

Kode Kelompok : C0K

Nama Kelompok : Debata Mangaramoti

1. 13522129 / Hugo Sabam Augusto

2. 13522136 / Muhammad Zaki

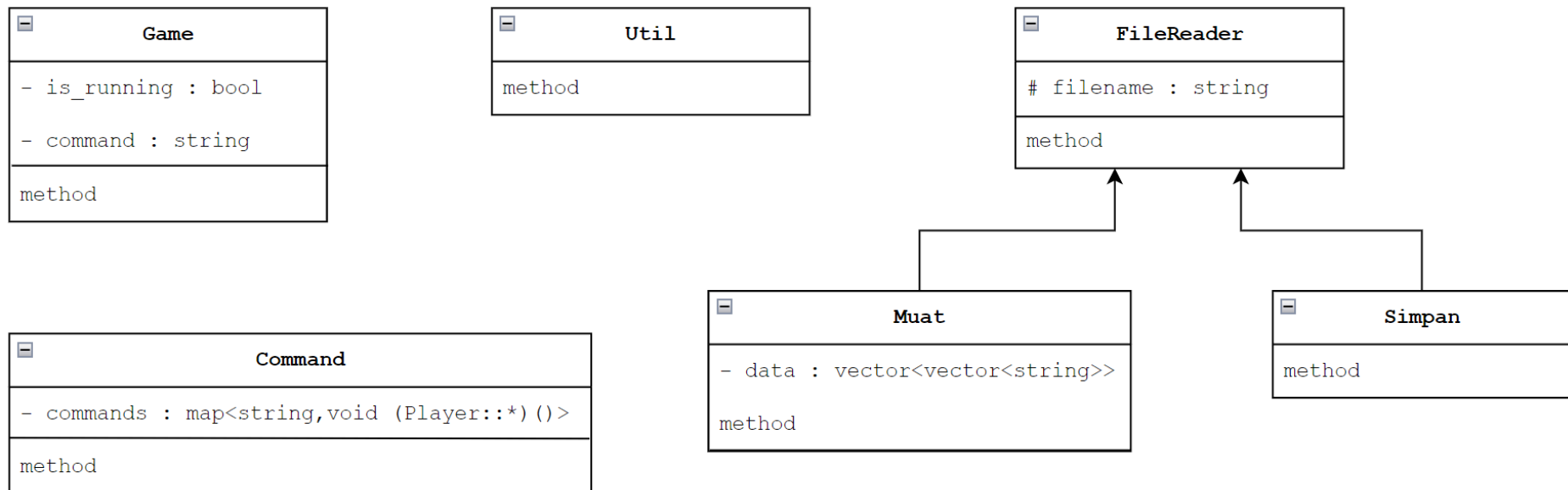
3. 13522137 / Ahmad Rafi Maliki

4. 13522144 / Nicholas Reymond Sihite

5. 13522149 / Muhammad Dzaki Arta

Asisten Pembimbing : Malik Akbar Hashemi Rafsanjani

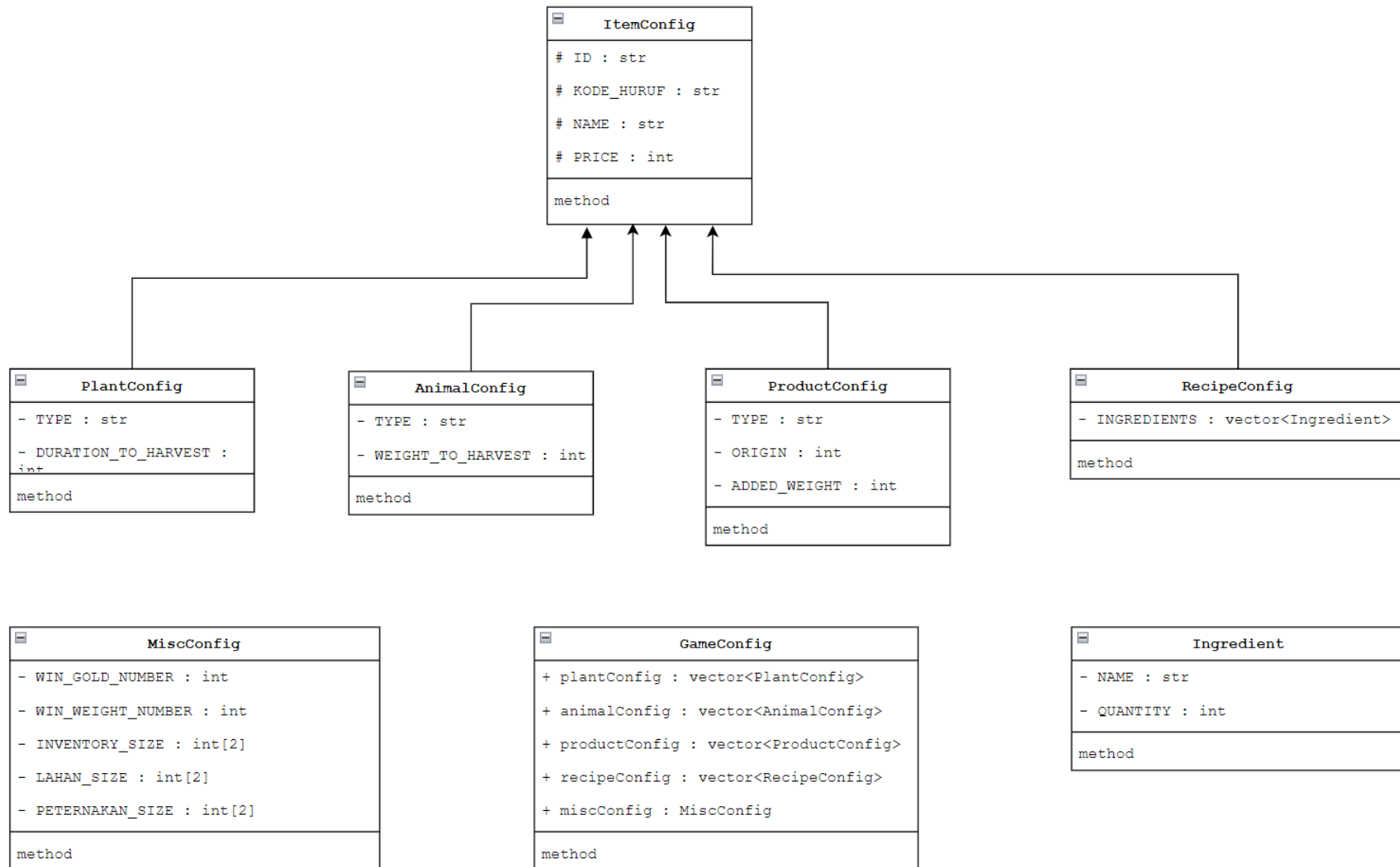
1. Diagram Kelas



Gambar 1.1 Diagram Kelas Game, Command, dan File

Tabel 1.1 Daftar Method Diagram Kelas 1.1

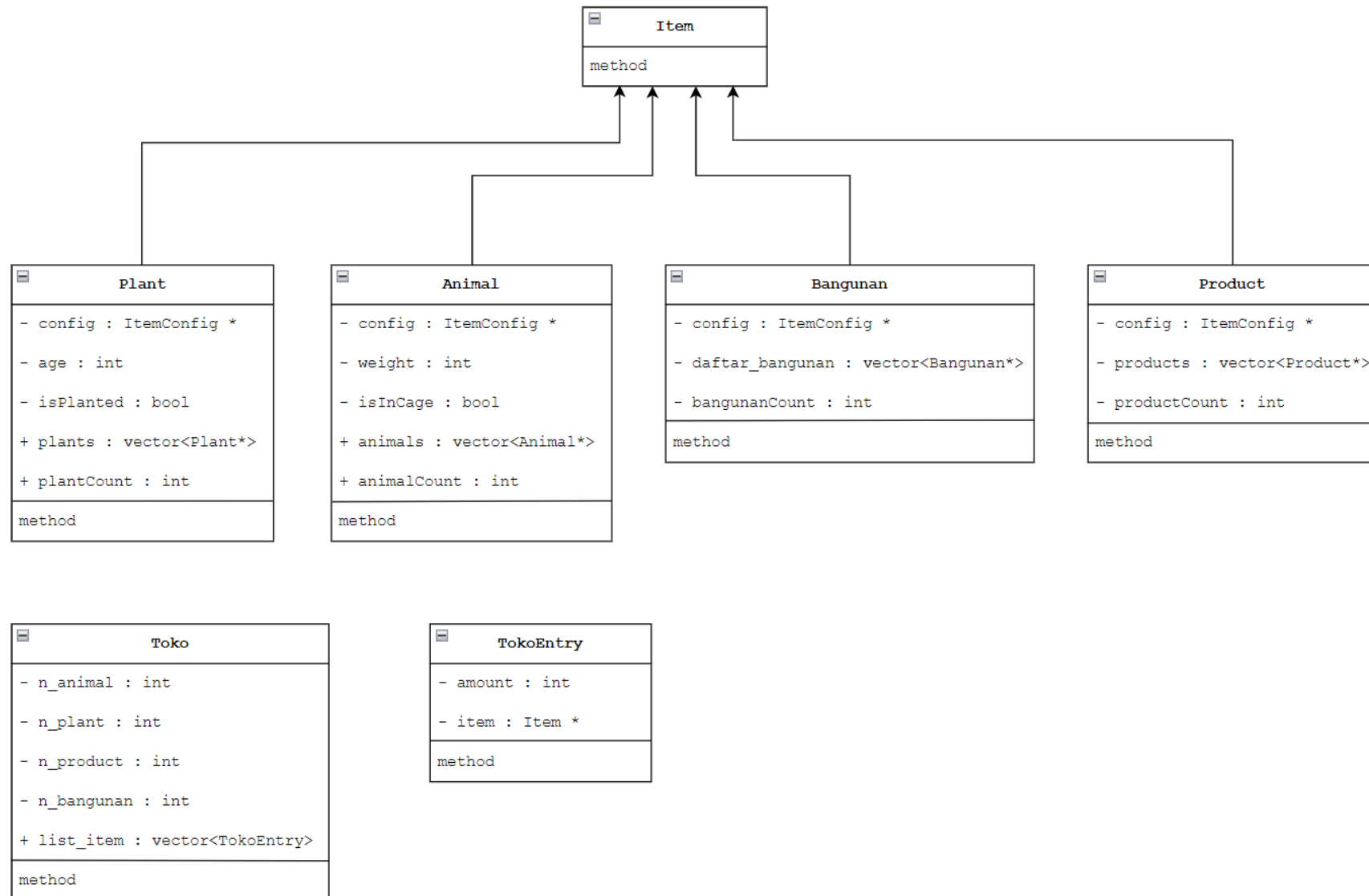
Kelas	Method
Game	Game(), ~Game(), start(), startNewGame(), inputCommand(), executeCommand(), loadGameData(), checkWin(), printWinner(), isRunning(), destroyObject()
Util	clearScreen(), printTitle(), split(), isValidChar(), isNumber(), isLetterLower(), isLetterUpper(), inputMultiplePetak(), isSameElement(), idxToInt(), IntToIdx(), stringToInt(), printColor(), printError()
Command	addCommand(), initCommand(), initCheat(), execute()
FileReader	FileReader(), readFile(), isValidName()
Muat	Muat(), composeName(), spawnPlayer(), spawnItem(), spawnPlant(), spawnAnimal(), tryReadFile(), loadSaveFile()
Simpan	Simpan(), convertLadangToString(), convertPeternakanToString(), convertPlayerToString(), convertTokoToString(), convertGameDataToString(), tryWriteFile(), writeToFile(), saveGame()



Gambar 1.2 Diagram Kelas Config

Tabel 1.2 Daftar Method Diagram Kelas 1.2

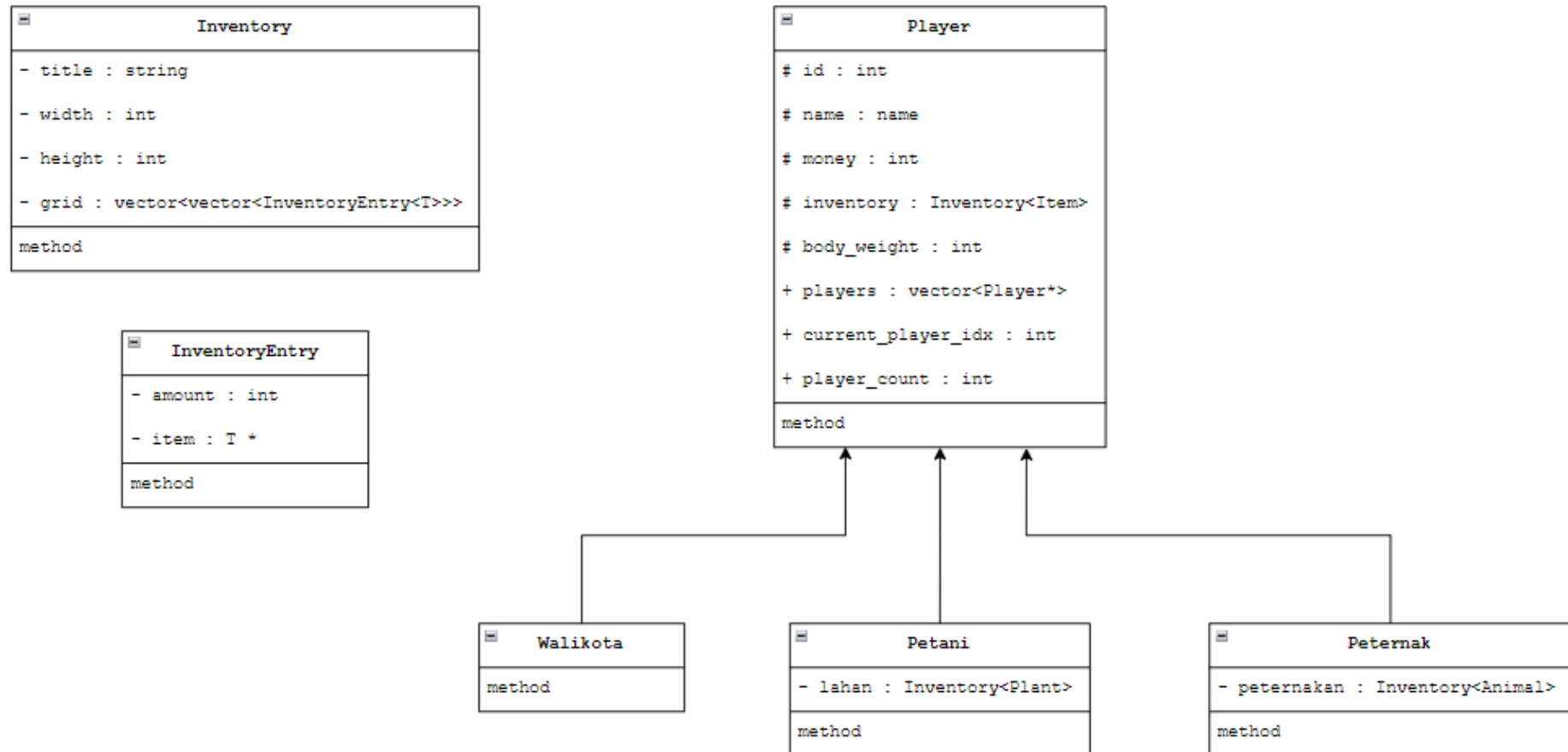
Kelas	Method
ItemConfig	ItemConfig(), print(), getID(), getKODE_HURUF(), getName(), getPrice(), getType(), getDURATION_TO_HARVEST(), getWEIGHT_TO_HARVEST(), getORIGIN(), getADDED_WEIGHT(), getINGREDIENTS()
PlantConfig	PlantConfig(), ReadPlantConfig(), print(), getType(), getDURATION_TO_HARVEST()
AnimalConfig	AnimalConfig(), ReadAnimalConfig(), print(), getType(), getWEIGHT_TO_HARVEST()
ProductConfig	ProductConfig(), ReadProductConfig(), print(), getType(), getORIGIN(), getADDED_WEIGHT()
RecipeConfig	RecipeConfig(), ReadRecipeConfig(), print(), getINGREDIENTS()
Ingredient	Ingredient(), print(), getName(), getQUANTITY()
MiscConfig	MiscConfig(), ReadMiscConfig(), print(), getWIN_GOLD_NUMBER(), getWIN_WEIGHT_NUMBER(), getINVENTORY_SIZE(), getLADANG_SIZE(), getPETERNAKAN_SIZE()
GameConfig	GameConfig(), loadGameConfig(), setAnimalConfig(), setPlantConfig(), setProductConfig(), setRecipeConfig(), setMiscConfig()



Gambar 1.3 Diagram Kelas Item dan Toko

Tabel 1.3 Daftar Method Diagram Kelas 1.3

Kelas	Method
Item	Item(), ~Item(), print(), getConfig(), setPlanted(), setIsInCage(), printKODE_HURUF(), isReadyToHarvest(), getID(), getKODE_HURUF(), getName(), getPrice(), getType(), getDURATION_TO_HARVEST(), getWEIGHT_TO_HARVEST(), getORIGIN(), getADDED_WEIGHT(), operator==, isFood(), isPlant() isAnimal(), getType_IDbyName()
Plant	Plant(), print(), getConfig(), setPlanted(), printKODE_HURUF(), isReadyToHarvest(), getType(), setAge(), getAge(), AgeAllPlants()
Animal	Animal(), print(), getConfig(), printKODE_HURUF(), isReadyToHarvest(), setIsInCage(), Feed(), getWeight(), setWeight()
Bangunan	Bangunan(), print(), getConfig(), printKODE_HURUF(), getType()
Product	Product(), print(), printKODE_HURUF(), getConfig(), getType(), getADDED_WEIGHT()
Toko	Toko(), displayToko(), addItem(), beliItemToko(), getItemToko(), getAmountEntry(), deleteItem(), checkValidItem(), isInfiniteItem() setItemEntryAmount(), getItemEntry(), checkEmptyItem()
TokoEntry	TokoEntry(), getItem(), getAmount(), tambahAmount(), setAmount()

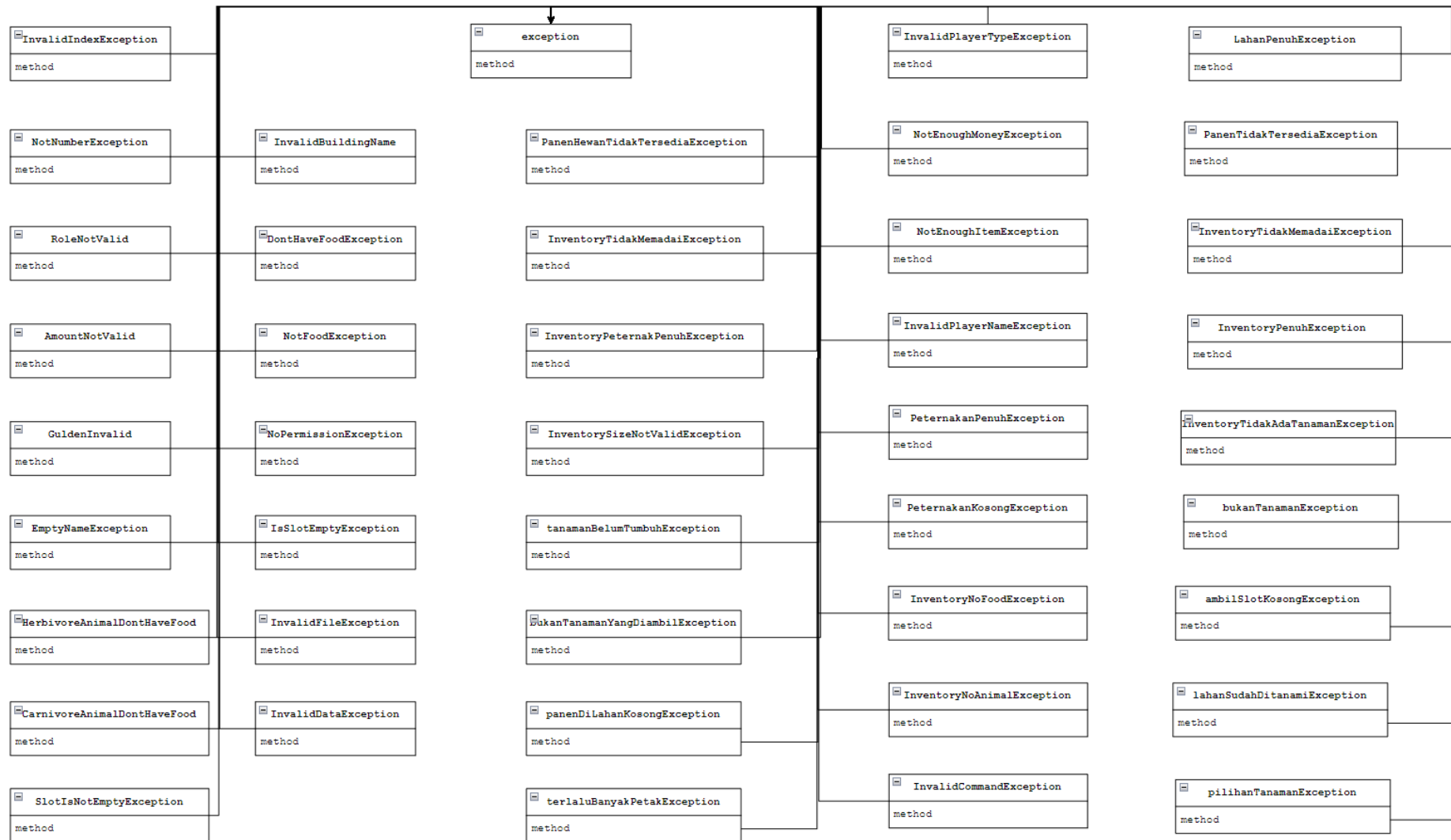


Gambar 1.4 Diagram Kelas Player dan Inventory

Tabel 1.4 Daftar Method Diagram Kelas 1.4

Kelas	Method
Player	Player(), ~Player(), printStats(), printInventory(), getCurrentPlayer(), printLahan(), printPternakan(), getInventory(), getLahan(), getPternakan(), addItem(), addPlant(), addAnimal(), addPlayer(), getWealth(), haveFood(), eat(), getType(), getMoney(), getWeight(), setMoney(), getName(), convertLowercase(), NEXT(), CETAK_PENYIMPANAN(), PUNGUT_PAJAK(), CETAK_LADANG(), CETAK_PETERNAKAN(), TANAM(), TERNAK(), BANGUN(), MAKAN(), KASIH_MAKAN(), BELI(), JUAL(), PANEN(), TAMBAH_PEMAIN(), operator>, operator<, SET(), GIVE(), STATS(), DELETE(), DELETE_INVENTORY(), DELETE_LADANG(), DELETE_PETERNAKAN()
Walikota	Walikota(), PUNGUT_PAJAK(), BANGUN(), TAMBAH_PEMAIN(), HITUNG_PAJAK(), getDaftarRecipe(), showRecipe(), inputNama(), getBangunanPrice(), useMoney(), cancelUseMoney(), useIngredients(), bangunBangunan(), inputJenisPemain(), inputPemainBaru(), addNewPlayer(), getType()
Petani	Petani(), printLahan(), addPlant(), getLadang(), getWealth(), CETAK_LADANG(), TANAM(), PANEN(), isLahanPenuh(), isPanenAvailable(), isInventoryMemadai(), isInventoryPenuh(), findHarvestablePlant(), cetakSiapPanen(), cetakHasilPanen(), isTanamanAda(), getPlantID(), tanamTanaman(), isPilihanTanamanValid(), panenTanaman(), tambahProduk(), isBanyakPetakValid(), ambilKodeTanaman(), getType()
Peternak	Peternak(), printPternakan(), addAnimal(), getPternakan(), getWealth(), CETAK_PETERNAKAN(), TERNAK(), KASIH_MAKAN(), PANEN(), isPternakanPenuh(), isPternakanKosong(), isFoodEmpty(), isAnimalEmpty(), isHarvestReady(),

	isInventoryMemadai(), isAnimalFoodEmpty(), isHerbivoreFoodReady(), isCarnivoreFoodReady(), getType(), pilihKandangMakan(), cekKondisiHewan(), checkAnimalFoodType(), checkFoodforAnimalBeforeLoop(), cekKondisiHewanSebelumLoop(), inputSlotTanahKasihMakan(), inputSlotInventoryKasihMakan(), inputSlotTanahTernak(), inputSlotInventoryTernak(), isInventoryPeternakPenuh(), findAnimaltoHarvest(), printAnimaltoHarvest(), chooseAnimaltoHarvest(), getKodeHewan(), inputPetakPanen(), panenHewan()
Inventory	Inventory(), getWidth(), getHeight(), getTitle(), add(), operator+=", remove(), print(), printItem(), isEmpty(), isFull(), calcEmptySpace(), count(), getItem()
InventoryEntry	InventoryEntry(), getAmount(), getItem()



Gambar 1.5 Diagram Kelas Exception

Program permainan yang kami buat dijalankan oleh kumpulan kelas pada gambar 1.1. Ketika memulai permainan, program akan meminta apakah pengguna ingin memuat *state* dari sebuah file .txt atau tidak. Jika ya, kelas 'Muat' akan membaca file tersebut dan mengintegrasikannya dengan program. Setelah itu, kelas 'Game' akan menginisiasi permainan dibantu oleh kelas 'Command' yang bertugas mengelola perintah masukan pengguna dan kelas 'Util' yang bertugas mengolah tampilan permainan.

Pengolahan konfigurasi permainan dilakukan oleh kumpulan kelas pada gambar 1.2. 'ItemConfig' adalah kelas *parent* dari 4 kelas *children*, yaitu 'PlantConfig', 'AnimalConfig', 'ProductConfig', dan 'RecipeConfig'. 'MiscConfig' tidak dimasukkan sebagai kelas *children* dari 'ItemConfig' karena menurut kami, informasi dari config tersebut tidak relevan untuk dijadikan item sehingga dibuat berdiri sendiri. Selain itu, terdapat kelas 'Ingredient' yang berfungsi menyimpan informasi resep bangunan pada 'RecipeConfig'. Kelas 'GameConfig' adalah kelas yang berfungsi menyimpan seluruh informasi konfigurasi permainan. Kelas ini dibuat untuk menyatukan berbagai informasi konfigurasi permainan agar tersentralisasi pada satu kelas.

Instansiasi objek-objek permainan (tumbuhan, hewan, bangunan, produk) dilakukan oleh kumpulan kelas pada gambar 1.3. Atribut pada kelas ini adalah pointer ke atribut masing-masing pada 'ItemConfig' beserta beberapa atribut tambahan yang diperlukan untuk keberjalanan permainan. Alasan kami membuat kumpulan kelas ini terpisah dengan 'ItemConfig' adalah untuk mengurangi kompleksitas ruang program. Dengan memanfaatkan atribut berupa pointer ke 'ItemConfig', setiap instansiasi kelas Item akan menunjuk pada atribut pada 'ItemConfig'. Hal ini tentunya lebih hemat penyimpanan dibanding apabila setiap objek baru dibuat, objek tersebut menyimpan informasi yang sebenarnya ada pada 'ItemConfig'. Selain itu, ada kelas 'Toko' yang berfungsi sebagai tempat dilakukannya transaksi semua 'Item' dan kelas 'ItemEntry' merupakan kelas yang digunakan untuk mewakili barang-barang yang ada pada 'Toko'.

Karakter-karakter dalam permainan digambarkan oleh kumpulan kelas pada gambar 1.4. Kelas *parent* 'Player' memiliki 3 kelas *children* yang masing-masing mewakili satu peran pemain, yaitu kelas 'Walikota', 'Petani', dan 'Peternak'. Setiap pemain

memiliki penyimpanan masing-masing yang diwakili oleh kelas 'Inventory' yang di dalamnya berisi barang-barang yang diwakili 'InventoryEntry'.

Penanganan kesalahan-kesalahan kondisi tertentu dilakukan oleh kumpulan kelas pada gambar 1.5. Satu kelas *children* dari kelas *parent* 'exception' digunakan untuk menangani satu kasus khusus. Oleh sebab itu, dibutuhkan banyak kelas *children* untuk menangani berbagai kesalahan dalam program secara keseluruhan.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

2.1.1. Player

Penerapan konsep inheritance pada kelas 'Player' dengan mewariskannya menjadi tiga kelas turunan, yaitu 'Walikota', 'Pemain', dan 'Peternak', adalah langkah yang bagus untuk mengatur hirarki objek dan mengelompokkan fungsionalitas yang berkaitan. Hal ini memungkinkan kami untuk menggunakan kelas dasar 'Player' sebagai kerangka umum dan mengkhususkan perilaku sesuai dengan peran masing-masing kelas turunan. Misalnya, pada kelas 'Player', metode 'getWealth()' bertanggung jawab untuk menghitung total kekayaan pemain, yang mencakup gulden dan nilai *inventory*. Namun, di kelas 'Petani' dan 'Peternak', tambahan pada kekayaan harus dipertimbangkan, yaitu nilai dari lahan dan peternakan.

Penggunaan konsep polymorphism sangat luas dalam kelas 'Player'. Sebagai contoh, kami mengimplementasikannya dalam metode getWealth() di kelas 'Player' sebagai gambaran dari implementasi polymorphism. Implementasi dari polymorphism memungkinkan kami untuk menggunakan metode yang sama

namun dengan perilaku yang berbeda tergantung pada jenis pemainnya. Ini membuat kode kami lebih fleksibel dan mudah diubah tanpa harus mengubah strukturnya.

Berikut cuplikan dari kode sumber kelas 'Player', 'Walikota', 'Pemain', dan 'Peternak':

player.hpp

```
class Player
{
protected:
    /* Attributes */

Public:
    /* Object Method */
    virtual int getWealth();
    ...

    /* Getter Setter */
    ...
    /* Game command related methods */
    ...

    /* Operator overloading */
    ...
};
```

petani.hpp

```
class Petani : public Player
{
private:
    Inventory<Plant> lahan;

public:
    /* Methods */
    int getWealth();
    ...
};
```

peternak.hpp

```
class Peternak : public Player
{
private:
    Inventory<Animal> peternakan;

public:
    /* Methods */
    int getWealth();
    ...
};
```

walikota.hpp

```
class Walikota : public Player {
public:
    /* Methods */
```

```
};
```

2.1.2. ItemConfig

Kelas 'PlantConfig', 'AnimalConfig', dan 'ProductConfig' semuanya mewarisi sifat-sifat dari kelas 'ItemConfig'. Dengan demikian, mereka memiliki akses ke atribut dan metode yang didefinisikan dalam kelas 'ItemConfig'. Implementasi polymorphism dilakukan melalui penggunaan metode virtual seperti `getType()`, `getDURATION_TO_HARVEST()`, `getINGREDIENTS()`, `getWEIGHT_TO_HARVEST()` dan `getADDED_WEIGHT()` yang dideklarasikan dalam kelas induk 'ItemConfig' dan diimplementasikan kembali dalam kelas turunannya seperti 'PlantConfig', 'ProductConfig', 'RecipeConfig', dan 'AnimalConfig'. Ini memungkinkan setiap kelas turunan untuk memberikan perilaku yang sesuai dengan tipe objeknya, yaitu menentukan tipe item, durasi panen untuk tanaman, berat panen untuk hewan, dst. Dengan pola ini, kami dapat memperlakukan objek dari kelas turunan secara seragam melalui kelas induknya sehingga meningkatkan fleksibilitas.

Berikut cuplikan dari kode sumber kelas 'ItemConfig', 'PlantConfig', 'AnimalConfig', dan 'ProductConfig':

item_config.hpp

```
class ItemConfig
{
protected:
    int ID;
    string KODE_HURUF;
```

```

    string NAME;
    int PRICE;

public:
    /* Methods */
    ...
    virtual string getType();
    virtual int getDURATION_TO_HARVEST();
    virtual int getWEIGHT_TO_HARVEST();
    virtual int getADDED_WEIGHT();
    virtual vector<Ingredient> getINGREDIENTS();
    ...
};

```

Kelas PlantConfig

```

class PlantConfig : public ItemConfig
{
private:
    string TYPE;
    int DURATION_TO_HARVEST;

public:
    /* Methods */
    int getDURATION_TO_HARVEST();
    ...
};

```


Kelas ProductConfig

```
class ProductConfig : public ItemConfig
{
private:
    string TYPE;
    string ORIGIN;
    int ADDED_WEIGHT;

public:
    /* Methods */

    string getTYPE();
    int getADDED_WEIGHT();
    ...
};
```

Kelas AnimalConfig

```
class AnimalConfig : public ItemConfig
{
private:
    string TYPE;
    int WEIGHT_TO_HARVEST;

public:
    /* Methods */

    string getTYPE();
```

```

    int getWEIGHT_TO_HARVEST();
    ...
};

Kelas RecipeConfig

class RecipeConfig : public ItemConfig
{
private:
    vector<Ingredient> INGREDIENTS;

public:
    /* Methods */
    vector<Ingredient> getINGREDIENTS();
    ...
};

```

2.1.3. Item

Penggunaan Inheritance dan Polimorfisme diimplementasikan pada kelas-kelas 'Product', 'Plant', 'Bangunan', dan 'Animal'. Kelas-kelas tersebut merupakan turunan dari kelas 'Item'. Setiap kelas turunan memiliki metode-metode virtual yang diwarisi dari kelas induk 'Item'. Implementasi dari getConfig() dilakukan di tiap kelas turunan 'Item' yang berfungsi mengambil pointer kelas 'ItemConfig' dengan pointer config yang sudah di implementasi pada constructor tiap kelas turunan.

Keuntungan dari penggunaan inheritance dan polymorphism dapat dilihat dalam penggunaan

metode-metode seperti `getDURATION_TO_HARVEST()`, `getType()`, `getWEIGHT_TO_HARVEST()` dan `getADDED_WEIGHT()`, `setPlanted()`, `setIsInCage()`, `isReadyToHarvest()`, dan metode-metode lain yang diwarisi dari kelas induk 'Item'. Dengan polymorphism, kita dapat menggunakan metode-metode ini pada objek kelas dasar Item tanpa perlu mengetahui jenis objek yang sebenarnya sehingga meningkatkan fleksibilitas dan kemudahan dalam mengelola dan memanipulasi objek-objek berbeda dalam program. Selain itu, penggunaan inheritance properti dan perilaku yang umum untuk semua jenis objek diwarisi dari kelas dasar mengurangi duplikasi kode dan meningkatkan efisiensi.

Berikut cuplikan dari kode sumber kelas 'Item', 'Product', 'Plant', 'Bangunan', dan 'Animal':

item.hpp

```
class Item
{
private:
public:
    /* Methods */
    ...
    virtual ItemConfig *getConfig() = 0;
    virtual void setPlanted(bool isPlanted);
    virtual void setIsInCage(bool isInCage);
    virtual void printKODE_HURUF();
    virtual bool isReadyToHarvest();
    ...
    /* Getter */
    ...
    virtual int getDURATION_TO_HARVEST();
```

```

    virtual int getWEIGHT_TO_HARVEST();
    virtual int getADDED_WEIGHT();
    ...
};

```

animal.hpp

```

class Animal : public Item
{
private:
    /* Attributes */
    ItemConfig *config;
    int weight;
    bool isInCage;
Public:
    /* Methods */
    ...
    ItemConfig *getConfig();
    bool isReadyToHarvest();
    void setIsInCage(bool isInCage);
    ...
};

```

bangunan.hpp

```

class Bangunan : public Item
{
private:

```

```

        ItemConfig *config;

Public:
    /* Methods */
    ...
    ItemConfig *getConfig();
    ...

};

```

product.hpp

```

class Product : public Item
{
private:
    ItemConfig *config;

public:
    /* Methods */
    ...
    ItemConfig *getConfig();
    string getTYPE();
    int getADDED_WEIGHT();
    ...

};

```

plant.hpp

```

class Plant : public Item
{

```

```
private:
    /* Attributes */
    ItemConfig *config;
    int Age;
    bool isPlanted;

Public:
    /* Methods */
    ...
    ItemConfig *getConfig();
    void setPlanted(bool isPlanted);
    bool isReadyToHarvest();
    string getTYPE();
    ...

};
```

2.1.4. FileReader

Untuk kelas 'FileReader', kami hanya mengimplementasikan inheritance saja, tidak ada polymorphism. Inheritance pada kelas ini digunakan untuk mewariskan method `readFile()` untuk kedua kelas turunannya. Pendekatan ini dilakukan untuk memastikan bahwa implementasi metode tersebut tidak perlu dibuat/diulang di kedua kelas turunan. Dengan ini, kelas 'Muat' dan 'Simpan' memiliki kemampuan membaca file yang nantinya dapat digunakan untuk menyimpan dan memuat data yang tertulis di file.

```
file_reader.hpp
```

```

class FileReader {
protected:
    string filename;
public:
    // constructor
    FileReader(string filename);

    // method
    vector<vector<string>> readFile();
    bool isValidName();
};

```

muat.hpp

```

class Muat : public FileReader
{
private:
    vector<vector<string>> data;

    void spawnObject();
    static string composeName(vector<string> vec);
    static Player *spawnPlayer(vector<string> vec);
    static Item *spawnItem(string name);
    static Plant *spawnPlant(string name, int age);
    static Animal *spawnAnimal(string name, int weight);

public:

```

```
Muat(string filename);  
  
bool tryReadFile();  
  
static void loadSaveFile();  
};
```

simpan.hpp

```
class Simpan : public FileReader  
{  
private:  
    // Convert object to string method  
    static string convertLadangToString(Inventory<Plant> *ladang);  
    static string convertPternakanToString(Inventory<Animal> *peternakan);  
    static string convertPlayerToString(Player *player);  
    static string convertTokoToString();  
    static string convertGameDataToString();  
  
public:  
  
    Simpan(string filename);  
  
    bool tryWriteFile();  
    void writeToFile(string data);  
  
    static void saveGame();  
};
```


2.2. Method/Operator Overloading

2.2.1. Operator+

Operator+ diimplementasikan pada kelas 'Inventory'. Operator ini membantu saat ada penambahan Item pada kelas Inventory tersebut. Contoh penggunaan bisa dilihat dari potongan kode di bawah ini:

```
/* Implementasi operator+ pada kelas Inventory */
template <class T>
void Inventory<T>::operator+=(T *item)
{
    add(item);
}

/* Potongan kode dari method spawnObject() pada kelas Muat */
void Muat::spawnObject(){

    ...
    // Spawning player inventory
    for (int j = 0; j < num_inventory; j++){
        Item* item = Muat::spawnItem(this->data[row][0]); row++;
        (*inventory) += item;
    }
    ...
}
```

2.2.2. Operator< dan Operator>

Operator< diimplementasikan pada kelas 'Player'. Operator ini membantu saat ada pengecekan urutan pemain yang nantinya akan digunakan dalam penentuan urutan secara leksikografis. Contoh penggunaan bisa dilihat dari potongan kode di bawah ini:

```
/* Implementasi operator< pada kelas Player*/
bool Player::operator<(Player* other){
    string currName = name;
    currName = convertLowercase(currName);
    string otherName = other->getName();
    otherName = convertLowercase(otherName);
    if (currName < otherName){
        return true;
    } else {
        return false;
    }
}
```

```
/* Implementasi operator> pada kelas Player*/
bool Player::operator>(Player* other){
    return !(*this < other);
}
```

```
/* Potongan kode dari method addPlayer pada kelas Player */
void Player::addPlayer(Player *player){
    bool added = false, beforeCurr = false;
    vector<Player*> newPlayerVec;
```

```

    for (int i = 0; i < players.size(); i++){
        if ((*player < players[i]) && (!added)){
            newPlayerVec.push_back(player);
            added = true;
        }
        newPlayerVec.push_back(players[i]);
    }
    if (!added){
        newPlayerVec.push_back(player);
    }
    players = newPlayerVec;
}

```

2.2.3. Operator==

Operator== diimplementasikan di kelas 'Item'. Operator ini membantu saat kasus kelas 'Player' melakukan method JUAL(). Operator ini digunakan untuk mencocokkan Item apabila Item yang dijual oleh kelas 'Player' sudah tersedia di kelas 'Toko'. Contoh penggunaan operator ini bisa dilihat dari potongan kode di bawah:

```

/* Implementasi operator== pada kelas Item */
bool Item::operator==(Item* item)
{
    return getConfig()->getKODE_HURUF() == item->getConfig()->getKODE_HURUF();
}

```

```
/* Potongan kode dari method addItem() pada kelas Toko */  
void Toko::addItem(Item *item)  
{  
    ...  
    if (*list_item[i].getItem() == item)  
    {  
        list_item[i].tambahAmount(1);  
    }  
    ...  
}
```

2.3. Template & Generic Classes

Contoh penerapan konsep template & generic classes terdapat pada kelas Inventory. Dengan adanya template kelas generik, kami tidak perlu mendefinisikan kembali kelas inventory ketika ada tipe elemen yang dibutuhkan oleh kelas yang berbeda untuk kebutuhan yang serupa. Sebagai contoh, kami menerapkannya pada kelas 'Petani' yang mempunyai lahan. Kami tidak perlu mendefinisikan kembali kelas 'Inventory', tetapi cukup menggantinya dengan tipe elemen yang sesuai yaitu 'Plant'. Keunggulan lainnya adalah kami dapat melakukan abstraksi terhadap atribut dan metode yang dimiliki oleh kelas generik tersebut dengan menggunakan tipe yang sudah disediakan. Penggunaan abstraksi ini dapat meningkatkan modularitas dan memungkinkan penggunaan kelas tersebut dalam berbagai konteks tanpa perlu memodifikasi kode secara signifikan. Konsep ini sesuai dengan kasus inventory karena dalam permainan ini, terdapat berbagai tipe elemen yang dibutuhkan sesuai dengan kelas contohnya Inventory<Plant> untuk 'Petani', Inventory<Animal> untuk 'Peternak', dan Inventory<Item> untuk kelas 'Player'.

inventory.hpp

```
template <class T>
class InventoryEntry
{
private:
    int amount; /* hanya 0 dan 1, ada dan tidak, gbs multiple*/
    T *item;

public:
    InventoryEntry();
    InventoryEntry(T *item);

    int getAmount();
    T *getItem();
};

template <class T>
class Inventory
{
private:
    /* data */
    string title;
    int width;
    int height;
    vector<vector<InventoryEntry<T>>> grid;
public:

    Inventory();
```

```
Inventory(int width, int height, string title);

int getWidth();
int getHeight();
string getTitle();

void add(T *item);
void add (T *item, int i, int j);
void add(T *item, string idx);

void remove(int i, int j);
void remove(string idx);

void print();
void printItem(int i, int j);

bool isEmpty(int j, int i);
bool isEmpty(string idx);
bool isFull();

int calcEmptySpace();
int count();

T *getItem(int i, int j);
T *getItem(string idx);

};
```

petani.hpp

```
class Petani : public Player
{
private:
    Inventory<Plant> lahan;

Public:
    .....
    .....
}
```

2.4. Exception

Dalam C++, Exception merupakan sebuah respons ketika ada masalah yang terjadi dalam eksekusi program. Komputer akan membuat sebuah exception dan jika kode tidak mempunyai cara untuk mengatasinya, program tersebut akan berhenti beroperasi. Exception pada C++ terdiri dari 3 kata kunci, yaitu: try, catch, dan throw. Try digunakan untuk mencoba kode yang digunakan apakah terjadi error ketika dieksekusi. Selanjutnya throw, yaitu program akan 'melempar' sebuah exception ketika ada problem yang terjadi, di sini kita bisa melakukan kustomisasi error yang terjadi. Terakhir ada catch, catch akan 'menangkap' exception yang telah di throw dan selanjutnya akan mengeksekusi potongan kode yang ada pada blok catch. Konsep exception ini hampir digunakan pada seluruh kelas karena banyaknya kasus yang perlu ditangani.

2.4.1. Contoh Exception pada Class Peternak

Exception untuk kelas 'Peternak' meliputi beberapa hal yaitu sebagai berikut.

PeternakanPenuhException, exception ini digunakan ketika peternakan sudah penuh dan pengguna ingin ternak hewan atau memasukkan hewan ke peternakan. Exception ini akan memberikan pesan error kepada pengguna berupa "Peternakan penuh". Exception ini memudahkan pengguna karena dapat membantu mengeluarkan program dari infinite loop yang mungkin terjadi pada fungsi TERNAK.

PeternakanKosongException, exception ini digunakan ketika peternakan kosong dan pengguna ingin memanen hewan atau memberi makan hewan. Exception ini akan memudahkan pengguna karena menghindarkan pengguna dari infinite loop yang akan terjadi pada fungsi KASIH_MAKAN.

InventoryNoFoodException, exception ini digunakan ketika pengguna ingin memberi makan hewan, tetapi tidak ada makanan di inventory.

InventoryNoAnimalException, exception ini digunakan ketika pengguna ingin ternak hewan, tetapi tidak ada hewan di inventory.

HerbivoreAnimalDontHaveFood, exception ini digunakan ketika pengguna memanggil fungsi KASIH_MAKAN, tetapi di peternakan hanya ada hewan herbivora sedangkan pengguna hanya memiliki makanan untuk hewan karnivora.

CarnivoreAnimalDontHaveFood, exception ini digunakan ketika pengguna memanggil fungsi KASIH_MAKAN, tetapi di peternakan hanya ada hewan karnivora sedangkan pengguna hanya memiliki makanan untuk hewan herbivora.

PanenHewanTidakTersediaException, exception ini digunakan ketika pengguna memanggil fungsi PANEN, tetapi belum ada hewan yang siap di panen di peternakan.

InventoryTidakMemadaiException, exception ini digunakan ketika pengguna memanggil fungsi PANEN, tetapi slot yang ada di inventory pengguna tidak cukup untuk menampung hasil panen.

InventoryPeternakPenuhException, exception ini digunakan ketika pengguna memanggil fungsi PANEN, tetapi inventory pengguna sedang penuh.

peternakException.hpp

```
class PeternakanPenuhException : public exception {
public:
    const char* what() const noexcept override {
        return "Peternakan penuh";
    }
};

class PeternakanKosongException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Peternakan kosong";
    }
}
```

```
};

class InventoryNoFoodException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Inventory tidak memiliki makanan";
    }
};

class InventoryNoAnimalException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Inventory tidak memiliki hewan";
    }
};

// peternakan cuma punya hewan herbivora sedangkan makanan yang ada adalah makanan
karnivora
class HerbivoreAnimalDontHaveFood : public std::exception {
public:
    const char* what() const noexcept override {
        return "Peternakan hanya memiliki hewan herbivora sedangkan makanan yang ada
adalah makanan karnivora";
    }
};

class CarnivoreAnimalDontHaveFood : public std::exception {
public:
```

```
        const char* what() const noexcept override {  
            return "Peternakan hanya memiliki hewan karnivora sedangkan makanan yang ada  
adalah makanan herbivora";  
        }  
};  
  
class PanenHewanTidakTersediaException : public std::exception {  
public:  
    const char* what() const noexcept override {  
        return "Tidak ada hewan yang siap dipanen";  
    }  
};  
  
class InventoryTidakMemadaiException : public std::exception {  
public:  
    const char* what() const noexcept override {  
        return "Inventory tidak memadai";  
    }  
};  
  
class InventoryPeternakPenuhException : public std::exception {  
public:  
    const char* what() const noexcept override {  
        return "Inventory peternak penuh";  
    }  
};
```

2.5. C++ Standard Template Library

C++ standard library adalah kumpulan struktur data dan algoritma yang disediakan sebagai bagian standar C++ untuk menyediakan solusi yang efisien dan fleksibel dalam pemrograman tanpa harus mengimplementasikannya terlebih dahulu untuk digunakan. C++ Standard Template Library cukup banyak digunakan dalam kelas kelas yang kami gunakan. Salah satu STL yang cukup banyak kami gunakan adalah <vector> karena kemudahan penggunaannya dan fleksibilitas ukuran. Contoh kasusnya adalah pada kelas muat, di situ kami menggunakan vector of vector karena kemudahan penggunaannya dan juga ukurannya yang dinamis cocok digunakan untuk memuat data yang belum tentu pasti ukurannya. Selanjutnya, contoh penggunaan library lainnya adalah <iostream> yang kami gunakan untuk input dan output. Selain itu, kelompok kami juga menggunakan <string> karena ada beberapa atribut yang disimpan dengan tipe string.

muat.hpp

```
class Muat : public FileReader {  
  
    private:  
        // Data attribute  
        vector<vector<string>> data;  
  
        // Object spawning method  
        void spawnObject();  
        static string composeName(vector<string> vec);  
        static Player* spawnPlayer(vector<string> vec);  
        static Item* spawnItem(string name);  
        static Plant* spawnPlant(string name, int age);  
};
```

```
        static Animal* spawnAnimal(string name, int weight);

public:
    // Constructor
    Muat(string filename);

    // File reading method
    bool tryReadFile();

    // Controller method
    static void loadSaveFile();

};
```

item_config.hpp

```
class Ingredient {
private:
    string NAME;
    int QUANTITY;

public:
    Ingredient(string NAME, int QUANTITY);
    void print();

    string getName();
    int getQUANTITY();
};
```

```
};
```

2.6. Abstract Class

Sebuah kelas bilang sebuah *abstract class* apabila setidaknya memiliki satu fungsi virtual murni. Dengan kata lain, sebuah fungsi yang tidak memiliki definisi. Para turunan dari kelas abstrak harus mendefinisikan fungsi virtual murni tersebut. Sebagai contoh, konsep *abstract base class* diterapkan pada kelas 'Player' yang di-inherit kepada kelas 'Petani', 'Peternak', dan juga 'Walikota'. Konsep ini cocok digunakan pada kasus ini dikarenakan memungkinkan kami untuk menggunakan kelas dasar 'Player' sebagai kerangka umum dan mengkhususkan perilaku sesuai dengan peran masing-masing kelas turunan. Misalnya, pada kelas 'Player', metode '~Player()' merupakan pure virtual yang digunakan sebagai destructor

player.hpp

```
class Player
{
protected:
    int id;
    string name;
    int money;
    Inventory<Item> inventory;
    int body_weight;

public:
    static vector<Player *> players;
```

```
static int current_player_idx;
static int player_count;

Player(string name, int money, int body_weight);
virtual ~Player() = 0;

void printStats();
void printInventory();
static Player *getCurrentPlayer();

virtual void printLahan();
virtual void printPeternakan();
Inventory<Item> *getInventory();
virtual Inventory<Plant> *getLadang();
virtual Inventory<Animal> *getPeternakan();

void addItem(Item *item);
virtual void addPlant(Plant *item);
virtual void addAnimal(Animal *item);
void addPlayer(Player *player);
virtual int getWealth();

bool haveFood();
void eat(Item *food);

virtual string getType();
int getMoney();
int getWeight();
void setMoney(int money);
```

```
string getName();
string convertLowercase(string str);

/* Game command related methods */
virtual void NEXT();
virtual void CETAK_PENYIMPANAN();
virtual void PUNGUT_PAJAK();
virtual void CETAK_LADANG();
virtual void CETAK_PETERNAKAN();
virtual void TANAM();
virtual void TERNAK();
virtual void BANGUN();
virtual void MAKAN();
virtual void KASIH_MAKAN();
virtual void BELI();
virtual void JUAL();
virtual void PANEN();
virtual void TAMBAH_PEMAIN();

/* Operator overloading */
bool operator<(Player* other);
bool operator>(Player* other);

/* Cheat Commands */
void SET();
void GIVE();
void STATS();
void DELETE();
void DELETE_INVENTORY();
```



```

void DELETE_LADANG();
void DELETE_PETERNAKAN();

};

```

3. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Next	13522137	13522137
Cetak Penyimpanan	13522137	13522137
Pungut Pajak	13522144	13522144
Cetak Ladang	13522136	13522136
Cetak Peternakan	13522149	13522149
Tanam	13522136	13522136
Ternak	13522149	13522149
Bangun Bangunan	13522144	13522144
Makan	13522137	13522137
Memberi Pangan	13522149	13522149
Membeli	13522129	13522129






Menjual	13522129	13522129
Memanen	13522136	13522136
Muat	13522137	13522137
Simpan	13522137	13522137
Tambah Pemain	13522144	13522144

4. Lampiran

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

No. Kelompok/Kode Kelompok : C0K
 Nama Kelompok : Debata Mangaramoti
 Anggota Kelompok (Nama/NIM) :
 1. Hugo Sabam Augusto / 13522129
 2. Muhammad Zaki / 13522136
 3. Ahmad Rafi Maliki / 13522137
 4. Nicholas Reymond Sihite / 13522144
 5. Muhammad Dzaki Arta / 13522149
 Asisten Pembimbing : Malik Akbar Hashemi Rafsanjani

Tanggal : 5 April 2024	Catatan Asistensi:
Tempat : Google Meet	<p>Apakah harus membuat <i>game manager</i> atau langsung di <i>loop</i> di <i>main</i>?</p> <ul style="list-style-type: none"> - Lebih baik membuat <i>game manager</i> <p>Lebih baik implemen di .hpp atau di .cpp ?</p> <ul style="list-style-type: none"> - Lebih baik implemennya di .cpp (<i>best practice</i>) <p>Apakah pemain baru langsung masuk <i>next turn</i>?</p> <ul style="list-style-type: none"> - Giliran pemain berdasarkan <i>priority queue</i>, termasuk pemain yang baru dibuat <p>TIPS :</p> <ul style="list-style-type: none"> - Lakukan validasi setiap kondisi yang mungkin - Masing masing pada setiap kelas yang dibuat make konsep SOLID - Sebaiknya gunakan linux atau WSL untuk <i>develop</i> dari awal untuk menghindari masalah yang tidak perlu - Sebaiknya gunakan konsep OOP secara keseluruhan dalam <i>develop</i> program - Contoh penggunaan map adalah pencocokan <i>string (input)</i> dan <i>command</i> terkait - <i>Game manager</i> cukup menyimpan tipe pemain, bukan mengatur bagaimana perilaku pemain

<p>Kehadiran Anggota Kelompok:</p> <p>No NIM Tanda tangan</p> <p>1 13522129 </p> <p>2 13522136 </p> <p>3 13522137 </p> <p>4 13522144 </p> <p>5 13522149 </p>	
	<p>Tanda Tangan Asisten:</p> 