). Jadi harus dibuat kelasnya masing-masing berdasarkan typenya. Intinya kalau behavior nya beda dibuat kelas yang berbeda.</li> <li>3. Kalo config itu pasti di-load. Jadi, pasti langsung baca config. Sedangkan, kalau di spesifikasi itu harus bisa milih. Yang penting dia harus bisa di-load di awal.</li> <li>4. Run dan compile di makefile. Harus bisa dijalanin lewat makefile.</li> <li>5. Class nya lebih di-detil-in lagi supaya single responsibility.</li> </ol>

**Kehadiran Anggota Kelompok:**

1



2






3



(13522151)

4

 5 13522152 	
	<b>Tanda Tangan Asisten:</b> 

**Repositori Github :** [https://github.com/mroi hn/Tubes\\_OOP](https://github.com/mroi hn/Tubes_OOP)