

Kode Kelompok : SMH

Nama Kelompok : OOop

1. 13522053 / Erdianti Wiga Putri Andini
2. 13522055 / Benardo
3. 13522113 / William Glory Henderson
4. 13522115 / Derwin Rustanly
5. 13522117 / Mesach Harmasendro
6. 10023575 / Renny Melanda Febriyanti

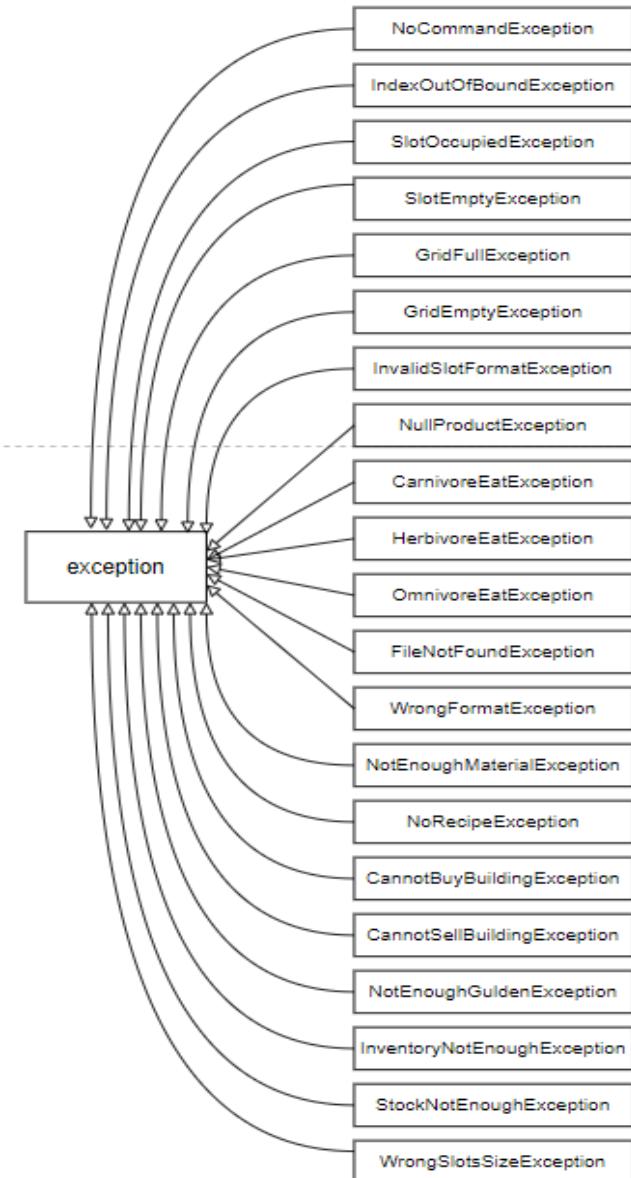
Asisten Pembimbing : Vincent Prasetya Atmadja / 13520099

## 1. Diagram Kelas

No	<i>Class Diagram</i>	Keterangan
1.	<pre> classDiagram     class Player     class Mayor     class Resident     class Farmer     class Breeder      Player &lt; -- Mayor     Player &lt; -- Resident     Resident &lt; -- Farmer     Resident &lt; -- Breeder   </pre>	Kelas Player menjadi kelas yang menjadi dasar pembentukan player dalam permainan. Player dapat diturunkan menjadi kelas Mayor dan Resident. Kemudian, Resident dapat diturunkan lagi menjadi Farmer dan Breeder.
2.	<pre> classDiagram     class Item     class Building     class Plant     class Animal     class Product     class Herbivore     class Carnivore     class Omnivore      Item &lt; -- Building     Item &lt; -- Plant     Item &lt; -- Animal     Item &lt; -- Product     Animal &lt; -- Herbivore     Animal &lt; -- Carnivore     Animal &lt; -- Omnivore   </pre>	Kelas Item adalah base class untuk semua benda yang ada di permainan ini. Kelas Item mempunyai 4 anak yaitu building, plant, animal, dan product. Untuk class Animal juga mempunyai 3 anak yaitu Herbivore, Carnivore, dan Omnivore.

3.	<pre>classDiagram     class Grid     class Inventory     class SpecializedGrid     class Ranch     class Farm      Grid &lt; -- Inventory     Grid &lt; -- SpecializedGrid     SpecializedGrid &lt; -- Ranch     SpecializedGrid &lt; -- Farm</pre> The diagram illustrates a class hierarchy. At the top level is the 'Grid' class, which is a template class. It has two subclasses: 'Inventory' and 'SpecializedGrid'. The 'SpecializedGrid' class is also a template class and has two subclasses: 'Ranch' and 'Farm'. Arrows point from the subclass names to their respective parent classes, indicating inheritance.	<p>Grid adalah sebuah template class yang menjadi superclass dari semua struktur yang digunakan untuk menampung objek tertentu. Grid mempunyai dua subclass yaitu Inventory dan SpecializedGrid. SpecializedGrid juga adalah template class yang menjadi super class untuk Ranch dan Farm</p>
----	---	---

4.



Semua exception yang dibuat pada tugas besar ini merupakan anak dari class exception bawaan C++. Hal tersebut dilakukan untuk mempermudah dalam melakukan catch dengan Polimorfisme.

Detail penjelasan tentang exception yang dibuat bisa dilihat pada bab **2.4 Exception**

5.	<pre>classDiagram     class Game     class Store     class LoadHandler     class Command     class Recipe     class AnimalConfig     class ProductConfig     class PlantConfig     class MainConfig</pre>	Di samping ada beberapa class yang juga digunakan di dalam program ini. Class yang berakhiran config dan class recipe adalah class yang akan menyimpan data dari file config yang dibaca pertama kali saat memulai game. Game adalah class utama yang mengontrol alur dari permainan. Class store adalah class yang merepresentasikan toko tempat para pemain melakukan beli dan jual dalam permainan ini. Class command adalah class yang mengurus hal-hal yang berkaitan dengan perintah yang diberikan pemain seperti pengecekan dan lain-lain. Class load handler adalah yang mengurus segala hal yang berkaitan dengan load baik load config maupun load state.
----	---	--

Untuk detail lebih lengkap dari diagram kelas bisa dilihat pada link yang akan diberikan di bawah ini. Pada link berikut diagram kelas juga sudah mengandung semua member yang ada pada suatu kelas beserta hubungan untuk semua kelas. Member untuk kelas tersebut tidak termasuk untuk constructor dan destructor.

Link diagram kelas dan uml: <https://drive.google.com/file/d/1-Dw1L2AnhFGMQmVtcUiJfJYb5VQ37hok/view?usp=sharing>

Ini adalah link draw.io Diagram class berada pada page UML OOP

## 2. Penerapan Konsep OOP

### 2.1. Inheritance dan Polymorphism

Inheritance adalah salah satu konsep dalam OOP yang menunjukkan kemampuan kelas untuk menurunkan atribut dan method dari kelas lain. Terdapat dua jenis kelas dalam inheritance yaitu *superclass* dan *subclass*. *Superclass* atau *base class* adalah kelas “induk” yang atribut dan *method*-nya diturunkan ke kelas lain. Sedangkan *subclass* atau *derived class* adalah kelas “anak” yang menurunkan atribut & *method* dari kelas lain. Dalam inheritance, *subclass* dapat memiliki metode dan atribut yang sama dengan *superclass*, tetapi juga dapat menambahkan metode dan atribut unik yang hanya dimilikinya sendiri. Dengan cara ini, inheritance membantu untuk mengurangi redudansi kode dan meningkatkan efisiensi karena *subclass* dapat memanfaatkan implementasi yang sudah ada dari *superclass*.

Polymorphism adalah konsep dalam OOP di mana objek dari kelas yang berbeda dapat berperilaku secara berbeda tergantung pada konteks pemanggilan metodenya. Polymorphism memungkinkan untuk menulis kode yang lebih fleksibel dan mudah dikelola karena dapat menggunakan kelas dasar untuk mengakses objek dari kelas turunan tanpa harus mengetahui jenis kelas turunan secara spesifik. Terdapat dua jenis polymorphism, yaitu:

1. **Polymorphism compile-time** (*Static Polymorphism* atau *Early Binding*) yang terjadi saat keputusan untuk memanggil metode ditentukan saat kompilasi. Contohnya adalah overload fungsi dan operator.
2. **Polymorphism runtime** (*Dynamic Polymorphism* atau *Late Binding*) yang terjadi saat keputusan untuk memanggil metode ditentukan saat runtime. Jenis ini biasanya dicapai melalui penggunaan fungsi virtual dan overriding dalam inheritance. Contoh yang paling umum adalah ketika membuat pointer ke *superclass* dan menggunakannya untuk memanggil metode yang diimplementasikan dalam *subclass*.

Secara keseluruhan penggunaan konsep Polymorphism dan Inheritance disini digunakan untuk membuat kode program yang lebih efisien dan mengurangi jumlah kasus kode yang berulang. Konsep inheritance disini juga dibuat untuk memecah tanggung jawab tertentu kepada class-class khusus sehingga program yang dibuat memang benar-benar menerapkan prinsip Single Responsibility.

Polimorfisme di C++ baru akan bisa berjalan dengan baik dengan memanfaatkan pointer. Jika ingin melakukan polimorfisme tetapi tidak menggunakan pointer maka akan ada kecenderungan terjadi objek slicing dimana suatu kelas anak akan dihilangkan member-membernya sehingga kelas anak tersebut benar-benar berubah menjadi kelas induk. Penggunaan pointer secara umum mengharuskan melakukan penghapusan atau pembersihan memory secara manual agar tidak terjadi memory leak. Dalam tubes ini agar kami tidak perlu melakukan pembersihan memori secara manual dan agar program yang kami buat lebih aman dari memori leak atau error-error sejenisnya kami memanfaatkan ***shared\_ptr*** yang telah disediakan oleh C++ dengan meng-include memory. Dengan menggunakan ***shared\_ptr***, kami menjadi tidak perlu lagi melakukan pembersihan memori secara manual karena itu semua akan dikontrol oleh C++ nya secara langsung. Penggunaan ***shared\_ptr*** juga mengharuskan kami untuk menggunakan ***make\_shared*** pada

saat ingin membuat objek. Secara sederhana ***shared\_ptr*** bekerja dengan cara menghitung berapa banyak pointer yang merujuk kepada suatu memory ketika sudah tidak ada lagi pointer yang merujuk ke memori tersebut maka secara otomatis memori tersebut akan dibersihkan. Dalam tubes ini penggunaan polimorfisme akan cukup banyak dilakukan yang berarti penggunaan pointer juga akan menjadi cukup banyak oleh sebab itu penggunaan ***shared\_ptr*** ini akan sangat membantu kami dalam membuat kode program yang lebih bagus lagi.

### **2.1.1. Penggunaan Inheritance dan Polymorphism Pada Player**

Super Class	Sub Class
<pre>#ifndef _PLAYER_HPP_ #define _PLAYER_HPP_  #include &lt;iostream&gt; #include "../Inventory/Inventory.hpp" #include "../Store/Store.hpp" using namespace std;  class Player { protected:     static int countIdPlayer;     int playerId;     string type;     string username;     int weight;     int gulden;     Inventory inventory;  public:     Player();     Player(string username, int weight, int gulden);     Inventory &amp;getInventory();     string &amp;getName();     virtual ~Player();     int &amp;getWeight();     virtual void buy(shared_ptr&lt;Item&gt; &amp;, int);     virtual pair&lt;vector&lt;shared_ptr&lt;Item&gt;&gt;, int&gt; sell(vector&lt;string&gt; &amp;);     int &amp;getGulden();     void setGulden(int);     bool operator==(const Player&amp;);      void eat(); };  #endif</pre>	<pre>#ifndef _MAYOR_HPP_ #define _MAYOR_HPP_  #include "../Player/Player.hpp" #include "../Item/Building.hpp" #include "../configClass/RecipeConfig.hpp" #include "../Utils/Utils.hpp" #include &lt;vector&gt; #include &lt;map&gt;  class Mayor : public Player {  public:     Mayor();     Mayor(string username, float weight, int gulden);     ~Mayor();     void taxCollection();     void buildBuilding();     void addPlayer();     pair&lt;vector&lt;shared_ptr&lt;Item&gt;&gt;, int&gt; sell(vector&lt;string&gt; &amp;);     void buy(shared_ptr&lt;Item&gt; &amp;, int);     void saveFile(const string &amp;filename); };  #endif</pre>

```
#ifndef _RESIDENT_HPP_
#define _RESIDENT_HPP_

#include "../Player/Player.hpp"
#include "../Utils/Utils.hpp"
#include <functional>
#include <string>
#include <climits>
using namespace std;
You 4 minutes ago | author (You)
class Resident : public Player
{
public:
    Resident();
    Resident(string, int, int);
    ~Resident();
    virtual int calculateTax() = 0;
    virtual void harvest() = 0;
    virtual int getWealth() = 0;

protected:
    template <class T, class U>
    void harvest(T &source, map<string, U> &config, map<string, vector<string>> harvestResult, const string &name)
    {

template <class T, class X, class U>
void placeTo(T &target, map<string, X> &config, function<bool()> checkInventory, const string &name)
{
```

Implementasi template function juga terdapat di file .hpp

## Keterangan

Super Class : Player

Sub Class : Mayor dan Resident

Penjelasan :

Penggunaan inheritance pada kelas player bertujuan untuk membagi player menjadi 2 subkelas, yakni Mayor (Walikota) dan Resident (penduduk). Mayor memiliki *member object* khusus yakni *method* taxCollection() dan buildBuilding() yang berturut-turut berfungsi untuk melakukan pemungutan pajak dari penduduk berdasarkan ketentuan yang telah ditetapkan, serta membangun bangunan dari bahan yang tersedia pada penyimpanan. Sementara itu, Resident merupakan subkelas abstrak yang memiliki *method* khusus yakni calculateTax() yang berfungsi untuk menghitung pajak yang harus dibayarkan kepada walikota berdasarkan ketentuan KKP, serta getWealth() yang berfungsi untuk menghitung neto kekayaan penduduk berdasarkan gulden dan barang yang tersimpan di penyimpanan maupun di ladang/peternakan, dan harvest() yang berfungsi untuk memanen hasil ternak/tani. Konsep inheritance cocok untuk digunakan disini karena para pemain mempunyai beberapa atribut dan fungsi yang sama yang bisa dikelompokkan di superclass sedangkan fungsi dan atribut spesifik untuk role tertentu bisa dibuat di subclassnya. Struktur dari class Player dan turunannya ini dibuat sedemikian rupa agar bisa mendukung konsep Polymorphism sehingga pemain dengan role yang berbeda bisa ditampung pada STL vector yang sama untuk mempermudah penyimpanan data pemain yang ada dalam game.

Super Class	Sub Class
<pre>#ifndef _RESIDENT_HPP_ #define _RESIDENT_HPP_  #include "../Player/Player.hpp" #include "../Utils/Utils.hpp" #include &lt;functional&gt; #include &lt;string&gt; #include &lt;limits&gt; using namespace std; You 4 minutes ago   author (You) class Resident : public Player { public:     Resident();     Resident(string, int, int);     ~Resident();     virtual int calculateTax() = 0;     virtual void harvest() = 0;     virtual int getWealth() = 0;  protected:     template &lt;class T, class U&gt;     void harvest(T &amp;source, map&lt;string, U&gt; &amp;config, map&lt;string, vector&lt;string&gt;&gt; harvestResult, const string &amp;name)     {          template &lt;class T, class X, class U&gt;         void placeTo(T &amp;target, map&lt;string, X&gt; &amp;config, function&lt;bool()&gt; checkInventory, const string &amp;name)         { </pre>	<pre>#ifndef _BREEDER_HPP_ #define _BREEDER_HPP_  #include "../Ranch/Ranch.hpp" #include "../Item/Building.hpp" #include "../Resident/Resident.hpp" using namespace std;  #define KTP_PETERNAK 11  class Breeder: public Resident{ private:     Ranch ranch; public:     Breeder();     Breeder(string username, float weight, int gulden);     ~Breeder();     int getWealth();     void cattle();     void feedAnimal();     void harvest();     int calculateTax();     Ranch&amp; getRanch();     void saveFile(const string&amp; filename); };  #endif </pre>

```

#ifndef _FARMER_HPP_
#define _FARMER_HPP_

#include "../Farm/Farm.hpp"
#include "../Item/Building.hpp"
#include "../Resident/Resident.hpp"

#define KTKP_PETANI 13

class Farmer: public Resident{
private:
    Farm farm;
public:
    Farmer();
    Farmer(string username, int weight, int gulden);
    ~Farmer();
    int getWealth();
    Farm& getFarm();
    void plant();
    void harvest();
    int calculateTax();
    void saveFile(const string& filename);
};

#endif

```

### Keterangan

Super Class : Resident

Sub Class : Breeder (Peternak) dan Farmer (Petani)

Penjelasan :

Penggunaan inheritance pada kelas Resident bertujuan untuk membedakan penduduk sebagai breeder (peternak) yang memiliki atribut ranch (peternakan) serta farmer (petani) yang memiliki atribut farm (ladang). Breeder memiliki *method* cattle() yang berfungsi untuk menempatkan hewan yang terletak dalam penyimpanan menuju ke peternakan. Sementara itu farmer memiliki method plant() yang bertujuan untuk menempatkan tanaman yang terletak dalam penyimpanan menuju ke ladang. Keberadaan class Resident sendiri ada untuk mengelompokkan sifat-sifat yang sama yang dimiliki oleh Farmer dan Breeder yang membedakan mereka dengan Mayor(Walikota).

## 2.1.2. Penggunaan Inheritance dan Polymorphism Pada Item

Super Class	Sub Class
<pre>#ifndef __ITEM_HPP_ #define __ITEM_HPP_  #include &lt;iostream&gt; #include &lt;string&gt; #include "../configClass/PlantConfig.hpp" #include "../configClass/AnimalConfig.hpp" #include "../configClass/ProductConfig.hpp" #include "../configClass/RecipeConfig.hpp"  using namespace std;  class Item { protected:     static int countIdItem;     int itemId;     string name;     string code;     int price;  public:     Item();     Item(string name, string code, int price);     virtual ~Item();     void setItemId(int itemId);     void setName(string name);     void setCode(string code);     void setPrice(int price);     const int&amp; getItemId() const;     const string&amp; getName() const;     const string&amp; getCode() const;     const int&amp; getPrice() const;     bool operator==(const Item &amp;);     friend ostream &amp;operator&lt;&lt;(ostream &amp;, const Item &amp;);  };  #endif</pre>	<pre>#ifndef __ANIMAL_HPP_ #define __ANIMAL_HPP_  #include "AnimalException.hpp" #include "../Harvester/Harvester.hpp" using namespace std;  class Animal : public Item{ private:     int animalId;     int weight;     int weighToHarvest;     Harvester harvester;     static map&lt;string, vector&lt;string&gt;&gt; harvestResult;  public:     Animal();     Animal(int weight, string name);     virtual ~Animal();     void setAnimalType(AnimalType type);     void setAnimalId(int animalId);     void setWeight(int weight);     void setWeightToHarvest(int weightToHarvest);     int getAnimalId() const;     int getWeight() const;     int getWeightToHarvest() const;     vector&lt;shared_ptr&lt;Product&gt;&gt; collect();     bool checkReadyToHarvest();     static map&lt;string, vector&lt;string&gt;&gt;&amp; getHarvestResult();      virtual void eat(const shared_ptr&lt;Product&gt;&amp; food);  };  #endif</pre>

```
#ifndef __BUILDING_HPP_
#define __BUILDING_HPP_

#include "Item.hpp"
#include <map>
#include <string>

class Building : public Item{
private:
    map<string, int> material;
public:
    Building();
    Building(string name);
    ~Building();
    int getQuantityPerMaterial(string materialName) const;
    map<string, int> getBuildingMaterial() const;
};

#endif
```

```
#ifndef __PLANT_HPP_
#define __PLANT_HPP_

#include "Item.hpp"
#include "../Harvester/Harvester.hpp"

class Plant : public Item
{
private:
    PlantType type;
    int plantId;
    int durationToHarvest;
    int age;
    Harvester harvester;

    static map<string, vector<string>> harvestResult;

public:
    Plant();
    Plant(int age, string name);
    ~Plant();
    Plant(string name);
    void setPlantType(PlantType type);
    void setPlantId(int plantId);
    void setDurationToHarvest(int durationToHarvest);
    void setAge(int age);
    PlantType getPlantType() const;
    int getPlantId() const;
    int getDurationToHarvest() const;
    int getAge() const;
    Plant &operator++();
    Plant operator++(int);
    vector<shared_ptr<Product>> collect();
    bool checkReadyToHarvest();
    static map<string, vector<string>> &getHarvestResult();
};

#endif
```

```
#ifndef __PRODUCT_HPP_
#define __PRODUCT_HPP_

#include "Item.hpp"

class Product : public Item{
private:
    ProductType type;
    int productId;
    string origin;
    float addedWeight;

public:
    Product();
    Product(string name);
    virtual ~Product();
    void setProductType(ProductType type);
    void setProductId(int productId);
    void setOrigin(float origin);
    void setAddedWeight(float addedWeight);
    ProductType getProductType() const;
    int getProductId() const;
    string getOrigin() const;
    float getAddedWeight() const;
};

#endif
```

### Keterangan

Super Class : Item

Sub Class : Animal, Building, Plant, Product

Penjelasan :

Kelas Item berperan sebagai kelas dasar yang merepresentasikan item dalam permainan. Atribut-atribut umum seperti nama, kode,

dan harga diwarisi kepada semua entitas lainnya. Penggunaan inheritance pada kelas Item bertujuan untuk membagi item menjadi 4 subkelas yaitu Animal (Hewan), Building (Bangunan), Plant (Tumbuhan), dan Product. Kelas Animal memiliki atribut tambahan yaitu berat, berat minimum untuk panen, dan objek Harvester yang berkaitan dengan proses panen. Metode collect() digunakan untuk mengumpulkan produk dari hewan dan metode checkReadyToHarvest() berfungsi untuk memeriksa apakah hewan siap dipanen. Kelas Plant memiliki atribut tambahan yaitu jenis tumbuhan, umur, dan waktu panen. Seperti Animal, Plant juga memiliki metode untuk mengumpulkan produk (collect()) dan memeriksa kesiapan untuk panen (checkReadyToHarvest()). Kelas Building menyimpan informasi tentang material/bahan bangunan yang dibutuhkan untuk membangun bangunan tersebut. Metode seperti getQuantityPerMaterial() dan getBuildingMaterial() digunakan untuk mengakses informasi tentang material bangunan. Kelas Product memiliki atribut tambahan seperti jenis produk, asal produk, dan berat tambahan. Semua kelas ini menggunakan konsep inheritance untuk memanfaatkan atribut dan metode yang umum diwarisi dari kelas Item, serta menambahkan atribut dan metode khusus yang unik untuk masing-masing jenis item. Struktur dari kelas item dan turunannya ini juga disusun sedemikian rupa untuk mendukung konsep Polymorphism yang akan digunakan di Inventory (Item), Ranch(Animal), dan Farm(Plant).

Super Class	Sub Class
-------------	-----------

```

#ifndef __ANIMAL_HPP_
#define __ANIMAL_HPP_

#include "Product.hpp"
#include <string>
#include <vector>
#include <memory>
#include "AnimalException.hpp"
using namespace std;

class Animal : public Item{
private:
    int animalId;
    int weight;
    int weighToHarvest;
    static map<string, vector<string>> harvestResult;

public:
    Animal();
    Animal(int weight, string name);
    virtual ~Animal();
    void setAnimalType(AnimalType type);
    void setAnimalId(int animalId);
    void setWeight(int weight);
    void setWeightToHarvest(int weightToHarvest);
    int getAnimalId() const;
    int getWeight() const;
    int getWeightToHarvest() const;
    vector<shared_ptr<Product>> collect();
    bool checkReadyToHarvest();
    static map<string, vector<string>>& getHarvestResult();

    virtual void eat(const shared_ptr<Product>& food);

};

#endif

```

```

#ifndef __CARNIVORE_HPP_
#define __CARNIVORE_HPP_

#include "Animal.hpp"

class Carnivore : public Animal{
public :
    Carnivore(string name, int weight);
    Carnivore(string name);
    void eat(const shared_ptr<Product>& product);
    ~Carnivore();

};

#endif

```

```

#ifndef __HERBIVORE_HPP_
#define __HERBIVORE_HPP_

#include "Animal.hpp"
class Herbivore : public Animal{
public :
    Herbivore(string name, int weight);
    Herbivore(string name);
    void eat(const shared_ptr<Product>& product);
    virtual ~Herbivore();

};

#endif

```

```

#ifndef __OMNIVORE_HPP_
#define __OMNIVORE_HPP_

#include "Animal.hpp"

class Omnivore : public Animal{
public :
    Omnivore(string name, int weight);
    Omnivore(string name);
    void eat(const shared_ptr<Product>& product);
    ~Omnivore();
};

#endif

```

### Keterangan

Super Class : Animal

Sub Class : Carnivore, Herbivore, Omnivore

Penjelasan :

Kelas Animal adalah subkelas dari kelas Item yang merepresentasikan entitas hewan dalam permainan. Kelas ini memiliki keterkaitan dengan kelas-kelas turunannya yaitu Carnivore, Herbivore, dan Omnivore, di mana setiap kelas turunan ini mengimplementasikan perilaku makan dari hewan sesuai dengan jenis makanannya. Kelas Carnivore, Herbivore, dan Omnivore adalah turunan dari kelas Animal yang mewakili jenis-jenis makanan yang berbeda yang bisa dimakan oleh hewan. Masing-masing kelas ini memiliki perilaku makan yang spesifik sesuai dengan jenis makanannya, yang diimplementasikan dalam metode eat(). Pada metode eat(), kelas Carnivore hanya bisa mengonsumsi makanan/produk dengan tipe PRODUCT\_ANIMAL, kelas Herbivore hanya bisa mengonsumsi makanan/produk dengan tipe PRODUCT\_FRUIT\_PLANT, sedangkan kelas Omnivore bisa mengonsumsi makanan/produk dengan tipe PRODUCT\_ANIMAL atau PRODUCT\_FRUIT\_PLANT. Struktur kelas-kelas ini menggambarkan hierarki objek di mana Animal menjadi kelas dasar yang mencakup semua hewan, dan turunannya merepresentasikan perbedaan perilaku spesifik dari setiap jenis hewan berdasarkan jenis makanannya.

### 2.1.3. Penggunaan Inheritance dan Polymorphism Pada Grid

Super Class	Sub Class
<pre>#ifndef _GRID_HPP_ #define _GRID_HPP_ #include &lt;memory&gt; #include &lt;vector&gt; #include &lt;map&gt; #include &lt;iomanip&gt; using namespace std; You, 2 minutes ago   1 author (You) template &lt;class T&gt; class Grid { protected:     vector&lt;vector&lt;shared_ptr&lt;T&gt;&gt;&gt; buffer;     int row;     int col;     int emptySlot;     void parseInput(string s, int &amp;intt, char &amp;charr) const;     void parseInput(string s, int &amp;row, int &amp;col) const;  public:     Grid(int r, int c);     ~Grid();     vector&lt;vector&lt;shared_ptr&lt;T&gt;&gt;&gt; &amp;getBuffer(); You, 2 minutes     int getRow() const;     int getCol() const;     bool isFull();     bool isEmpty();     bool isEmpty(const string &amp;slot);     int countEmpty() const;     int countOccupied() const;     void operator+(const shared_ptr&lt;T&gt; item);     shared_ptr&lt;T&gt; &amp;see(int i, int j);     void remove(int i, int j);     void put(string slot, const shared_ptr&lt;T&gt; val);     shared_ptr&lt;T&gt; take(string slot);     void remove(string slot);     shared_ptr&lt;T&gt; &amp;see(string slot);     template &lt;class U&gt;         friend ostream &amp;operator&lt;&lt;(ostream &amp;os, const Grid&lt;U&gt; &amp;g); };  #include "Grid.hpp" #endif</pre>	<pre>#ifndef __INVENTORY_HPP__ #define __INVENTORY_HPP__  #include "../Grid/Grid.hpp" #include "../Item/Item.hpp" #include "../Item/Product.hpp"  class Inventory: public Grid&lt;Item&gt; { public:     Inventory(int r, int c, Item defaultValue);     Inventory();     ~Inventory();     void displayStorage(bool printInfo);     bool checkInventoryEdible();     bool checkInventoryAnimal();     bool checkInventoryPlant();     int countItemStock(const Item&amp;);      void useItem(const Item&amp;, int);     bool isInventoryEnough(const int&amp;);      void putRandom(const shared_ptr&lt;Item&gt; );      bool checkInventoryFruit();     bool checkInventoryMeat();     int countInventoryItem(); };  #endif</pre>

```
#ifndef _SPECIALIZED_GRID_HPP_
#define _SPECIALIZED_GRID_HPP_

#include <string>
#include <iostream>
#include "../Grid/Grid.hpp"      You, 2 days ago ·

using namespace std;

You, 4 minutes ago | 1 author (You)
template <class T>
class SpecializedGrid : public Grid<T>
{

public:
    SpecializedGrid(int r, int c);
    ~SpecializedGrid();
    void displayStorage(const string &title);
    bool checkReadyToHarvest();
    map<string, int> countItemsReadyToHarvest();
};

#include "SpecializedGrid.hpp"

#endif
```

### Keterangan

Super Class : Grid  
Sub Class : Inventory, SpecializedGrid

**Penjelasan :**

Penggunaan inheritance pada Grid bertujuan untuk membagi kelas Grid menjadi beberapa subkelas, yakni Inventory (Penyimpanan) dan SpecializedGrid. Kelas Grid pada dasarnya merupakan implementasi dari struktur penyimpanan generik dua dimensi, yang memungkinkan penggunanya untuk melakukan *method* untuk memanipulasi objek di dalam penyimpanan seperti *see*, *put*, *take*, dan *remove*. Kelas Inventory merupakan grid of items, di mana terdapat *method* yang memanipulasi dan memvalidasi objek item yang terdapat di dalam kelas tersebut, seperti *checkInventoryEdible()*, *checkInventoryPlant()*, *checkInventoryAnimal()*, *displayStorage()*, dan *method* lainnya. Kelas SpecializedGrid merupakan subclass dari Grid yang tipe objeknya adalah template, sama seperti kelas Grid. Kelas SpecializedGrid ini memiliki method tambahan yaitu *displayStorage()*. Inheritance cocok diterapkan pada kelas-kelas ini karena subclass bisa menggunakan method-method superclass untuk memanipulasi isi gridnya seperti *see*, *put*, *take*, dan *remove*. Sedangkan polymorphism memungkinkan method *displayStorage()* dalam kelas SpecializedGrid dan Inventory untuk berperilaku secara berbeda sesuai dengan kebutuhan kelas tersebut. Tampilan yang dihasilkan oleh *displayStorage* dapat berbeda antara grid untuk Inventory dan grid untuk SpecializedGrid, namun keduanya dapat dipanggil menggunakan metode yang sama.

<b>Super Class</b>	<b>Sub Class</b>
--------------------	------------------

```
#ifndef _SPECIALIZED_GRID_HPP_
#define _SPECIALIZED_GRID_HPP_

#include <string>
#include <iostream>
#include "../Grid/Grid.hpp"      You, 2 days ago ·

using namespace std;

You, 4 minutes ago | 1 author (You)
template <class T>
class SpecializedGrid : public Grid<T>
{
public:
    SpecializedGrid(int r, int c);
    ~SpecializedGrid();
    void displayStorage(const string &title);
    bool checkReadyToHarvest();
    map<string, int> countItemsReadyToHarvest();
};

#include "SpecializedGrid.tpp"

#endif
```

```
#ifndef __FARM_HPP__
#define __FARM_HPP__

#include "../SpecializedGrid/SpecializedGrid.hpp"
#include "../Item/Plant.hpp"
#include <map>
using namespace std;

class Farm : public SpecializedGrid<Plant> {
public:
    Farm(int r, int c, Plant defaultValue);
    Farm();
    ~Farm();
    void displayStorage(bool printInfo);
    void addPlantAge();
};

#endif |
```

```

#ifndef __RANCH_HPP_
#define __RANCH_HPP_

#include "../SpecializedGrid/SpecializedGrid.hpp"
#include "../Item/Animal.hpp"
using namespace std;

class Ranch: public SpecializedGrid<Animal> {
public:
    Ranch(int r, int c, Animal defaultValue);
    Ranch();
    ~Ranch();
    void displayStorage(bool printInfo);
    int countAnimalType();
};

#endif

```

Super Class : SpecializedGrid

Sub Class : Farm, Ranch

Penjelasan :

Penggunaan kelas SpecializedGrid ditujukan untuk mengelompokkan fungsi-fungsi yang sama yang terdapat pada kelas Ranch dan Farm. Kelas ini dibagi menjadi dua kelas yaitu Ranch (Pertanian) dan Farm (Ladang). Kelas Farm merupakan grid of plants, di mana *method* yang terdapat di kelas tersebut bertujuan untuk memvalidasi dan memanipulasi objek plant yang terdapat di kelas tersebut, yakni displayStorage() dan addPlantAge(). Kelas Ranch merupakan grid of animals, di mana *method* yang terdapat di kelas tersebut berfungsi untuk melakukan validasi dan manipulasi terhadap objek animal yang terdapat pada kelas tersebut, yakni displayStorage(). Alasan cocok untuk diterapkan inheritance pada kelas-kelas ini adalah karena method displayStorage untuk kelas Ranch dan kelas Farm pada dasarnya adalah sama, hanya berbeda pada tipe objeknya saja dan detail info yang diperlukan. Penggunaan polimorfisme dalam metode displayStorage memungkinkan kelas SpecializedGrid untuk mengubah perilaku tampilan grid sesuai dengan kebutuhan Farm dan Ranch. Dalam method displayStorage, dapat menentukan tampilan yang berbeda untuk hewan di Ranch dan tanaman di Farm. Penggunaan inheritance di sini juga dipilih karena Farm dan Ranch memiliki beberapa

fungsi yang sama yang bisa diletakkan di SpecializedGrid (superclass) dan terdapat juga fungsi-fungsi spesifik yang khusus untuk masing-masing class yang bisa diletakkan di masing-masing class tersebut.

## 2.2. Method/Operator Overloading

Operator overloading memungkinkan pengembangan ulang fungsi operator spesifik untuk kelas atau struktur yang ditentukan pengguna, sehingga mereka dapat beroperasi mirip dengan tipe data asli yang disediakan oleh bahasa pemrograman. Ini berarti operator standar seperti +, -, \*, /, ==, <<, ++, dan != bisa disesuaikan untuk bekerja dengan tipe data yang dibuat oleh pengguna. Dalam praktiknya, operator tersebut diimplementasikan sebagai metode dalam kelas atau struktur tersebut. Sebagai ilustrasi, untuk memungkinkan membandingkan dua objek dari sebuah kelas kustom menggunakan operator ==, kita perlu mengatur ulang implementasi metode operator == di dalam kelas itu.

### 2.2.1. Penggunaan Operator Overloading pada Grid

Screenshot	Keterangan
------------	------------

```

template <class T>
ostream &operator<<(ostream &os, const Grid<T> &g)
{
    // Cetak header kolom
    os << "    ";
    for (int j = 0; j < g.col; ++j)
    {
        os << setw(5) << (char)('A' + j) << " ";
    }
    os << "\n";

    // Cetak garis pemisah
    os << "    ";
    for (int j = 0; j < g.col; ++j)
    {
        os << "-----";
    }
    os << "+\n";

    // Cetak isi grid
    for (int i = 0; i < g.row; ++i)
    {
        // Cetak nomor baris
        os << setw(2) << setfill('0') << (i + 1) << " |";
        for (int j = 0; j < g.col; ++j)
        {
            // Cetak isi sel
            if (g.buffer[i][j] != nullptr)
            {
                os << " " << setw(3) << setfill(' ') << *g.buffer[i][j] << " |";
            }
            else
            {
                os << " " << setw(3) << setfill(' ') << "   "
                    << " |";
            }
        }
        os << "\n";

        // Cetak garis pemisah setelah setiap baris
        os << "    ";
        for (int j = 0; j < g.col; ++j)
        {
            os << "-----";
        }
        os << "+\n";
    }
    return os;
};

```

Operator overloading pada kelas Grid dapat mencetak Grid beserta dengan isi dari grid tersebut yang dijadikan penyimpanan, peternakan , dan pertanian dengan mudah menggunakan operator “<<”.

```
template <class T>
void Grid<T>::operator+(const shared_ptr<T> item)
{
    if (isFull())
    {
        throw GridFullException();
    }
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (this->buffer[i][j] == nullptr)
            {
                this->buffer[i][j] = item;
                emptySlot--;
                return;
            }
        }
    }
};
```

Operator overloading pada kelas Grid dapat menambahkan item ke dalam objek Objek Grid seperti penyimpanan, peternakan dan pertanian dengan mudah menggunakan operator “+” .

## 2.2.2. Penggunaan Operator Overloading pada Item

Screenshot	Keterangan
<pre>#ifndef __ITEM_HPP_ #define __ITEM_HPP_  #include &lt;iostream&gt; #include &lt;string&gt; #include "../configClass/PlantConfig.hpp" #include "../configClass/AnimalConfig.hpp" #include "../configClass/ProductConfig.hpp" #include "../configClass/RecipeConfig.hpp"  using namespace std;  class Item { protected:     static int countIdItem;     int itemId;     string name;     string code;     int price;  public:     Item();     Item(string name, string code, int price);     // Item(Item&amp; other);     virtual ~Item();     void setId(int itemId);     void setName(string name);     void setCode(string code);     void setPrice(int price);     const int&amp; getId() const;     const string&amp; getName() const;     const string&amp; getCode() const;     const int&amp; getPrice() const;     bool operator==(const Item &amp;);      friend ostream &amp;operator&lt;&lt;(ostream &amp;, const Item &amp;); };  #endif</pre>	<p>Operator overloading pada kelas Item dapat membandingkan dua objek Item dengan mudah menggunakan operator “==” , serta melakukan output kode dari Item dengan mudah menggunakan operator “&lt;&lt;” .</p>

### 2.2.3. Penggunaan Operator Overloading pada Plant

Screenshot	Keterangan
<pre>#ifndef __PLANT_HPP_ #define __PLANT_HPP_  #include "Item.hpp" #include "../Harvester/Harvester.hpp"  class Plant : public Item { private:     PlantType type;     int plantId;     int durationToHarvest;     int age;     // vector&lt;Product&gt; result;     Harvester harvester;      static map&lt;string, vector&lt;string&gt;&gt; harvestResult;  public:     Plant();     Plant(int age, string name);     ~Plant();     Plant(string name);     void setPlantType(PlantType type);     void setPlantId(int plantId);     void setDurationToHarvest(int durationToHarvest);     void setAge(int age);     PlantType getPlantType() const;     int getPlantId() const;     int getDurationToHarvest() const;     int getAge() const;     Plant &amp;operator++();     Plant operator++(int);     vector&lt;shared_ptr&lt;Product&gt;&gt; collect();     bool checkReadyToHarvest();     static map&lt;string, vector&lt;string&gt;&gt; &amp;getHarvestResult(); };  #endif</pre>	<p>Operator overloading pada kelas Plant dapat dengan mudah menambahkan umur dari Plant menggunakan operator “++”.</p>

## 2.2.4. Penggunaan Operator Overloading pada Player

Screenshot	Keterangan
<pre>#ifndef _PLAYER_HPP_ #define _PLAYER_HPP_  #include &lt;iostream&gt; #include "../Inventory/Inventory.hpp" #include "../Store/Store.hpp" using namespace std;  class Player { protected:     static int countIdPlayer;     int playerId;     string type;     string username;     int weight;     int gulden;     Inventory inventory;  public:     Player();     Player(string username, int weight, int gulden);     Inventory &amp;getInventory();     string &amp;getName();     virtual ~Player();     int &amp;getWeight();     virtual string getType() = 0;     virtual void buy(shared_ptr&lt;Item&gt; &amp;, int);     virtual pair&lt;vector&lt;shared_ptr&lt;Item&gt;&gt;, int&gt; sell(vector&lt;string&gt; &amp;);     int &amp;getGulden();     void setGulden(int);     bool operator==(const Player&amp;);     void eat(); };  #endif</pre>	Operator overloading pada kelas Player dapat dengan mudah membandingkan dua objek Player menggunakan operator “==”

## 2.2.5. Penggunaan Operator Overloading pada Store

Screenshot	Keterangan
<pre>#ifndef __STORE_HPP_ #define __STORE_HPP_  #include &lt;iostream&gt; #include &lt;string&gt; #include &lt;vector&gt; #include &lt;memory&gt; #include &lt;map&gt; #include "../Item/Item.hpp" using namespace std;  class Store{ private :     vector&lt;string&gt; unlimitedAnimalSell;     vector&lt;string&gt; unlimitedPlantSell;     vector&lt;string&gt; itemsCanSell() const;     map&lt;string, vector&lt;shared_ptr&lt;Item&gt;&gt;&gt; items;     bool checkIsLivingBeings(const string&amp; name);  public :     Store();     void setUnlimitedAnimalSell();     void setUnlimitedPlantSell();     bool checkQuantity(const string&amp;, const int&amp;);     map&lt;string, vector&lt;shared_ptr&lt;Item&gt;&gt;&gt; getItems() const;     vector&lt;shared_ptr&lt;Item&gt;&gt; takeItem(const string&amp; name, const int&amp;);     void addItem(shared_ptr&lt;Item&gt; item);     void addItem(vector&lt;shared_ptr&lt;Item&gt;&gt;&amp; items);      friend ostream&amp; operator&lt;&lt;(ostream&amp; os, const Store&amp; store);     void handleCustomerBuy();     void handleCustomerSell();      ~Store();     void saveFile(const string&amp; filename); };  #endif</pre>	<p>Operator overloading pada kelas Store dapat dengan mudah menampilkan isi dari store beserta harganya menggunakan operator “&lt;&lt;”</p>

## 2.2.6. Penggunaan Operator Overloading pada RecipeConfig

Screenshot	Keterangan
<pre>#ifndef __RECIPECONFIG_HPP__ #define __RECIPECONFIG_HPP__  #include &lt;string&gt; #include &lt;map&gt;  using namespace std;  class RecipeConfig { public :     int id;     string name;     string code;     int price;     map&lt;string, int&gt; materials;      RecipeConfig();     RecipeConfig(int id, string name, string code, int price, map&lt;string, int&gt; materials);     ~RecipeConfig();     friend ostream&amp; operator&lt;&lt;(ostream&amp;, map&lt;string, RecipeConfig&gt;&amp;);  };  #endif</pre>	Operator overloading pada kelas RecipeConfig dapat dengan mudah menampilkan bahan-bahan yang diperlukan untuk membangun sebuah rumah.

## 2.3. Template & Generic Classes/Function

Template & Generic Class ini berfungsi untuk mempermudah dalam membuat kelas yang dapat bekerja dengan berbagai jenis tipe data tanpa harus menulis implementasi kelas yang terpisah untuk setiap tipe data. Template & Generic Class digunakan pada bagian Grid karena berdasarkan spek yang diberikan terdapat persamaan prinsip dasar untuk penyimpanan, peternakan, dan ladang namun hanya berbeda tipe data yang akan ditampung saja. Oleh sebab itu untuk membuat kode yang lebih efisien dapat digunakan pendekatan Template & Generic Class ini untuk membuat class Grid yang nantinya bisa digunakan sebagai dasar untuk membuat class lain seperti Inventory, Ranch, dan Farm. Pada Class Grid dipakai template<class T> sehingga template ini dapat diterima dalam bentuk class Item pada inventory, Animal pada ranch, dan Plant pada farm. Selain digunakan pada Grid template class juga digunakan pada class SpecializedGrid yang merupakan turunan dari Grid itu sendiri. Class SpecializedGrid ini dibuat untuk dijadikan dasar dari

pembuatan class Ranch dan Farm dikarenakan kedua class tersebut memiliki beberapa fungsi yang sama yang bisa dikelompokkan seperti fungsi pengecekan apakah hewan atau tanaman di dalamnya sudah bisa dipanen atau tidak dan lain sebagainya. Penggunaan template class untuk membuat berbagai class Grid ini memberikan keuntungan dimana membuat kode yang dibuat menjadi lebih efisien dan tidak banyak kode yang berulang hanya karena tipe datanya yang berbeda.

Selain menggunakan template & generic class pada tubes ini juga digunakan generic & template function. Generic & Template function ini digunakan untuk mengimplementasi beberapa perintah pada peternak dan petani. Peternak dan petani memiliki beberapa perintah yang sama namun hanya berbeda di objek apa yang akan digunakan untuk menjalankannya, sebagai contoh adalah perintah panen dimana peternak membutuhkan Peternakan (Ranch) sedangkan petani membutuhkan Ladang(Farm). Untuk membuat kode yang lebih efisien pendekatan generic dan template function juga akhirnya dipilih untuk membuat sebuah fungsi template dasar yang nantinya akan diimplementasikan tipe datanya pada fungsi-fungsi di petani dan peternak. Ada dua fungsi template pada tubes ini yang semuanya terletak pada class Resident. Fungsi template pertama adalah harvest yang menjadi template dasar fungsi harvest di Petani (Farmer) dan Peternak (Breeder). Fungsi template kedua adalah placeTo yang menjadi fungsi template dasar untuk fungsi cattle (ternak) di peternak dan fungsi plant (tanam) di petani. Keuntungan dari penggunaan template function ini adalah membuat kode lebih efisien dan mengurangi kode yang berulang. Selain menggunakan template function sebenarnya permasalahan ini bisa saja diselesaikan dengan menggunakan Polymorphism namun tidak dipilih untuk mengurangi jumlah casting yang mungkin akan dilakukan.

<b>Template &amp; Generic Class</b>	
<b>Class Grid</b>	<b>Penjelasan</b>

```

#ifndef _GRID_HPP_
#define _GRID_HPP_
#include <memory>
#include <vector>
#include <map>
#include <iomanip>
using namespace std;
You, 2 minutes ago | 1 author (You)
template <class T>
class Grid
{
protected:
    vector<vector<shared_ptr<T>>> buffer;
    int row;
    int col;
    int emptySlot;
    void parseInput(string s, int &intt, char &charr) const;
    void parseInput(string s, int &row, int &col) const;

public:
    Grid(int r, int c);
    ~Grid();
    vector<vector<shared_ptr<T>>> &getBuffer();| You, 2 minutes
    int getRow() const;
    int getCol() const;
    bool isFull();
    bool isEmpty();
    bool isEmpty(const string &slot);
    int countEmpty() const;
    int countOccupied() const;
    void operator+(const shared_ptr<T> item);
    shared_ptr<T> &see(int i, int j);
    void remove(int i, int j);
    void put(string slot, const shared_ptr<T> val);
    shared_ptr<T> take(string slot);
    void remove(string slot);
    shared_ptr<T> &see(string slot);
    template <class U>
    friend ostream &operator<<(ostream &os, const Grid<U> &g);
};

#include "Grid.tpp"

#endif

```

Implementasi template class terletak di file .tpp

Class Grid digunakan untuk menampung objek dengan tipe data yang berbeda yang tipenya bisa ditentukan saat inisiasi class Grid. Pada tubes ini Grid digunakan untuk membuat Penyimpanan, Peternakan, dan Ladang. Untuk Penyimpanan tipe data yang digunakan adalah Item, untuk Peternakan adalah Animal, dan terakhir untuk Ladang tipe data yang digunakan adalah Plant

### Class SpecializedGrid

### Penjelasan

```
#ifndef _SPECIALIZED_GRID_HPP_
#define _SPECIALIZED_GRID_HPP_

#include <string>
#include <iostream>
#include "../Grid/Grid.hpp"      You, 2 days ago ·

using namespace std;

You, 4 minutes ago | 1 author (You)
template <class T>
class SpecializedGrid : public Grid<T>
{

public:
    SpecializedGrid(int r, int c);
    ~SpecializedGrid();
    void displayStorage(const string &title);
    bool checkReadyToHarvest();
    map<string, int> countItemsReadyToHarvest();
};

#include "SpecializedGrid.tpp"

#endif
```

Implementasi template class terletak di file .tpp

Class ini adalah class turunan dari Grid class ini dibuat untuk mengelompokkan fungsi-fungsi yang sama yang ada di Ladang dan Peternakan untuk mengurangi jumlah kode yang berulang. Class ini perlu dibuat karena terdapat fungsi-fungsi yang hanya ada di Ladang dan Peternakan namun tidak ada di Penyimpanan sehingga fungsi-fungsi yang sama tersebut tidak bisa diletakkan sembarangan di class Grid. Class ini akan menjadi superclass untuk class Ranch (peternakan) dan Farm (Ladang).

## Template & Generic Functions

Function harvest	Penjelesan
<pre> 5   template &lt;class T, class U&gt; 6   void Resident::harvest(T &amp;source, map&lt;string, U&gt; &amp;config, map&lt;string, vector&lt;string&gt;&gt; harvestResult, const string &amp;name) 7   { 8       if (source.isEmpty()) 9       { 10           cout &lt;&lt; endl 11             &lt;&lt; name &lt;&lt; " anda kosong" &lt;&lt; endl; 12       } 13       else if (!source.checkReadyToHarvest()) 14       { 15           cout &lt;&lt; endl 16             &lt;&lt; name &lt;&lt; " anda tidak ada yang siap dipanen" &lt;&lt; endl; 17       } 18       else 19       { 20           source.displayStorage(true); 21           cout &lt;&lt; endl 22             &lt;&lt; name &lt;&lt; "(name == \"Peternakan\" ? \"hewan\" : \"tanaman\")" 23             &lt;&lt; " siap panen yang kamu miliki" &lt;&lt; endl; 24           map&lt;string, int&gt; plantReady = source.countItemsReadyToHarvest(); 25           int number = 1; 26           vector&lt;int&gt; total; 27           vector&lt;string&gt; kode; 28           vector&lt;int&gt; sizeResult; 29 30           for (const auto &amp;entry : plantReady) 31           { 32               cout &lt;&lt; " " &lt;&lt; number &lt;&lt; ". " &lt;&lt; config[entry.first].code &lt;&lt; " (" &lt;&lt; entry.second &lt;&lt; " petak siap panen)" &lt;&lt; endl; 33               number++; 34               kode.push_back(config[entry.first].code); 35               total.push_back(entry.second); 36               sizeResult.push_back(harvestResult[entry.first].size()); 37           } 38 39           int answer1, answer2; 40           bool sukses1 = false; 41           while (!sukses1) 42           { 43               cout &lt;&lt; endl 44                 &lt;&lt; "Nomor " &lt;&lt; (name == "Peternakan" ? "hewan" : "tanaman") 45                 &lt;&lt; " yang ingin dipanen (ketik -1 untuk keluar) : "; 46               cin &gt;&gt; answer1; 47 48               if (cin.fail()) 49               { 50                   cin.clear(); 51                   cin.ignore(numeric_limits&lt;streamsize&gt;::max(), '\n'); 52                   cout &lt;&lt; "Input harus berupa angka. Silahkan coba lagi." &lt;&lt; endl; 53                   continue; 54               } 55           } 56       } 57   } 58 }</pre>	<p>Fungsi ini terletak pada class Resident. Fungsi ini akan menjadi template dasar untuk fungsi-fungsi harvest di petani dan peternak. Fungsi ini dibuat dikarenakan perintah panen baik untuk petani dan peternak memiliki behaviour yang sama namun hanya berbeda pada tipe objek apa yang akan digunakan. Contoh penggunaan fungsi ini juga terlihat pada gambar terakhir di samping ini.</p>

```
53
54     if (answer1 == -1)
55     {
56         cout << "Tidak jadi panen!!" << endl;
57         return;
58     }
59     if (answer1 > 0 && answer1 < number)
60     {
61         sukses1 = true;
62     }
63     else
64     {
65         cout << "Masukan salah silahkan masukan kembali!!!" << endl;
66     }
67 }

68 bool sukses2 = false;
69 while (!sukses2)
70 {
71     cout << endl
72     | << "Berapa petak yang ingin dipanen : ";
73     cin >> answer2;
74
75     if (answer2 <= total[answer1 - 1])
76     {
77         if (answer2 * sizeResult[answer1 - 1] > this->inventory.countEmpty())
78         {
79             cout << endl
80             | << "Jumlah penyimpanan tidak cukup!" << endl;
81             return;
82         }
83         else
84         {
85             sukses2 = true;
86         }
87     }
88     else
89     {
90         cout << "Jumlah yang bisa dipanen hanya " << total[answer1 - 1] << "!!!" << endl;
91     }
92 }

93 string slot;
94 vector<string> petak;
95 cout << "Pilih petak yang ingin dipanen: " << endl;
96 for (int i = 0; i < answer2; i++)
97 {
98     bool sukses3 = false;
```

```

101     while (!sukses3)
102     {
103         try
104         {
105             cout << "Petak ke-" << i + 1 << " : ";
106             cin >> slot;
107             if (source.see(slot)->getCode() != kode[answer1 - 1])
108             {
109                 cout << "Petak tidak sesuai. Silahkan input kembali!" << endl;
110             }
111             else
112             {
113                 petak.push_back(slot);
114                 sukses3 = true;
115             }
116         }
117         catch (const exception &e)
118         {
119             cout << e.what() << endl;
120         }
121     }
122
123     vector<shared_ptr<Product>> tempP = source.take(slot)->collect();
124     unsigned int k = 0;
125     while (k < tempP.size())
126     {
127         this->inventory + tempP[k];
128         k++;
129     }
130     cout << endl
131     << answer2 << " petak " << (name == "Peternakan" ? "hewan " : "tanaman ") << kode[answer1 - 1] << " pada petak ";
132     for (int l = 0; l < answer2; l++)
133     {
134         if (l != answer2 - 1)
135         {
136             cout << petak[l] << ",";
137         }
138         else
139         {
140             cout << petak[l];
141         }
142     }
143     cout << " telah dipanen" << endl;
144 }
145 }; You, 13 minutes ago • feat: separate template implementation and fix ...

```

Contoh penggunaan:

```

void Farmer::harvest()
{
    Resident::harvest<Farm, PlantConfig>(this->farm, Game::getPlantConfig(), Plant::getHarvestResult(), "Ladang");
}

```

### Function placeTo

### Penjelasan

```

149 template <class T, class X, class U>
150 void Resident::placeTo(T* target, map<string, X> &config, function<bool()> checkInventory, const string &name)
151 {
152     if (this->inventory.isEmpty())
153     {
154         cout << endl
155         << "Inventory anda kosong!!" << endl;
156     }
157     else if (!checkInventory())
158     {
159         cout << endl
160         << "Inventory anda tidak ada tanaman!!" << endl;
161     }
162     else if (target.isFull())
163     {
164         cout << endl
165         << name << " anda penuh!!" << endl;
166     }
167     else
168     {
169         bool success = false;
170         while (!success)
171         {
172             cout << "Pilih " << (name == "Peternakan" ? "hewan" : "tanaman")
173             << " dari penyimpanan" << endl;
174             this->inventory.displayStorage(false);
175
176             cout << endl
177             << "Slot (ketik q untuk keluar) : ";
178             string slot;
179             cin >> slot;
180
181             if (slot == "q")
182             {
183                 cout << "Tidak jadi melakukan " << (name == "Peternakan" ? "ternak!!" : "tanam!!")
184                 << endl;
185                 return;
186             }
187
188             try
189             {
190                 if (config.find(this->inventory.see(slot)->getName()) != config.end())
191                 {
192                     string itemName = this->inventory.see(slot)->getName();
193                     cout << endl
194                     << "Kamu memilih " << Utils::toTitleCase(itemName) << "." << endl;
195                     cout << endl
196                     << "Pilih petak tanah yang akan " << (name == "Peternakan" ? "ditinggali" : "ditanami") << endl;
197                     shared_ptr<Item> itemRef = this->inventory.take(slot);
198                     shared_ptr<Item> ref = dynamic_pointer_cast<Item>(itemRef);
199
200                     bool done = false;
201                     while (!done)
202                     {
203                         try
204                         {
205                             target.displayStorage(false);
206
207                             cout << endl
208                             << "Petak tanah (ketik q untuk kembali) : ";
209                             string petak;
210                             cin >> petak;
211
212                             if (petak == "q")
213                             {
214                                 this->inventory.put(slot, itemRef);
215                                 done = true;
216                                 break;
217                             }
218
219                             if (!target.isEmpty(petak))
220                             {
221                                 cout << endl
222                                 << "Petak sudah terisi." << endl;
223                                 << "Silahkan masukan petak yang kosong." << endl;
224                             }
225                             else
226                             {
227                                 target.put(petak, ref);
228
229                                 if (name == "Ladang")
230                                 {
231
232                                     cout << endl
233                                     << "Cangkul, cangkul, cangkul yang dalam!!" << endl;
234                                     << Utils::toTitleCase(itemName) << " berhasil ditanam!" << endl;
235
236                                 }
237                                 else
238                                 {
239                                     cout << endl
240                                     << "Dengan hati-hati, kamu meletakkan seekor " << Utils::toTitleCase(itemName) << " di kandang." << endl;
241                                     cout << Utils::toTitleCase(itemName) << " telah menjadi peliharaanmu sekarang!" << endl;
242
243                                 }
244                                 done = true;
245                             }
246                         }
247                         success = true;
248                     }
249                 }
250             }
251         }
252     }
253 }
```

Fungsi ini juga berada pada class Resident. Fungsi ini dibuat karena perintah ternak dan tanam pada petani dan peternak memiliki behaviour yang sama namun hanya berbeda tipe data objek yang dipakai saja. Fungsi ini akan menjadi dasar dari fungsi plant di petani dan fungsi cattle di peternak. Contoh penggunaan dari fungsi ini terdapat pada gambar terakhir disamping ini. Pada fungsi ini juga dikirim sebuah fungsi sebagai parameter yang mana fungsi tersebut akan digunakan untuk melakukan pengecekan apakah terdapat benda dengan tipe yang diinginkan pada penyimpanan.

```

243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283 };

```

Contoh penggunaan:

```

void Farmer::plant()
{
    Resident::placeTo<Farm, PlantConfig, Plant>(
        this->farm, Game::getPlantConfig(), [this]()
    { return this->inventory.checkInventoryPlant(); },
    "Ladang");
}

```

## 2.4. Exception

Exception Class yang dibuat pada program ini merupakan inheritance dari Exception Class bawaan pada C++. Kami memilih untuk membuat class Exception dari turunan class Exception dari C++ untuk mempermudah penggunaan Polymorphism pada saat melakukan catch exception. Exception Class ini berfungsi untuk mengidentifikasi jenis kesalahan yang terjadi selama eksekusi program. Setiap jenis kesalahan biasanya direpresentasikan oleh objek dari kelas-kelas yang berbeda. Exception ini akan memisahkan logika penanganan kesalahan dari logika normal program dan memungkinkan program untuk berjalan secara bersih dan aman bahkan ketika

kesalahan terjadi. Ketika kesalahan terjadi, objek pada Exception Class dibuat dan dilemparkan ke dalam blok try-catch. Blok try-catch digunakan untuk menangkap (catch) objek Exception yang dilemparkan. Exception yang ditangkap biasanya akan memberikan pesan kesalahan atau error.

Class Grid Exception	Keterangan
<pre> 1 #ifndef _GRIDEXCEPTION_HPP 2 #define _GRIDEXCEPTION_HPP 3 4 #include &lt;iostream&gt; 5 #include &lt;exception&gt; 6 7 using namespace std; 8 9 class IndexOutOfBoundsException : public exception 10 { 11 private: 12     string problemSlot; 13     string message; 14 15 public: 16     IndexOutOfBoundsException() : exception(), problemSlot(""), message("Slot tersebut tidak tersedia") { 17     } 18 19     IndexOutOfBoundsException(const string &amp;pSlot) : exception(), problemSlot(pSlot) { 20         string slot = int(this-&gt;problemSlot.size()) &gt; 0 ? "(" + this-&gt;problemSlot + ")" : ""; 21         message = "Slot tersebut tidak tersedia " + slot; 22     } 23 24     ~IndexOutOfBoundsException() {} 25 26     const char *what() const noexcept override 27     { 28         return message.c_str(); 29     } 30 }; 31 </pre>	<p>IndexOutOfBoundsException dilemparkan ketika index yang dimasukkan melebihi batas index pada ukuran Grid.</p>

```

32 class SlotOccupiedException : public exception
33 {
34 private:
35     string problemSlot;
36     string message;
37
38 public:
39     SlotOccupiedException() : exception(), problemSlot(""), message("Slot tersebut telah diisi") {}
40
41     SlotOccupiedException(const string &pSlot) : exception(), problemSlot(pSlot) {
42         string slot = int(this->problemSlot.size()) > 0 ? "(" + this->problemSlot + ")" : "";
43         this->message = "Slot tersebut telah diisi " + slot;
44     }
45
46     ~SlotOccupiedException() {}
47     const char *what() const noexcept override
48     {
49         return message.c_str();
50     }
51 };
52
53

```

SlotOccupiedException dilemparkan ketika slot yang dipilih untuk dimasukkan sesuatu sudah terisi.

```

54 class SlotEmptyException : public exception
55 {
56 private:
57     string problemSlot;
58     string message;
59
60 public:
61     SlotEmptyException() : exception(), problemSlot(""), message("Slot tersebut kosong") {}
62
63     SlotEmptyException(const string &pSlot) : exception(), problemSlot(pSlot) {
64         string slot = int(this->problemSlot.size()) > 0 ? "(" + this->problemSlot + ")" : "";
65         message = "Slot tersebut kosong " + slot;
66     }
67
68     ~SlotEmptyException() {}
69     const char *what() const noexcept override
70     {
71
72         return message.c_str();
73     }
74 };
75

```

SlotEmptyException dilemparkan ketika ingin mengambil sesuatu dari slot yang dipilih pada Grid dan isi dari slot tersebut ternyata kosong.

```

77 class GridFullException : public exception
78 {
79 public:
80     GridFullException() : exception() {}
81
82     ~GridFullException() {}
83     const char *what() const noexcept override
84     {
85         return "Grid telah penuh";
86     }
87 };

```

GridFullException dilemparkan ketika ingin memasukkan sesuatu pada Grid tetapi semua slot sudah penuh.

```

89 class GridEmptyException : public exception
90 {
91 public:
92     GridEmptyException() : exception() {}
93
94     ~GridEmptyException() {}
95     const char *what() const noexcept override
96     {
97         return "Grid kosong";
98     }
99 };

```

GridEmptyException dilemparkan ketika ingin mengambil sesuatu tetapi semua slot pada Grid kosong.

```

101 class InvalidSlotFormatException : public exception
102 {
103 public:
104     InvalidSlotFormatException() : exception() {}
105
106     ~InvalidSlotFormatException() {}
107     const char *what() const noexcept override
108     {
109         return "Masukan slot yang diberikan tidak sesuai dengan format!!!";
110     }
111 };
112
113 #endif

```

InvalidSlotFormatException dilemparkan ketika salah memasukkan format input slot. Contoh masukan slot seharusnya A01, tapi user hanya memasukan A saja atau 01 saja maka exception ini akan dilemparkan.

Class Command Exception	Keterangan
<pre> 1  #ifndef __COMMAND_EXCEPTION_HPP__ 2  #define __COMMAND_EXCEPTION_HPP__ 3 4  #include &lt;exception&gt; 5  #include &lt;string&gt; 6  using namespace std; 7 8  class NoCommandException : public exception 9  { 10 11     public: 12         const char *what() const noexcept override 13         { 14             return "Perintah tersebut tidak ada!!!"; 15         } 16     }; 17 18 19 #endif </pre>	NoCommandException dilemparkan ketika perintah yang dimasukkan salah atau tidak ada.

Class Mayor Exception	Keterangan

```

1  ifndef __MAYOR_EXCEPTION_HPP__
2  define __MAYOR_EXCEPTION_HPP__
3
4  #include <iostream>
5  #include <exception>
6  #include <map>
7  #include <string>
8  using namespace std;
9
10 class NotEnoughMaterialException : public exception
11 {
12     private:
13         string message;
14
15     public:
16         NotEnoughMaterialException(map<string, int> notEnoughMaterial) : exception()
17     {
18             unsigned int i = 1;
19             message = "Kamu tidak punya sumber daya yang cukup! Masih memerlukan ";
20             for (auto &material : notEnoughMaterial)
21             {
22                 message += to_string(material.second) + " " + material.first + (i == notEnoughMaterial.size() ? "!" : ", ");
23                 i++;
24             }
25         }
26         const char *what() const noexcept override
27     {
28
29             return message.c_str();
30     }
31 };

```

NotEnoughMaterialException dilemparkan ketika material atau sumberdaya yang dimiliki tidak mencukupi sehingga tidak dapat melakukan perintah bangun.

```

33     class NoRecipeException : public exception
34     {
35
36     public:
37         const char *what() const noexcept override
38     {
39             return "Kamu tidak punya resep bangunan tersebut!";
40     }
41 };

```

NoRecipeException dilemparkan ketika ingin membangun bangunan tapi resep bangunannya tidak dimiliki atau tidak tersedia.

```
43 class CannotBuyBuildingException : public exception
44 {
45 public:
46     const char *what() const noexcept override
47     {
48
49         return "Tidak bisa membeli bangunan!!!";
50     }
51 };
52
53 #endif
```

CannotBuyBuildingException dilemparkan ketika player tidak dapat membeli bangunan (mayor tidak dapat membeli bangunan).

Class Load Exception	Keterangan
----------------------	------------

```

1 #ifndef _LOAD_EXCEPTION_HPP
2 #define _LOAD_EXCEPTION_HPP
3
4 #include <iostream>
5 #include <exception>
6
7 using namespace std;
8
9 class FileNotFoundException : public exception
10 {
11 private:
12     string filename;
13     string message;
14
15 public:
16     FileNotFoundException() : exception(), filename(""), message("File tersebut tidak tersedia") {
17 }
18
19     FileNotFoundException(const string &filename) : exception(), filename(filename) {
20         string file = int(this->filename.size()) > 0 ? this->filename : "";
21         message = "File " + file + " tidak ditemukan!!";
22     }
23
24     ~FileNotFoundException() {}
25
26     const char *what() const noexcept override
27     {
28         return message.c_str();
29     }
30 };

```

FileNotFoundException dilemparkan ketika nama file yang dimasukkan untuk dibaca tidak ada atau tidak ditemukan.

```

32 class WrongFormatException : public exception
33 {
34 private:
35     string filename;
36     string message;
37
38 public:
39     WrongFormatException() : exception(), filename(""), message("Terdapat kesalahan format pada file yang diberikan!!!") {
40 }
41
42     WrongFormatException(const string &filename) : exception(), filename(filename) {
43         string file = int(this->filename.size()) > 0 ? this->filename : "";
44         message = "Terdapat kesalahan format pada file " + file + " yang diberikan!!!";
45     }
46
47     ~WrongFormatException() {}
48
49     const char *what() const noexcept override
50     {
51         return message.c_str();
52     }
53 };
54
55
56
57 #endif

```

WrongFormatException dilemparkan ketika format pada file state maupun config yang dibaca salah.

Class Player Exception	Keterangan
<pre> 1 #ifndef __PLAYER_EXCEPTION_HPP__ 2 #define __PLAYER_EXCEPTION_HPP__ 3 4 #include &lt;exception&gt; 5 using namespace std; 6 7 class CannotSellBuildingException : public exception 8 { 9 public: 10     const char *what() const noexcept override 11     { 12         return "Tidak bisa menjual bangunan!!!"; 13     } 14 }; 15 </pre>	<p>CannotSellBuildingException dilemparkan ketika player yang rolenya petani atau peternak ingin menjual bangunan. Role tersebut tidak dapat menjual bangunan tetapi dapat membeli bangunan.</p>
<pre> 19 class NotEnoughGuldenException : public exception 20 { 21 public: 22     const char *what() const noexcept override 23     { 24         return "Gulden tidak mencukupi!!!"; 25     } 26 }; 27 </pre>	<p>NotEnoughGuldenException dilemparkan ketika uang yang ingin dipakai untuk membeli tidak mencukupi.</p>

```

29 class InventoryNotEnoughException : public exception
30 {
31 public:
32     const char *what() const noexcept override
33     {
34         return "Inventory tidak cukup!!!!";
35     }
36 };
37 #endif
38
39 
```

InventoryNotEnoughException dilemparkan ketika inventory yang dimiliki sudah penuh.

Class Animal Exception	Keterangan
<pre> 1 #ifndef __ANIMAL_EXCEPTION_HPP__ 2 #define __ANIMAL_EXCEPTION_HPP__ 3 4 #include &lt;exception&gt; 5 #include &lt;string&gt; 6 using namespace std; 7 8 class NullProductException : public exception 9 { 10 11 public: 12     const char *what() const noexcept override 13     { 14         return "Item tersebut bukanlah produk!!!!"; 15     } 16 }; 17 </pre>	<p>NullProductException dilemparkan ketika ingin memberi makan hewan tetapi item yang dipilih dari slot bukan produk atau bisa dikatakan bahwa exception ini akan dilempar ketika hewan mau makan tetapi produk yang akan dimakan ternyata tidak ada (nullptr).</p>

```

19 class CarnivoreEatException : public exception
20 {
21 public:
22     const char *what() const noexcept override
23     {
24         return "Hewan karnivora tidak bisa memakan makanan tersebut dan hanya bisa makan daging (animal product)";
25     }
26 };
27
28

```

CarnivoreEatException dilemparkan ketika item product yang dipilih untuk dikasih makan ke hewan karnivora bukan merupakan produk daging.

```

29 class HerbivoreEatException : public exception
30 {
31 public:
32     const char *what() const noexcept override
33     {
34         return "Hewan herbivora tidak bisa memakan makanan tersebut dan hanya bisa makan tumbuhan (fruit product)";
35     }
36 };
37
38

```

HerbivoreEatException dilemparkan ketika item product yang dipilih untuk dikasih makan ke hewan herbivora bukan merupakan produk buah.

```

public:
    const char *what() const noexcept override
    {
        return "Hewan omnivora tidak bisa memakan makanan tersebut, hanya bisa makan tumbuhan (fruit product) dan daging (animal product)";
    }
};


```

OmnivoreEatException dilemparkan ketika item product yang dipilih untuk dikasih makan ke hewan omnivora bukan merupakan produk daging atau produk buah.

Class Store Exception	Keterangan
-----------------------	------------

```
1 #ifndef __STORE_EXCEPTION_HPP__
2 #define __STORE_EXCEPTION_HPP__
3
4 #include <exception>
5 using namespace std;
6
7 class StockNotEnoughException : public exception
8 {
9 public:
10     const char *what() const noexcept override
11     {
12         return "Jumlah stock pada toko tidak cukup!!!";
13     }
14 };
15 
```

StockNotEnoughException dilemparkan ketika jumlah barang yang ingin dibeli dari toko lebih dari jumlah stok pada toko.

```
17 ~ class WrongSlotsSizeException : public exception
18 {
19 public:
20 ~     const char *what() const noexcept override
21     {
22         return "Jumlah slots penyimpanan tidak sesuai!!!";
23     }
24 };
25 };
26
27
28
29 #endif
```

WrongSlotsSizeException dilemparkan ketika jumlah slot penyimpanan yang dimasukkan user tidak sama dengan jumlah item yang dibeli.

## 2.5. C++ Standard Template Library

STL (Standard Template Library) adalah kumpulan dari berbagai struktur data dan algoritma yang telah didefinisikan secara standar dalam bahasa pemrograman C++. STL menyediakan kelas-kelas dan fungsi-fungsi yang dapat digunakan untuk melakukan berbagai operasi dengan efisien dan mudah. Kami memilih untuk menggunakan STL untuk mempermudah proses kami dalam pembuatan program dan membuat program kami menjadi terlihat lebih sederhana dan efisien. Kode bawaan standard yang sudah disediakan biasanya juga lebih efisien sehingga bisa meningkatkan performa dari program yang kami buat. Pada program ini, STL yang digunakan ada 4 yaitu vector, map, pair, dan algorithm.

1. STL Vector

Struktur data dinamis yang memungkinkan penyimpanan elemen-elemen dalam urutan linier. Vector memungkinkan akses cepat ke elemen-elemennya dan penambahan atau penghapusan elemen dari ujungnya dengan kompleksitas waktu konstan. Vector ini memiliki beberapa fungsi bawaan khusus yang dapat dipakai seperti `push_back()` untuk menambahkan elemen terakhir, `pop_back()` untuk menghapus elemen terakhir, `size()` untuk mengembalikan jumlah elemen, dan fungsi-fungsi lainnya.

2. STL Map

Struktur data asosiatif yang memetakan kunci unik ke nilai tertentu. Setiap elemen dalam map terdiri dari sebuah pasangan kunci-nilai. Jika ingin mengakses sebuah elemen pada map maka kita hanya perlu mencari kuncinya. Beberapa fungsi bawaan khusus yang dapat dipakai yaitu `insert()` untuk memasukkan pasangan kunci-nilai yang baru ke map, `erase()` untuk menghapus pasangan kunci-nilai dari map, `size()` untuk mengembalikan jumlah elemen, `find()` untuk mencari elemen dengan kunci tertentu, dan fungsi-fungsi lainnya.

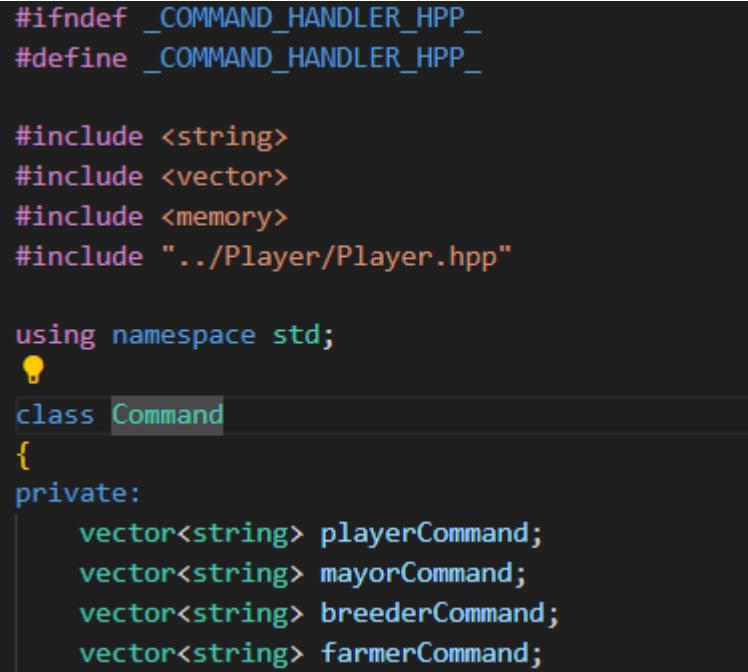
3. STL Pair

Struktur data yang menyimpan dua nilai dan biasanya dengan jenis yang berbeda dalam satu objek. Pair sering digunakan untuk mengembalikan dua nilai dari sebuah fungsi atau memetakan dua nilai bersama-sama. Tidak ada fungsi bawaan khusus tetapi bisa dibuat dan diakses seperti atribut.

4. STL Algorithm

Kumpulan fungsi untuk melakukan operasi umum pada data seperti pencarian, pengurutan, dan transformasi. Beberapa fungsi bawaan yang dapat dipakai yaitu `sort()` untuk mengurutkan elemen, `find_if()` untuk mencari elemen dengan syarat tertentu, `transform()` untuk mengubah setiap elemen menjadi bentuk lain pada suatu struktur data yang berisi kumpulan elemen, dan fungsi-fungsi lainnya.

### 2.5.1. Penggunaan STL Vector

Screenshot	Keterangan
 A screenshot of a code editor showing the implementation of the Command class. The code uses #ifndef and #define directives to handle header files. It includes headers for string, vector, and memory, and a dependency on Player.hpp. The Command class has a private section containing four vector<string> members: playerCommand, mayorCommand, breederCommand, and farmerCommand.	Penggunaan STL vektor untuk kelas Command. Class ini mempunyai atribut vektor String untuk menyimpan command-command apa saja yang bisa dilakukan oleh breeder, farmer, player, dan mayor.

```
class Game
{
private:
    static map<string, AnimalConfig> animalConfig;
    static map<string, PlantConfig> plantConfig;
    static map<string, RecipeConfig> recipe;
    static map<string, ProductConfig> productConfig;
    static MainConfig mainConfig;
    static vector<shared_ptr<Player>> players;
    static shared_ptr<Player> currentPlayer;
    LoadHandler loadHandler;
    Command commandHandler;

    static Store store;
```

Penggunaan STL vektor untuk kelas Game. Class game ini mempunyai atribut vektor share pointer Player untuk menyimpan player-player yang bermain di permainan.

```
#ifndef _GRID_HPP_
#define _GRID_HPP_
#include "GridException.hpp"
#include <iostream>
#include <memory>
#include <vector>
#include <map>
#include <iomanip>
using namespace std;
template <class T>
class Grid
{
protected:
    vector<vector<shared_ptr<T>>> buffer;
    int row;
    int col;
    int emptySlot;
```

Penggunaan STL vektor untuk kelas Grid. Class Grid ini mempunyai atribut vektor dari vektor share pointer untuk template class T untuk menampung objek-objek dengan tipe T.

```
#ifndef __STORE_HPP_
#define __STORE_HPP_

#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <map>
#include "../Item/Item.hpp"
using namespace std;

class Store{
private :
    vector<string> unlimitedAnimalSell;
    vector<string> unlimitedPlantSell;
    vector<string> itemsCanSell() const;
    map<string, vector<shared_ptr<Item>>> items;
    bool checkIsLivingBeings(const string& name);

public :
    Store();
    void setUnlimitedAnimalSell();
    void setUnlimitedPlantSell();
    bool checkQuantity(const string&, const int&);
    map<string, vector<shared_ptr<Item>>> getItems() const;
    vector<shared_ptr<Item>> takeItem(const string& name, const int&);
    void addItem(shared_ptr<Item> item);
    void addItem(vector<shared_ptr<Item>>& items);

    friend ostream& operator<<(ostream& os, const Store& store);
    void handleCustomerBuy();
    void handleCustomerSell();
}
```

Penggunaan STL vektor untuk class Store . Class Store ini mempunyai atribut *unlimitedAnimalSell* dan *unlimitedPlantSell* yang bertipe vektor String untuk menyimpan nama-nama hewan dan tumbuhan yang mempunyai kuantitas unlimited pada Store.

Ada juga sebuah function *itemsCanSell()* yang mengembalikan vektor String berisi items apa saja yang dapat dijual (stock tersedia).

Ada juga sebuah funtion *takeItem()* yang mengembalikan vektor share pointer Item yang akan digunakan untuk menampung item yang akan dibeli atau diambil dari toko dengan kuantitas sesuai yang diinput oleh pembeli.

## 2.5.2. Penggunaan STL Map

Screenshot	Keterangan
------------	------------

```
1 #ifndef _RECIPECONFIG_HPP_
2 #define _RECIPECONFIG_HPP_
3
4 #include <string>
5 #include <map>
6
7 using namespace std;
8
9 class RecipeConfig {
10     public :
11         int id;
12         string name;
13         string code;
14         int price;
15         map<string, int> materials;
16
17     RecipeConfig();
18     RecipeConfig(int id, string name, string code, int price, map<string, int> materials);
19     ~RecipeConfig();
20     friend ostream& operator<<(ostream&, map<string, RecipeConfig>&);
21 };
22
23 #endif
```

Penggunaan STL map untuk class RecipeConfig. Class RecipeConfig memiliki atribut materials yang bertipe map<string, int> dengan string sebagai key dan int sebagai value. Atribut ini digunakan untuk menyimpan nama material dari bangunan (key) dan jumlah material tersebut (value) yang dibaca dari file txt recipe config.

```

1  #ifndef _GAME_HPP_
2  #define _GAME_HPP_
3
4
5  #include "../loadhandler/LoadHandler.hpp"
6  #include "../Command/Command.hpp"
7  #include "../Player/Player.hpp"
8  #include <memory>
9  #include <sys/stat.h>
10 #include <sys/types.h>
11 #include <dirent.h>
12
13 using namespace std;
14
15 class Game
16 {
17 private:
18     static map<string, AnimalConfig> animalConfig;
19     static map<string, PlantConfig> plantConfig;
20     static map<string, RecipeConfig> recipe;
21     static map<string, ProductConfig> productConfig;
22     static MainConfig mainConfig;
23     static vector<shared_ptr<Player>> players;
24     static shared_ptr<Player> currentPlayer;
25     LoadHandler loadHandler;
26     Command commandHandler;
27
28     static Store store;
29

```

Penggunaan STL map untuk class LoadHandler. Class LoadHandler memiliki atribut animalConfig yang bertipe map<string, AnimalConfig> dengan string sebagai key dan AnimalConfig sebagai value. Atribut ini digunakan untuk menyimpan nama hewan (key) dan seluruh data dari hewan tersebut (value) yang dibaca dari file txt animal (config bawaan).

Class LoadHandler memiliki atribut plantConfig yang bertipe map<string, PlantConfig> string sebagai key dan PlantConfig sebagai value. Atribut ini digunakan untuk menyimpan nama tanaman (key) dan seluruh data dari tumbuhan tersebut (value) yang dibaca dari file txt plant (config bawaan).

Class LoadHandler memiliki atribut recipe yang bertipe map<string, RecipeConfig> string sebagai key dan RecipeConfig sebagai value. Atribut ini digunakan untuk menyimpan nama bangunan (key) dan seluruh data dari bangunan tersebut termasuk map material (value) yang dibaca dari file txt recipe (config bawaan).

Class LoadHandler memiliki atribut productConfig yang bertipe map<string, ProductConfig> string sebagai key dan ProductConfig sebagai value. Atribut ini digunakan untuk menyimpan nama produk (key) dan seluruh data dari produk tersebut (value) yang dibaca dari file txt product (config bawaan).

```
1  #ifndef __ANIMAL_HPP_
2  #define __ANIMAL_HPP_
3
4  #include "AnimalException.hpp"
5  #include "../Harvester/Harvester.hpp"
6  using namespace std;
7
8  class Animal : public Item{
9  private:
10    int animalId;
11    int weight;
12    int weighToHarvest;
13    // vector<Product> result;
14    Harvester harvester;
15    static map<string, vector<string>> harvestResult;
16
```

Penggunaan STL map untuk class Animal. Class Animal memiliki atribut harvestResult yang bertipe map<string, vector<string>> dengan string sebagai key dan vector<string> sebagai value. Pada atribut ini menyimpan nama hewan (key) dan nama produk yang dihasilkan dari hewan tersebut (value). Tipe data vector dipakai dalam value karena hasil produknya belum tentu hanya tunggal tetapi bisa juga ada hasil produk yang lain dari hewan tersebut.

```
1 #ifndef __BUILDING_HPP_
2 #define __BUILDING_HPP_
3
4 #include "Item.hpp"
5 #include <map>
6 #include <string>
7
8 class Building : public Item{
9 private:
10     map<string, int> material;
11 public:
12     Building();
13     Building(string name);
14     ~Building();
15     int getQuantityPerMaterial(string materialName) const;
16     map<string, int> getBuildingMaterial() const;
17 };
18
19 #endif
```

Penggunaan STL map untuk class Building. Class Building memiliki atribut material yang bertipe map<string, int> dengan string sebagai key dan int sebagai value. Atribut ini digunakan untuk menyimpan nama material dari bangunan (key) dan jumlah material yang diperlukan untuk membangun suatu bangunan (value).

```
1 ~ #ifndef __PLANT_HPP_
2   #define __PLANT_HPP_
3
4 ~ #include "Item.hpp"
5   #include "../Harvester/Harvester.hpp"
6
7 ~ class Plant : public Item
8 {
9   private:
10     PlantType type;
11     int plantId;
12     int durationToHarvest;
13     int age;
14     // vector<Product> result;
15     Harvester harvester;
16
17     static map<string, vector<string>> harvestResult;
18 }
```

Penggunaan STL map untuk class Plant. Class Plant memiliki atribut harvestResult yang bertipe map<string, vector<string>> dengan string sebagai key dan vector<string> sebagai value. Pada atribut ini menyimpan nama tanaman (key) dan nama produk yang dihasilkan dari tanaman tersebut (value).

```
1  #ifndef __STORE_HPP_
2  #define __STORE_HPP_
3
4  #include <iostream>
5  #include <string>
6  #include <vector>
7  #include <memory>
8  #include <map>
9
10
11
12 #include "../Item/Item.hpp"
13 using namespace std;
14
15 class Store{
16     private :
17         vector<string> unlimitedAnimalSell;
18         vector<string> unlimitedPlantSell;
19         vector<string> itemsCanSell() const;
20         map<string, vector<shared_ptr<Item>>> items;
21         bool checkIsLivingBeings(const string& name);
```

Penggunaan STL map untuk class Store. Class Store memiliki atribut items yang bertipe map<string, vector<shared\_ptr<Item>>> dengan string sebagai key dan vector<shared\_ptr<Item>> sebagai value. Pada atribut ini menyimpan nama item (key) dan kumpulan pointer dari objek item tersebut (value). Atribut ini digunakan untuk menampung item-item pada store yang memiliki jumlah terbatas. Daripada harus menghancurkan dan membuat objek baru setiap kali player menjual atau membeli lebih baik objek tersebut disimpan sehingga bisa digunakan kembali nantinya.

### 2.5.3. Penggunaan STL Pair

Screenshot	Keterangan
<pre>class Command { private:     vector&lt;string&gt; playerCommand;     vector&lt;string&gt; mayorCommand;     vector&lt;string&gt; breederCommand;     vector&lt;string&gt; farmerCommand;     pair&lt;vector&lt;string&gt;, string&gt; allCommandFor(shared_ptr&lt;Player&gt;&amp;);</pre>	Penggunaan STL pair untuk kelas Command. Class Command ini mempunyai sebuah function yang mengembalikan sebuah STL pair ini menyimpan vektor String yang berisi command-command sebagai anggota “first” dan role player nya sebagai anggota “second”. Pair ini digunakan untuk menampung command-command apa saja yang bisa dilakukan oleh sebuah player sesuai dengan role player tersebut.

```
#ifndef _MAYOR_HPP_
#define _MAYOR_HPP_

#include "../Player/Player.hpp"
#include "../Item/Building.hpp"
#include "../configClass/RecipeConfig.hpp"
#include "../Utils/Utils.hpp"
#include <vector>
#include <map>

class Mayor : public Player
{
public:
    Mayor();
    Mayor(string username, float weight, int gulden);
    ~Mayor();
    void taxCollection();
    void buildBuilding();
    void addPlayer();
    pair<vector<shared_ptr<Item>>, int> sell(vector<string> &);
    void buy(shared_ptr<Item> &, int);
    void saveFile(const string &filename);
};

#endif
```

Penggunaan STL pair untuk kelas Mayor. Pada class Mayor ini menggunakan pair sebagai hasil return fungsi sell . Pair ini akan menampung vector pointer Item yang dijual sebagai anggota “first” dan jumlah total harga Item yang dijual sebagai anggota “second”.

```
#ifndef _PLAYER_HPP_
#define _PLAYER_HPP_

#include <iostream>
#include "../Inventory/Inventory.hpp"
#include "../Store/Store.hpp"
using namespace std;

class Player
{
protected:
    static int countIdPlayer;
    int playerId;
    string type;
    string username;
    int weight;
    int gulden;
    Inventory inventory;

public:
    Player();
    Player(string username, int weight, int gulden);
    Inventory &getInventory();
    string &getName();
    virtual ~Player();
    int &getWeight();
    string getType();
    virtual void buy(shared_ptr<Item> &, int);
    virtual pair<vector<shared_ptr<Item>>, int> sell(vector<string> &);
    int &getGulden();
    void setGulden(int);
    bool operator==(const Player&);
    void eat();
};

#endif
```

Penggunaan STL pair untuk kelas Player. Pada class Player ini menggunakan pair sebagai hasil return fungsi sell . Pair ini akan menampung vector pointer Item yang dijual sebagai anggota “first” dan jumlah total harga Item yang dijual sebagai anggota “second”.

## 2.5.4. Penggunaan STL Algorithm

Screenshot	Keterangan
<pre> void Mayor::addPlayer() {     if (guiden &gt;= 50)     {         string playerType;         string playerName;         cout &lt;&lt; "Masukkan jenis pemain: ";         cin &gt;&gt; playerType;         if(playerType != "Paternak" &amp;&amp; playerType != "Petani" &amp;&amp; playerType != "petani" &amp;&amp; playerType != "peternak"){             cout &lt;&lt; "Jenis permainan tersebut tidak ditemukan!" &lt;&lt; endl;             return;         }         cout &lt;&lt; "Masukkan nama pemain: ";         cin &gt;&gt; playerName;         auto itr = find_if(Game::getPlayers().begin(), Game::getPlayers().end(), [&amp;playerName](auto player)                            { return player-&gt;getName() == playerName; });         if (itr == Game::getPlayers().end())         {             shared_ptr&lt;Player&gt; newPlayer;             if (playerType == "Paternak"    playerType == "peternak")             {                 newPlayer = make_shared&lt;Breeder&gt;(playerName, 40, 50);             }             else if (playerType == "Petani"    playerType == "petani")             {                 newPlayer = make_shared&lt;Farmer&gt;(playerName, 40, 50);             }             Game::getPlayers().push_back(newPlayer);             guiden -= 50;             cout &lt;&lt; "Pemain baru ditambahkan!" &lt;&lt; endl;             cout &lt;&lt; "Selamat datang \"'" &lt;&lt; playerName &lt;&lt; "\\" di kota ini!" &lt;&lt; endl;             sort(Game::getPlayers().begin(), Game::getPlayers().end(), [](const auto &amp;a, const auto &amp;b)                  { return a-&gt;getName() &lt; b-&gt;getName(); });         }         else         {             cout &lt;&lt; "Pemain dengan nama " &lt;&lt; playerName &lt;&lt; " telah terdaftar!" &lt;&lt; endl;         }     }     else     {         cout &lt;&lt; "Uang tidak cukup!" &lt;&lt; endl;     } } </pre>	<p>Penggunaan STL Algorithm digunakan untuk melakukan pencarian elemen data yang terdapat pada struktur data STL lainnya dengan kondisi tertentu. Kami menggunakan fungsi <code>find_if</code> yang terdapat pada STL Algorithm. Fungsi <code>find_if</code> ini digunakan untuk memudahkan pencarian sesuai kondisi yang diinginkan tanpa harus melakukan hardcoded dengan iterator.</p> <p>Pada code disamping juga menggunakan STL Algorithm untuk melakukan pengurutan pemain yang ada dalam permainan berdasarkan nama pemain tersebut. Kami menggunakan fungsi <code>sort</code> yang terdapat pada STL Algorithm ini.</p>

```

void Mayor::taxCollection()
{
    vector<shared_ptr<Resident>> residents;
    int tax, sumTax = 0;
    cout << "Cring cring cring..." << endl;
    cout << "Pajak sudah dipungut!" << endl
        << endl;
    cout << "Berikut adalah detil dari pemungutan pajak:" << endl;

    for (auto &player : Game::getPlayers())
    {
        shared_ptr<Resident> resident = dynamic_pointer_cast<Resident>(player);

        if (resident != nullptr)
        {
            residents.push_back(resident);
        }
    }

    sort(residents.begin(), residents.end(), [](auto a, auto b)
    {
        if(a->getWealth() == b->getWealth()){
            return a->getName() < b->getName();
        }
        return a->getWealth() > b->getWealth();
    });

    for (int i = 0; i < int(residents.size()); i++)
    {
        shared_ptr<Resident> &resident = residents[i];
        cout << " " << i + 1 << ". " << residents[i]->getName() << " - " << residents[i]->getType() << ":" ;
        tax = (resident->getGulden() >= resident->calculateTax()) ? resident->calculateTax() : resident->getGulden();
        cout << tax << " gulden" << endl;
        resident->setGulden(resident->getGulden() - tax);
        sumTax += tax;
    }

    cout << "Negara mendapatkan pemasukan sebesar " << sumTax << " gulden." << endl;
    cout << "Gunakan dengan baik dan jangan dikorupsi ya!" << endl;
    gulden += sumTax;
}

```

Penggunaan STL Algorithm digunakan untuk melakukan pengurutan resident yang berisi semua player yang terdapat di pertandingan berdasarkan dengan jumlah kekayaan dari player tersebut. Kami menggunakan fungsi sort untuk melakukan pengurutan pada resident.

```

void Mayor::buildBuilding()
{
    cout << Game::getRecipe() << endl;
    string buildingName;
    bool isValid = false;
    while (!isValid)
    {
        try
        {
            cout << "Bangunan yang ingin dibangun (Ketik q untuk keluar): ";
            cin >> buildingName;
            if (buildingName == "q")
            {
                break;
            }
            auto itr = find_if(Game::getRecipe().begin(), Game::getRecipe().end(), [buildingName](pair<string, RecipeConfig> building)
            {
                return buildingName == building.second.name;
            });
            if (itr != Game::getRecipe().end())
            {
                map<string, int> notEnoughMaterials;
                if (this->gulden < Game::getRecipe()[buildingName].price)
                {
                    notEnoughMaterials["gulden"] = Game::getRecipe()[buildingName].price - this->gulden;
                }
                for (auto material : itr->second.materials)
                {
                    int count = this->inventory.countItemStock(Item{material.first, Game::getProductConfig()[material.first].code, Game::getProductConfig()[material.first].name});
                    if (count < material.second)
                    {
                        notEnoughMaterials[material.first] = material.second - count;
                    }
                }
                if (notEnoughMaterials.size() > 0)
                {
                    throw NotEnoughMaterialException(notEnoughMaterials);
                }
            }
            // Kasus ketika SDA cukup untuk membangun building
            this->gulden -= itr->second.price;
            for (auto material : itr->second.materials)
            {
            }
        }
    }
}

```

Penggunaan STL Algorithm digunakan untuk melakukan pencarian elemen data yang terdapat pada struktur data STL lainnya dengan kondisi tertentu. Kami menggunakan fungsi `find_if` yang terdapat pada STL Algorithm. Pada code disamping, fungsi `find_if` ini digunakan untuk mencari dalam `recipeConfig` apakah terdapat nama bangunan yang sesuai dengan nama bangunan yang diinput oleh pemain.

```

void LoadHandler::loadStateConfig(string filename)
{
    string slot;
    int umur;
    for (int j = 0; j < totalItem; j++)
    {
        // file >> slot >> itemName >> umur;
        if (!(file >> slot >> itemName >> umur))
        {
            throw WrongFormatException(filename);
        }

        player->getFarm().put(slot, make_shared<Plant>(umur, itemName));
    }

    Game::getPlayers().push_back(player);
}

sort(Game::getPlayers().begin(), Game::getPlayers().end(), [](const auto &a, const auto &b)
    { return a->getName() < b->getName(); });

int storeAmount;
string storeItemName;
int amount;
// file >> storeAmount;
if (!(file >> storeAmount))
{
    throw WrongFormatException(filename);
}

for (int j = 0; j < storeAmount; j++)
{
}

```

Penggunaan STL Algorithm digunakan untuk melakukan pengurutan resident yang berisi semua player yang terdapat di pertandingan berdasarkan dengan nama pemain tersebut. Kami menggunakan fungsi sort untuk melakukan pengurutan pada vektor Players yang baru di load dari file config.

```

void Store::setUnlimitedAnimalSell()
{
    transform(Game::getAnimalConfig().begin(), Game::getAnimalConfig().end(), back_inserter(this->unlimitedAnimalSell), [](auto &el)
    { return el.first; });
}
void Store::setUnlimitedPlantSell()
{
    transform(Game::getPlantConfig().begin(), Game::getPlantConfig().end(), back_inserter(this->unlimitedPlantSell), [](auto &el)
    { return el.first; });
}

```

Penggunaan STL Algorithm digunakan untuk mengubah semua data dalam sebuah struktur data sesuai dengan kondisi yang diinginkan. Kami menggunakan fungsi transform untuk mengubah data dalam bentuk map yang menyimpan data Animal dan Plant Config menjadi data bertipe vector dengan mengambil key dari map untuk dijadikan elemen pada vector.

## 2.6. Konsep OOP lain

### 2.6.1. Abstract Base Class

Screenshot	Keterangan
<pre> class Resident : public Player { public:     Resident();     Resident(string, int, int);     ~Resident();     virtual int calculateTax() = 0;     virtual void harvest() = 0;     virtual int getWealth() = 0;  protected:     template &lt;class T, class U&gt; </pre>	<p>Penggunaan dari Abstract Base Class (ABC) yang terdapat pada kelas Resident memiliki setidaknya satu fungsi murni virtual yang tidak diimplementasikan di dalamnya (calculateTax(), harvest(), dan getWealth()). Karena itu, kelas Resident tidak dapat di-instansiasi secara langsung, tetapi harus diturunkan oleh kelas-kelas turunan yang mengimplementasikan fungsi-fungsinya. Ini memungkinkan untuk polimorfisme, di mana kelas-kelas turunan dapat memperluas fungsionalitas sesuai dengan kebutuhan mereka sendiri.</p>

## 2.6.2. Composition

Screenshot	Keterangan

```
class Plant : public Item
{
private:
    PlantType type;
    int plantId;
    int durationToHarvest;
    int age;
    // vector<Product> result;
    Harvester harvester;

    static map<string, vector<string>> harvestResult;

public:
    Plant();
    Plant(int age, string name);
    ~Plant();
    Plant(string name);
    void setPlantType(PlantType type);
    void setPlantId(int plantId);
    void setDurationToHarvest(int durationToHarvest);
    void setAge(int age);
    PlantType getPlantType() const;
    int getPlantId() const;
    int getDurationToHarvest() const;
    int getAge() const;
    Plant &operator++();
    Plant operator++(int);
    vector<shared_ptr<Product>> collect();
```

Penggunaan Composition berfungsi untuk mengelompokkan fungsi terkait. Objek yang saling berhubungan dapat dikelompokkan bersama dalam satu objek yang lebih besar. Misalnya, dalam kasus Plant dan Harvester atau Animal dan Harvester, Harvester adalah bagian yang penting dalam fungsionalitas Plant dan Animal, hubungan antara kelas Plant dan Harvester adalah composition karena Plant memiliki objek Harvester sebagai anggota yang integral begitu juga dengan hubungan Animal dengan Plant. Ini berarti Harvester tidak dapat hidup secara independen dari Plant atau Animal dan keberadaannya terikat pada siklus hidup Plant atau Animal. Dalam composition, objek satu kelas menjadi bagian dari objek kelas lain dan tidak dapat eksis secara terpisah. Kelas Harvester sendiri diperlukan oleh class Plant dan Animal untuk memenuhi fungsi dari Animal dan Plant yaitu collect yang akan dipanggil ketika Animal atau Plant di harvest. Fungsi collect akan menghasilkan Product yang merupakan hasil dari Animal atau Plant ketika dipanen.

```
#ifndef __ANIMAL_HPP__
#define __ANIMAL_HPP__

#include "AnimalException.hpp"
#include "../Harvester/Harvester.hpp" You, 15 hours ago • feat: ad
using namespace std;

You, 15 hours ago | 2 authors (wigaandini and others)
class Animal : public Item{
private:
    int animalId;
    int weight;
    int weighToHarvest;
    Harvester harvester;
    static map<string, vector<string>> harvestResult;

public:
    Animal();
    Animal(int weight, string name);
    virtual ~Animal();
    void setAnimalType(AnimalType type);
    void setAnimalId(int animalId);
    void setWeight(int weight);
    void setWeightToHarvest(int weightToHarvest);
    int getAnimalId() const;
    int getWeight() const;
    int getWeightToHarvest() const;
    vector<shared_ptr<Product>> collect();
    bool checkReadyToHarvest();
    static map<string, vector<string>>& getHarvestResult();

    virtual void eat(const shared_ptr<Product>& food);

};

#endif
```

Pada kelas Animal memiliki anggota Harvester yang disebut harvester. Ini berarti bahwa setiap kali kami membuat objek Animal, itu juga menciptakan objek Harvester yang terkait dengannya. Dalam konsep composition, objek bagian (dalam hal ini, Harvester) tidak dapat eksis tanpa objek utama (Animal). Jika objek Animal dihancurkan, objek Harvester yang terkait juga akan dihancurkan. Ini berbeda dengan hubungan asosiasi di mana objek dapat ada secara independen satu sama lain.

```

class Game
{
private:
    static map<string, AnimalConfig> animalConfig;
    static map<string, PlantConfig> plantConfig;
    static map<string, RecipeConfig> recipe;
    static map<string, ProductConfig> productConfig;
    static MainConfig mainConfig;
    static vector<shared_ptr<Player>> players;
    static shared_ptr<Player> currentPlayer;
    LoadHandler loadHandler;
    Command commandHandler;

    static Store store;
}

```

Class Game sebenarnya ada sebuah class besar yang mempunyai tanggung jawab yang cukup besar yaitu untuk mengatur alur keberlangsungan dari program permainan ini. Untuk meringankan tanggung jawab dari class Game ini digunakan prinsip composition untuk mendelegasikan tanggung jawab yang dimiliki oleh class Game ini kepada class-class lain. Penerapan composition dilakukan disini juga untuk membuat kode program yang dibuat bisa memenuhi prinsip Single Responsibility. Composition yang ada di class Game adalah class LoadHandler yang bertanggung jawab dalam segala proses yang berkaitan dengan muat baik muat config maupun muat state. Selain itu ada juga class Command yang bertanggung jawab dalam hal-hal yang berkaitan dengan command-command yang ada dalam permainan baik untuk pengecekan, pencetakan, maupun untuk “routing” (command apa akan memanggil fungsi apa). Composition disini juga berarti bahwa keberadaan dari LoadHandler dan Command tidak bisa dipisahkan dari Game begitu juga dengan Game yang tidak bisa hidup tanpa adanya LoadHandler dan Command.

### 2.6.3. Static

Screenshot	Keterangan
------------	------------

```

class Game
{
private:
    static map<string, AnimalConfig> animalConfig;
    static map<string, PlantConfig> plantConfig;
    static map<string, RecipeConfig> recipe;
    static map<string, ProductConfig> productConfig;
    static MainConfig mainConfig;
    static vector<shared_ptr<Player>> players;
    static shared_ptr<Player> currentPlayer;
    LoadHandler loadHandler;
    Command commandHandler;

    static Store store;

public:
    Game();
    ~Game();
    static map<string, AnimalConfig> &getAnimalConfig();
    static map<string, PlantConfig> &getPlantConfig();
    static map<string, RecipeConfig> &getRecipe();
    static map<string, ProductConfig> &getProductConfig();
    static MainConfig &getMainConfig();
    static vector<shared_ptr<Player>>& getPlayers();
    static shared_ptr<Player>& getCurrentPlayer();
    static Store &getStore();
    static void setAnimalConfig(const map<string, AnimalConfig> &);
    static void setPlantConfig(const map<string, PlantConfig> &);
    static void setRecipe(const map<string, RecipeConfig> &);
    static void setProductConfig(const map<string, ProductConfig> &);
    static void setMainConfig(const MainConfig &);
    static void setPlayers(vector<shared_ptr<Player>>& );
    static void setCurrentPlayer(int);
    void start();
    static void handleSave();
    void handleLoadState();
    void initializeGame();
    void handleLoadConfig();
    static void handleNext(int);
    void displayWinner(shared_ptr<Player>& );
    static shared_ptr<Player> checkWinner();
};


```

Dalam class Game , juga terdapat implementasi atribut static. Static digunakan untuk mendeklarasikan anggota class yang bersifat statis (atribut milik kelas bukan objek), memungkinkan anggota tersebut berbagi antara semua instance class tanpa terikat pada instance spesifik manapun. Ini berguna untuk situasi seperti berbagi data yang konsisten di semua objek, memungkinkan akses ke fungsi/atribut tanpa harus membuat instance dari class, atau mengelola sumber daya global seperti konfigurasi permainan yang digunakan oleh semua instance. Dalam konteks class Game, atribut static memungkinkan pengaturan atau variabel konfigurasi global, seperti animalConfig, plantConfig, recipe, productConfig, mainConfig, players, currentPlayer , store , untuk diakses dan dimodifikasi secara langsung melalui nama class tanpa perlu instance. Penggunaan ‘static’ meningkatkan efisiensi dan mempermudah kontrol akses.

```
#ifndef __ANIMAL_HPP_
#define __ANIMAL_HPP_

#include "AnimalException.hpp"
#include "../Harvester/Harvester.hpp"
using namespace std;

class Animal : public Item{
private:
    int animalId;
    int weight;
    int weighToHarvest;
    Harvester harvester;
    static map<string, vector<string>> harvestResult;

};

class Plant : public Item
{
private:
    PlantType type;
    int plantId;
    int durationToHarvest;
    int age;
    Harvester harvester;
    static map<string, vector<string>> harvestResult;
};
```

Penggunaan static pada atribut harvestResult pada class Animal dan Plant , untuk menyimpan data hasil panen dari setiap animal dan plant. Hal ini membuat atribut harvestResult ini dapat diakses oleh seluruh objek Animal dan Plant , nilai dari harvestResult sama untuk semua objek Animal dan Plant.

### 3. Bonus Yang Dikerjakan

#### 3.1. Bonus yang diusulkan oleh spek

##### 3.1.1. ERD

Link ERD: <https://drive.google.com/file/d/1-Dw1L2AnhFGMQmVtcUiJfJYb5VQ37hok/view?usp=sharing>

Ini adalah link draw.io ERD berada pada page ERD OOP

## 3.2. Bonus Kreasi Mandiri

### 3.2.1. Coin Flip

Pada fitur beli yang tersedia di toko, ditambahkan opsi pembayaran dengan adanya coin flip. Fitur ini diimplementasikan berdasarkan hal yang sedang lumayan trending saat kegiatan jual beli barang. Fitur ini tidak mengubah fitur beli spek tetapi hanya menambahkan opsi. Fitur coin flip ini pada awalnya akan diminta untuk memilih head and tail sama seperti koin pada umumnya memiliki 2 sisi. Kemudian akan di generate angka secara random antara 1 dan 2 dimana 1 adalah head dan 2 adalah tail. Ketika pilihan dan hasil sama maka harga barang yang dibeli menjadi gratis dan jika salah maka harga menjadi 2 kali lipat.

```

try
{
    itemBuyChoose = this->takeItem(allItemsSell.at(numItemBuy - 1), 1).at(0);
    itemBuys.push_back(itemBuyChoose);
    cout << endl;
    cout << "Terdapat opsi dalam pembayaran: " << endl;
    cout << "1. Bayar dengan harga normal" << endl;
    cout << "2. Coin Flip (Jika benar maka barang gratis, Jika salah maka harga menjadi 2x)" << endl;
    cout << endl;
    bool coinFlipValid = false;
    int answer;

    while (!coinFlipValid)
    {
        cout << "Masukkan pilihan opsi pembayaran: ";
        cin >> answer;

        if (cin.fail())
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Input harus berupa angka. Silahkan coba lagi." << endl;
            continue;
        }

        if (answer == 1)
        {
            Game::getCurrentPlayer()->buy(itemBuyChoose, quantity, 1);
            coinFlipValid = true;
        }
        else if (answer == 2)
        {
            cout << endl
                << "Silahkan pilih head atau tail" << endl;
            string pilihan;

            bool coinChooseValid = false;
            while (!coinChooseValid)
            {
                cout << "Masukkan pilihan: ";
                cin >> pilihan;

                if (pilihan != "head" && pilihan != "tail")
                {
                    cout << "Masukan salah silahkan masukan kembali!!!" << endl;
                }
                else
                {
                    coinChooseValid = true;
                    random_device rd;
                    mt19937 gen(rd());
                    uniform_int_distribution<int> dist(1, 2);

                    int random_number = dist(gen);
                    string x;
                    if (random_number == 1){
                        x = "head";
                    } else {
                        x = "tail";
                    }
                }
            }
        }
    }
}
cout << endl
    << "Koin dilempar" << endl;
cout << "ting ting ting" << endl
    << "ting ting ting" << endl
    << "ting ting ting" << endl;

if (x != pilihan)
{
    cout << endl
        << "Pilihan anda salah, harga menjadi 2x lipat" << endl;
    Game::getCurrentPlayer()->buy(itemBuyChoose, quantity, 2);
}
else
{
    cout << endl
        << "Pilihan anda benar, anda beruntung, harga gratis" << endl;
    Game::getCurrentPlayer()->buy(itemBuyChoose, quantity, 0);
}

coinFlipValid = true;

```

### 3.2.2. Permainan Slot

Pemain dapat melakukan perintah “PERMAINAN\_SLOT” untuk bermain permainan slot. Untuk bermain slot pemain harus membayar 1 gulden. Jika pemain bisa mendapatkan 3 angka yang sama pada hasil slot, maka pemain akan mendapatkan reward berdasarkan symbol slot yang diperoleh. Permainan Slot ini menggunakan sebuah class Slot yang merupakan anak dari class Grid. Untuk pemilihan 3 angka dilakukan dengan random number generator dan hasilnya akan ditampilkan dengan format grid seperti format inventory tetapi dengan ukuran berbeda.

Implementasi Class Slot dapat dilihat di gambar di bawah ini

```
#ifndef __SLOT_HPP_
#define __SLOT_HPP_

#include "../Grid/Grid.hpp"
#include <map>
using namespace std;

You, 4 hours ago | 2 authors (You and others)
class Slot : public Grid<string> {
private:
vector<string> prize;

public:
Slot();
~Slot();
void displaySlot();
pair<int, string> play();
};

#endif
```

```
void Player::playSlot(){
bool isAnswerValid = false;
while (!isAnswerValid){
    string answerState = "";
    cout << "=====";
    cout << " Selamat Datang di Mesin Slot! ";
    cout << "=====";
    cout << " Mutai Permainan dengan 1 Gulden ";
    cout << " Berpeluang untuk Meraih Berbagai Keuntungan! ";
    cout << "=====";
    cout << "Peraturan Permainan:\n";
    cout << ". Jika Anda mendapatkan 3 huruf 'A' yang sama, Anda memenangkan 10 gulden.\n";
    cout << ". Jika Anda mendapatkan 3 huruf 'B' yang sama, Anda memenangkan 20 gulden.\n";
    cout << ". Jika Anda mendapatkan 3 huruf 'C' yang sama, Anda memenangkan 30 gulden.\n";
    cout << "=====";
    cout << endl;

    bool isFinish = false;
    while (!isFinish)
    {
        cout << "Permainan ini memakai 1 gulden\n";
        cout << "Apakah Anda ingin bermain mesin slot? (Y/N) ";
        cin >> answerState;

        Slot slotGame;
        if (answerState == "Y"){
            isAnswerValid = true;
            this->gulden--;
            pair<int, string> res = slotGame.play();
            this->gulden += res.first;
            cout << res.second << endl;
        }
        else if (answerState == "N"){
            break;
        }
        else{
            cout << "Masukan salah silahkan ulangi kembali!!" << endl;
        }
    }

    cout << endl;
    cout << "Terima kasih telah mengunjungi!" << endl;
}
}
```

#### 4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
MAKAN	13522053, 13522113, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
CETAK_PENYIMPANAN	13522113, 13522115, 13522117, 10023575	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
SIMPAN	13522053, 13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
PUNGUT_PAJAK	13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
BANGUN_BANGUNAN	13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
TAMBAH PEMAIN	13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
TANAM	13522113, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
CETAK_LADANG/PETERNAKAN	13522113, 13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
PANEN	13522055, 13522113, 13522117, 10023575	13522053, 13522055, 13522113, 13522115, 13522117, 10023575

TERNAK	13522055, 13522053, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
KASIH_MAKAN	13522055, 13522053, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
JUAL	13522117, 13522053	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
BELI	13522117, 13522053	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
MUAT	13522053, 13522055, 13522113, 13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
Bonus	13522053, 13522055, 13522113, 13522115, 13522117	13522053, 13522055, 13522113, 13522115, 13522117, 10023575
Laporan	13522053, 13522055, 13522113, 13522115, 13522117, 10023575	

## 5. Lampiran

Pranala ke repository: [https://github.com/wigaandini/Tubes1\\_IF2210\\_OOop](https://github.com/wigaandini/Tubes1_IF2210_OOop)

Pranala ke drawio: <https://drive.google.com/file/d/1-Dw1L2AnhFGMQmVtcUiJfJYb5VQ37hok/view?usp=sharing>

## Form asistensi:

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

No. Kelompok/Kode Kelompok	:	SMH
Nama Kelompok	:	OOp
Anggota Kelompok (Nama/NIM)	:	1. Erdianti Wiga / 13522053 2. Benardo / 13522055 3. William Glory / 13522113 4. Derwin Rustanly / 13522115 5. Mesach Harmasendro / 13522117 6. Renny Melanda / 10023575

## Asisten Pembimbing

Tanggal : 4 / 4 / 2024	Catatan Asistensi:
Tempat : Labpro	Jangan ada kelas kosong. Boleh aral dia ada alasan such as polymorphism. Buat store → bikin class game yg punya object store. Kalo mau store nya disimpan dr static, simpen di game controller (dibuat singeton) semua config simpan di object loadConfig di game controller blar nat di main class game yg jalan cm t. Tampilan cout di class masing2 aja, gaperlu bikin class view. Grid jd vector of vector Exception bikin turunan dr exception cpp nya
Kehadiran Anggota Kelompok:	
No NIM Tanda tangan	
1 13522053 	
2 13522055 	
3 13522113 	
4 13522115 	
5 13522117 	
6 10023575 	
	Tanda Tangan Asisten: 