LAPORAN TUGAS BESAR I
IF2210 Pemrograman Berorientasi Objek

“Kelola Kerajaan Bersama Labpro”

Dipersiapkan oleh Kelompok OOPsie (SHT):
13522012 / Thea Josephine Halim
13522040 / Dhidit Abdi Aziz
13522046 / Raffael Boymian Siahaan
13522096 / Novelya Putri Ramadhani
13522104 / Diana Tri Handayani

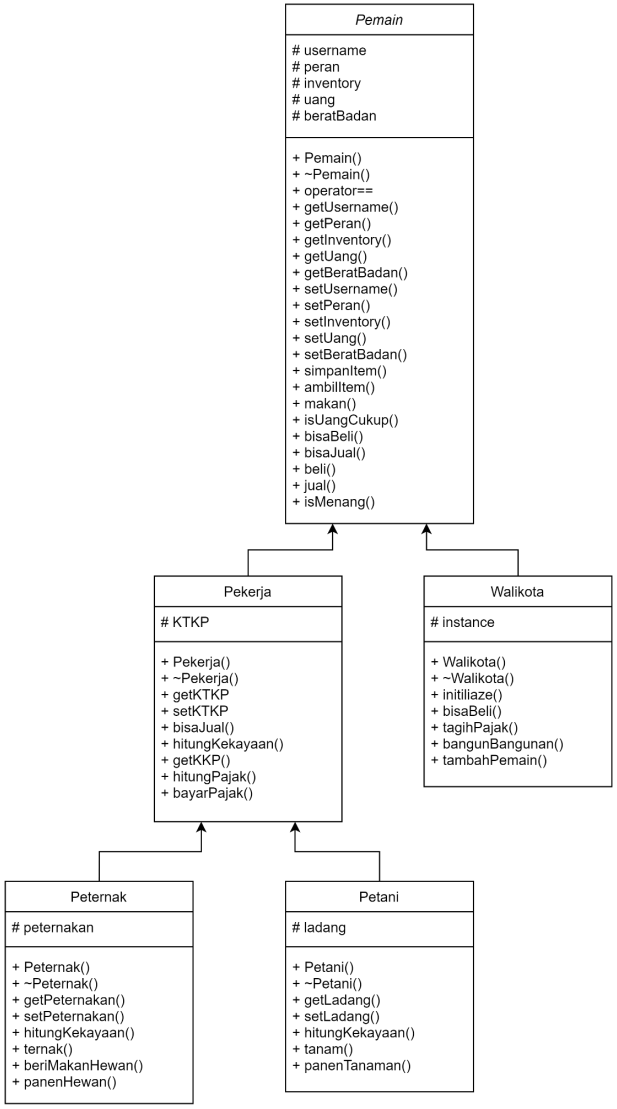
Asisten Pembimbing: M Syahrul Surya Putra

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

DAFTAR ISI

1. Diagram Kelas

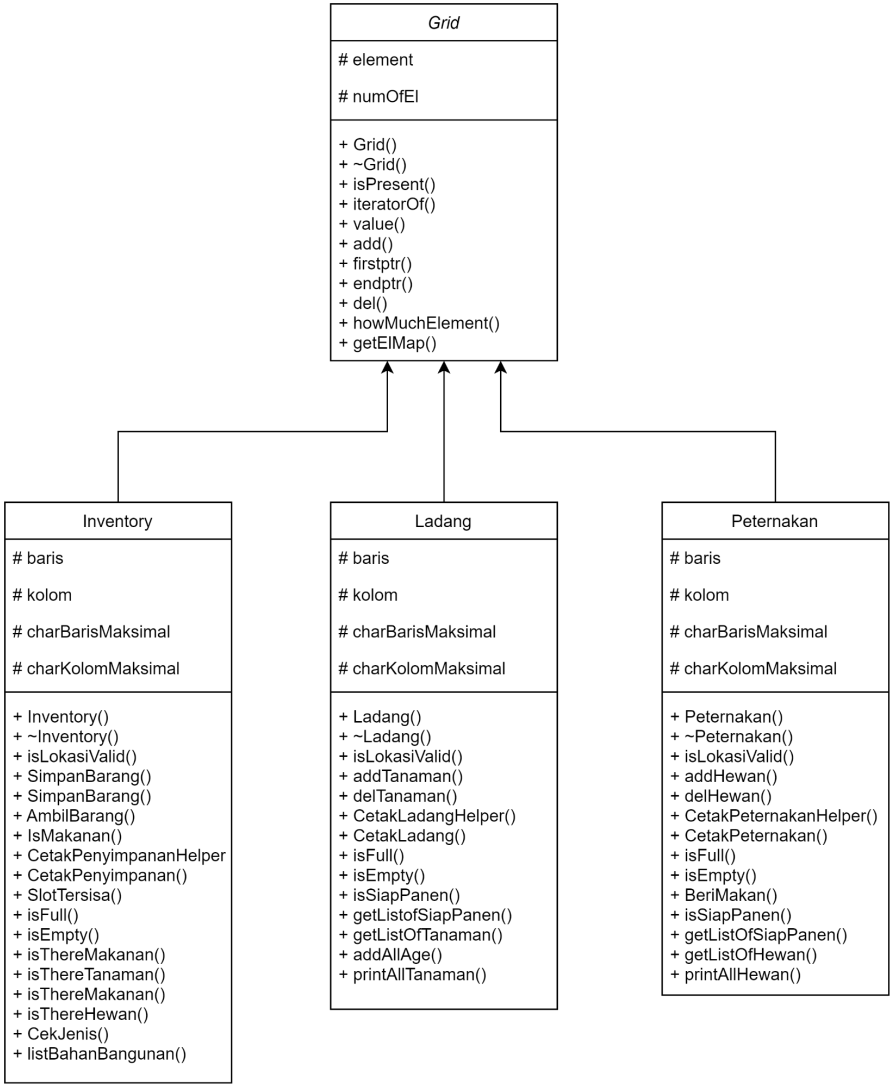
1.1 Kelas Pemain, Pekerja, Walikota, Peternak dan Petani



Kelas Pemain menerapkan konsep inheritance dan polymorphism. Kelas Pemain memiliki 2 kelas turunan: Pekerja dan Walikota, sedangkan kelas Pekerja itu sendiri memiliki 2 turunan kelas: Petani dan Peternak. Hal tersebut dilakukan karena kelas Pekerja dan Pemain memiliki atribut dan sifat yang sama dengan pemain, tetapi memiliki atribut dan sifat tambahan yang berbeda. Berlaku pula pada kelas turunan Pekerja yaitu kelas Peternak dan Petani.

Kelebihan dari menggunakan konsep inheritance dan polymorphism pada kelas-kelas ini adalah pembuatan kelas yang serupa tidak perlu membuat lagi dari awal, serta mengurangi penulisan kode yang redundant atau berulang-ulang. Namun, tidak dapat dipungkiri juga terdapat kekurangan, adanya ketergantungan antar kelas yang terkadang membuat lebih sulit dalam debugging.

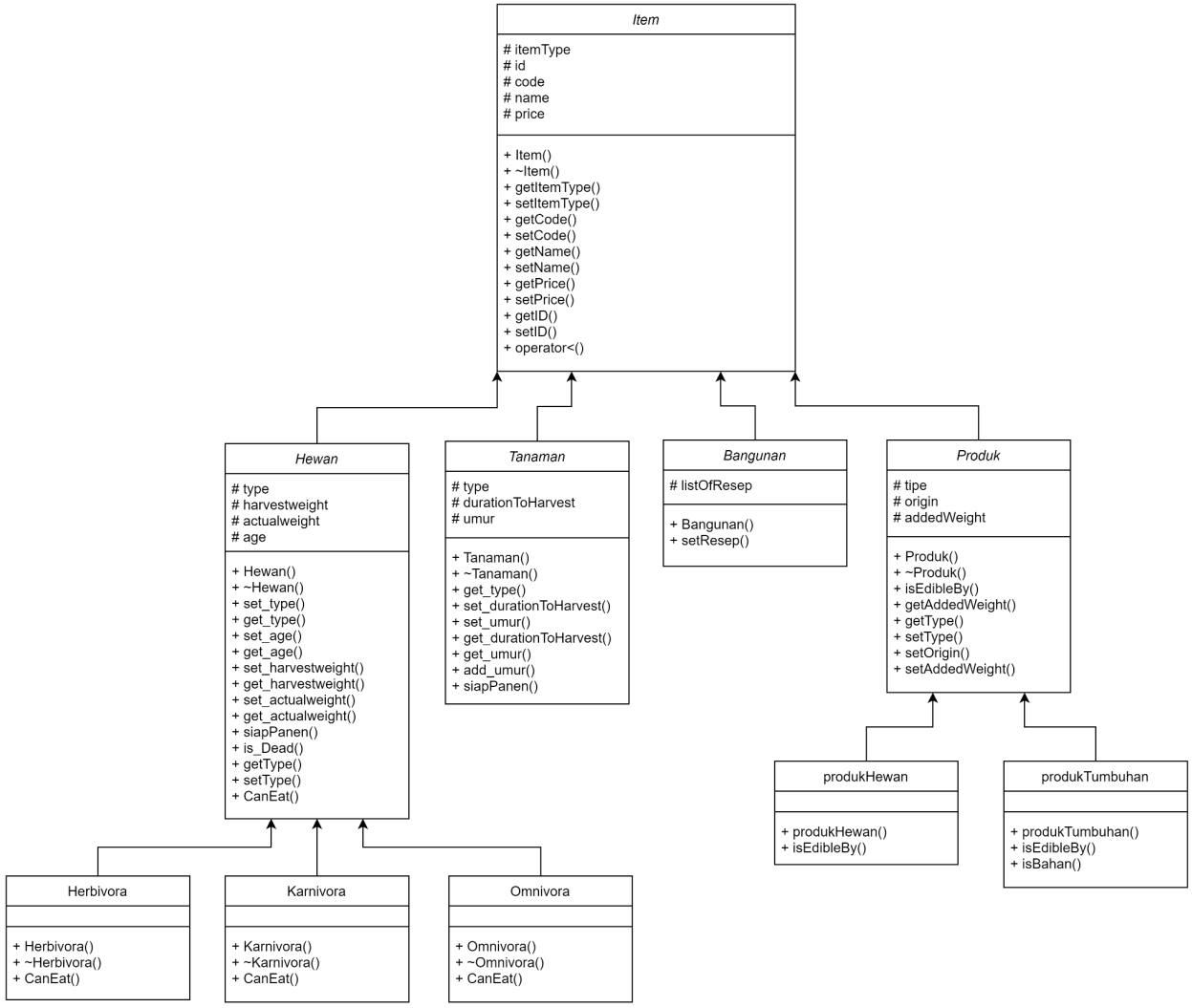
1.2 Kelas Grid, Inventory, Ladang, dan Peternakan



Kelas Grid menerapkan konsep inheritance dan polymorphism. Kelas Grid memiliki 3 kelas turunan: Inventory, Ladang, dan Peternakan. Hal tersebut dilakukan karena kelas Inventory, Ladang, dan Peternakan memiliki atribut dan sifat yang sama dengan Grid, tetapi memiliki sifat tambahan yang berbeda.

Kelebihan dari menggunakan konsep inheritance dan polymorphism pada kelas-kelas ini adalah pembuatan kelas yang serupa tidak perlu membuat lagi dari awal, serta mengurangi penulisan kode yang redundant atau berulang-ulang. Namun, tidak dapat dipungkiri juga terdapat kekurangan, adanya ketergantungan antar kelas yang terkadang membuat lebih sulit dalam debugging.

1.3 Kelas Item, Tanaman, Hewan, Bangunan, Produk, Herbivora, Karnivora, Omnivora, produkHewan, dan produkTumbuhan



Kelas Item menerapkan konsep inheritance dan polymorphism. Kelas Item memiliki 4 kelas turunan: Hewan, Tanaman, Bangunan, dan Produk. Selanjutnya kelas Hewan dan Produk menerapkan konsep Abstract Base Class, inheritance, dan polymorphism. Kelas Hewan memiliki turunan Herbivora, Karnivora, dan Omnivora. Sedangkan kelas Produk memiliki turunan produkHewan dan produkTumbuhan. Hal tersebut dilakukan karena kelas Hewan, Tanaman, Bangunan, dan Produk memiliki atribut dan sifat yang sama dengan Item yaitu akan disimpan dalam tempat yang sama, Inventory. Tetapi keempat kelas tersebut tetap memiliki atribut dan sifat tambahan yang berbeda. Berlaku pula pada kelas turunan Hewan dan Produk yaitu kelas Herbivora, Karnivora, Omnivora, produkHewan dan produkTumbuhan.

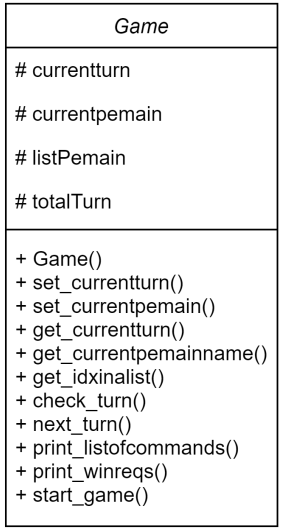
Kelebihan dari menggunakan konsep Abstract Base Class, inheritance, dan polymorphism pada kelas-kelas ini adalah pembuatan kelas yang serupa tidak perlu membuat lagi dari awal, serta mengurangi penulisan kode yang redundant atau berulang-ulang. Namun, tidak dapat dipungkiri juga terdapat kekurangan, adanya ketergantungan antar kelas yang terkadang membuat lebih sulit dalam debugging.

1.4 Kelas Toko

Kelas Toko tidak menerapkan konsep Abstract Base Class, inheritance, maupun polymorphism atau bisa disebut berdiri sendiri. Tetapi tidak berarti kelas ini tidak memiliki keterhubungan dengan kelas lain. Dalam implementasinya kelas ini membutuhkan kelas lainnya seperti kelas Pemain, Item beserta turunan-turunannya.

Kelebihan dari pemakaian konsep ini ialah ketika terjadi pengubahan atribut maupun method dalam kelas ini tidak akan memengaruhi kelas lain secara langsung. Kecuali kelas lain memiliki ketergantungan dengan kelas Toko.

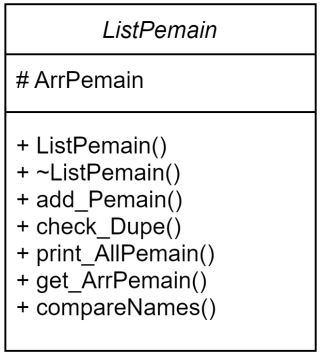
1.5 Kelas Game



Kelas Game tidak menerapkan konsep Abstract Base Class, inheritance, maupun polymorphism atau bisa disebut berdiri sendiri. Tetapi tidak berarti kelas ini tidak memiliki keterhubungan dengan kelas lain. Dalam implementasinya kelas ini membutuhkan kelas lainnya seperti kelas Pemain beserta turunan-turunannya.

Kelebihan dari pemakaian konsep ini ialah ketika terjadi pengubahan atribut maupun method dalam kelas ini tidak akan memengaruhi kelas lain secara langsung. Kecuali kelas lain memiliki ketergantungan dengan kelas Game.

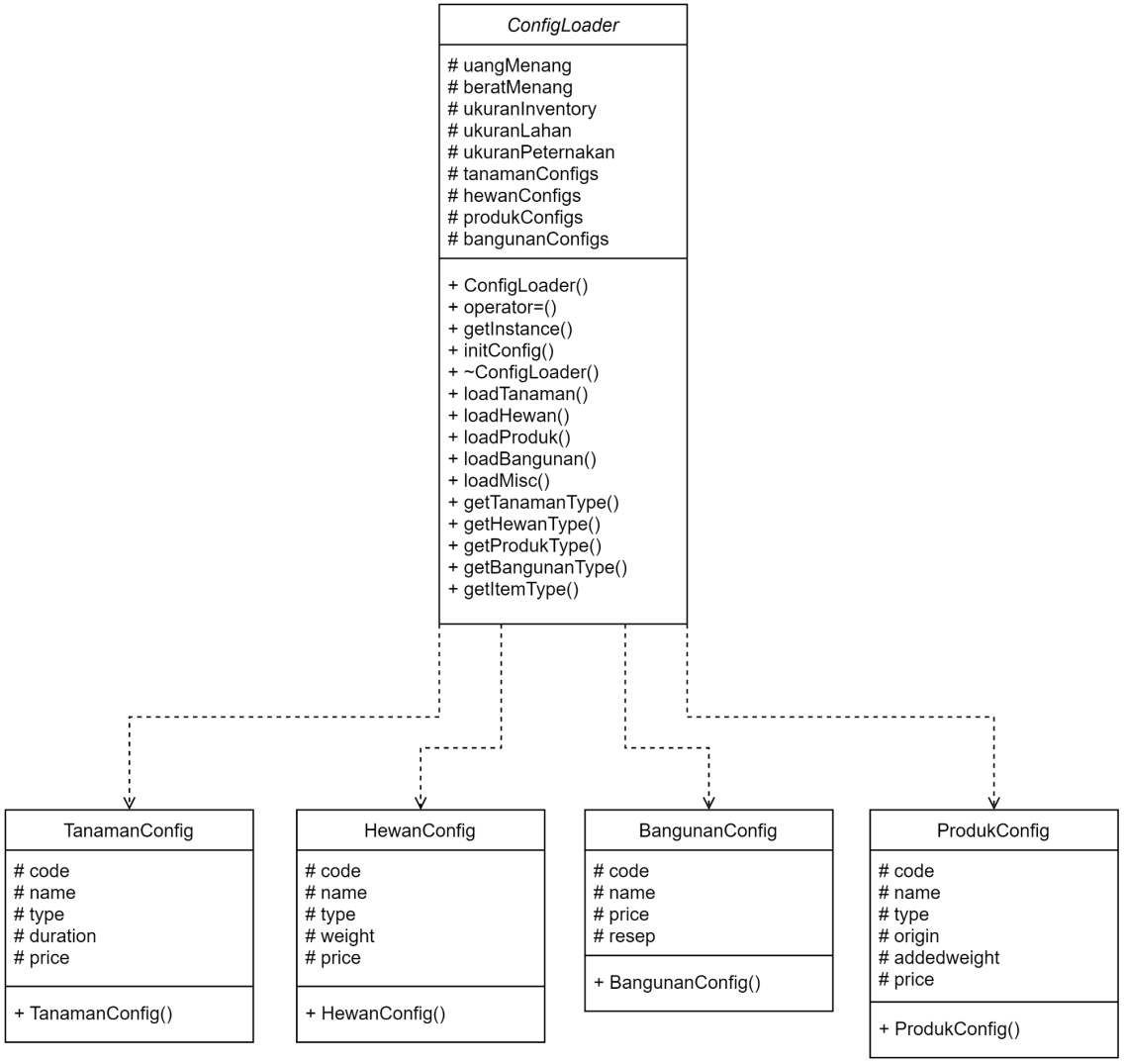
1.6 ListPemain



Kelas ListPemain tidak menerapkan konsep Abstract Base Class, inheritance, maupun polymorphism atau bisa disebut berdiri sendiri. Tetapi tidak berarti kelas ini tidak memiliki keterhubungan dengan kelas lain. Dalam implementasinya kelas ini membutuhkan kelas lainnya seperti kelas Pemain beserta turunan-turunannya.

Kelebihan dari pemakaian konsep ini ialah ketika terjadi pengubahan atribut maupun method dalam kelas ini tidak akan memengaruhi kelas lain secara langsung. Kecuali kelas lain memiliki ketergantungan dengan kelas ListPemain.

1.7 Kelas ConfigLoader, TanamanConfig, HewanConfig, BangunanConfig, dan ProdukConfig



Kelas ConfigLoader memiliki ketergantungan (*dependency*) secara langsung dengan kelas TanamanConfig, HewanConfig, BangunanConfig, dan ProdukConfig. Dalam grafik di samping digambarkan dengan garis panah putus-putus. Kelebihan dari pemakaian konsep ini ialah memudahkan dalam memahami dan membuat kelas ConfigLoader, karena telah dibantu dengan kelas-kelas lainnya.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep *inheritance* dan *polymorphism* digunakan oleh kelas Item yang punya turunan kelas Hewan, Tanaman, Bangunan, dan Produk. Kelas Hewan sendiri memiliki 3 turunan kelas: Omnivora, Herbivora, dan Karnivora, sedangkan kelas Tanaman memiliki 2 turunan kelas: Fruit dan Material. Kelas Produk memiliki 2 turunan kelas: produkHewan dan produkTumbuhan. Kelas Item mewakilkan segala objek yang bisa dimasukkan dalam inventory, hal ini meliputi hewan, tanaman, bangunan, dan produk. Penggunaan kelas item ini diperlukan supaya kita bisa mengetahui jenis item yang disimpan (hewan atau tanaman atau yang lain) lewat atribut itemType, penting untuk validasi isi slot pada command. Pada kelas Item terdapat atribut-atribut umum yang dimiliki keempat turunannya, yaitu id, code, name, dan price yang akan diturunkan pada kelas turunannya.

Penggunaan konsep *inheritance* dan *polymorphism* ini penting untuk melakukan *sharing* atribut dan method yang sama pada kelas anak. Kelas turunan pun juga dapat memiliki atribut dan metodenya masing-masing, atau melakukan *override* method dari kelas *parent*, menyesuaikan dengan fungsi kelas itu sendiri. Contohnya seperti pada kelas Hewan yang memiliki atribut tambahan harvestweight (berat minimal untuk bisa dipanen) dan *pure virtual method* CanEat yang akan di-*override* oleh turunannya. Dengan berpikir secara *inheritance* ini kita dimudahkan untuk membayangkan objek dalam kategori-kategori, berbagi kesamaan dalam atribut maupun metode.

```
// FILE ITEM.HPP
class Item {
    protected:
        string itemType;
        int id;
        string code;
        string name;
        int price;
    public:
        Item(string itemType, int id, string code, string name, int price);
        virtual ~Item();
        string getItemType() const;
        void setItemType(string itemType);
        string getCode() const;
```

```
        void setCode(string code);
        string getName() const;
        void setName(string name);
        int getPrice() const;
        void setPrice(int price);
        void setID(int id);
        bool operator<(const Item& other) const;
};

// FILE HEWAN.HPP
class Hewan : public Item {
private:
    string type;
    int harvestweight;
    int actualweight;
    int age;
public:
    static int DEADAGE;
    Hewan(int id);
    ~Hewan();
    void set_type(string type);
    string get_type();
    void set_age(int age);
    int get_age();
    void set_harvestweight(int harvestweight);
    int get_harvestweight() const;
    void set_actualweight(int actualweight);
    int get_actualweight() const;
    bool siapPanen();
    bool is_Dead();
    string getType() const;
    void setType(string type);
    virtual bool CanEat(Produk* makanan) = 0;
};
```

```
// FILE OMNIVORA.HPP
class Omnivora: public Hewan
{
public:
    Omnivora(int id);
    ~Omnivora();
    bool CanEat(Produk* makanan);
};

// FILE KARNIVORA.HPP
class Karnivora: public Hewan {
public:
    Karnivora(int id);
    ~Karnivora();
    bool CanEat(Produk* makanan);
};

// FILE HERBIVORA.HPP
class Herbivora: public Hewan
{
public:
    Herbivora(int id);
    ~Herbivora();
    bool CanEat(Produk* makanan);
};

// FILE TANAMAN.HPP
class Tanaman : public Item {
    private:
        string type;
        int durationToHarvest;
        int umur;
    public:
```

```
        Tanaman(int ID);
        ~Tanaman();
        string get_type();
        int get_durationToHarvest();
        int get_umur();
        void add_umur();
        bool siapPanen();
        virtual bool CanEat(Produk* makanan);
};


// FILE FRUIT.HPP
class Fruit: public Tanaman {
    public:
        Fruit(int ID);
        ~Fruit();
        bool CanEat(Produk* makanan);
};

// FILE MATERIAL.HPP
class Material : public Tanaman {
    public:
        Material(int ID);
        ~Material();
        bool CanEat(Produk* makanan);
};


// FILE PRODUK.HPP
class Produk : public Item {
    private:
        string tipe;
        string origin;
        int addedWeight;
```

```
    public:
        Produk(int id);
        virtual bool isEdibleBy() = 0;
        int getAddedWeight() const;
        string getType() const;
        void setType(string type);
        void setOrigin(string origin);
        void setAddedWeight(int weight);

};


class produkHewan : public Produk {
    public:
        produkHewan(int id);
        bool isEdibleBy() override;
};


class produkTumbuhan : public Produk {
    public:
        produkTumbuhan(int id);
        bool isEdibleBy() override;
};

// FILE BANGUNAN.HPP
class Bangunan : public Item {
    private:
        map<string, int> listOfResep;
    public:
        Bangunan(int ID);
        void setResep(map<string, int> resep);
};
```

Selain kelas Item, pada kelas Pemain juga dilakukan *inheritance* dan *polymorphism.* Kelas Pemain memiliki 2 kelas turunan: pekerja dan walikota, sedangkan kelas Pekerja itu sendiri memiliki 2 turunan kelas: Petani dan Peternak. Sama seperti sebelumnya, penggunaan kelas Pemain ini penting untuk *sharing* atribut umum, seperti username, peran, inventory, uang, dan beratbadan, yang dimiliki oleh semua peran. Pembuatan kelas Pekerja digunakan untuk membedakan pekerja (petani dan peternak) dengan walikota. Walikota dapat menarik pajak dari pekerja, sehingga kelas Pekerja memiliki atribut baru KTKP. Di kelas Pekerja ini akan diisi dengan metode-metode perpajakan dan validasi barang yang akan dijual di toko. Kelas turunan Pekerja, kelas Petani dan kelas Peternak hanya dibedakan oleh kepemilikan ladang atau peternakan dengan masing-masing metode yang dibutuhkan.

```
// FILE PEMAIN.HPP
class Pemain {
    protected:
        string username;
        string peran;
        Inventory inventory;
        int uang;
        int beratBadan;
    public:
        /* Default Constructor */
        Pemain();
        /* User-Defined Constructor */
        Pemain(string username, string peran);
        Pemain(string username, string peran, int uang, int beratBadan);
        /* Destructor */
        virtual ~Pemain();
        /* Operator == */
        bool operator==(const Pemain& pemain);
        /* Getter */
        string getUsername() const;
        string getPeran() const;
        Inventory getInventory() const;
        int getUang() const;
```

```
        int getBeratBadan() const;
        /* Setter */
        void setUsername(string username);
        void setPeran(string peran);
        void setInventory(Inventory inventory);
        void setUang(int uang);
        void setBeratBadan(int beratBadan);
        /* Menyimpan item ke inventory */
        void simpanItem(Item* item, string lokasi);
        /* Mengambil item dari inventory */
        Item* ambilItem(string lokasi);
        /* Makan */
        void makan(Produk* produk);
        /* Mengecek apakah pemain bisa membeli barang */
        virtual bool bisaBeli(Item* item);
        /* Mengecek apakah pemain bisa menjual barang */
        virtual bool bisaJual(Item* item);
        /* Membeli barang */
        void beli(Item* item, int kuantitas);
        /* Menjual barang */
        void jual(Item* item, int kuantitas);
        /* Mengecek apakah pemain memenuhi kondisi menang */
        bool isMenang();
        /* Operator == untuk membandingkan pointer pemain */
        bool operator==(const Pemain* other) const;
};

// FILE WALIKOTA.HPP
class Walikota : public Pemain {
    private:
        /* Membuat instance dari Walikota */
        static Walikota* instance;
        /* User-Defined Constructor */
        Walikota(string username);
```

```
        // Walikota(string username, int uang, int beratBadan);
    public:
        /* Mengembalikan instance dari Walikota */
        static Walikota* getInstance(string username);
        /* Destructor */
        ~Walikota();
        /* Mengecek apakah pemain bisa membeli barang */
        bool bisaBeli(Item* item) override;
        /* Menagih pajak */
        void tagihPajak(ListPemain list_pemain);
        /* Membangun bangunan */
        void bangunBangunan(Bangunan bangunan);
        /* Menambah pemain */
        void tambahPemain(string username, string peran, ListPemain list_pemain);
};


// FILE PEKERJA.HPP
class Pekerja : public Pemain {
    protected:
        int KTKP;
    public:
        /* User-Defined Constructor */
        Pekerja(string username, string peran, int KTKP);
        Pekerja(string username, string peran, int uang, int beratBadan, int KTKP);
        /* Destructor */
        virtual ~Pekerja();
        /* Getter */
        int getKTKP() const;
        /* Setter */
        void setKTKP(int KTKP);
        /* Mengecek apakah pemain bisa menjual barang */
        bool bisaJual(Item* item) override;
        /* Menghitung kekayaan pemain */
```

```
        virtual int hitungKekayaan();
        /* Menghitung KKP */
        int getKKP();
        /* Menghitung pajak yang harus dibayar oleh pemain */
        int hitungPajak();
        /* Membayar pajak */
        void bayarPajak();
};


// FILE PETANI.HPP
class Petani : public Pekerja {
    private:
        Ladang ladang;
    public:
        /* Default Constructor */
        Petani();
        /* User-Defined Constructor */
        Petani(string username);
        Petani(string username, int uang, int beratBadan);
        /* Destructor */
        ~Petani();
        /* Getter */
        Ladang getLadang() const;
        /* Menghitung kekayaan petani */
        int hitungKekayaan();
        /* Menanam tanaman di ladang */
        void tanam(Tanaman tanaman, string lokasi);
        /* Memanen tanaman yang ada di ladang */
        void panenTanaman(string lokasi);
};


// FILE PETERNAK.HPP
```

```
class Peternakan{
    private:
        int baris;
        int kolom;
        Grid<Hewan*> kotak;
        int charBarisMaksimal;
        int charKolomMaksimal;
        //49 = 1
        //65 = A

    public:
        // Constructor
        Peternakan(); // ukuran ambil dari config

        Peternakan(int, int);

        ~Peternakan();

        bool isLokasiValid(string);

        void Ternak(string, Hewan*); //Menambahkan ternak ke slot lahan
        void addHewan(Hewan*, string); // Ganti jadi ini

        void Panen(string); //Memanen hewan dengan kode yang sama dengan inputan
        Hewan* delHewan(string); // Ganti jadi ini

        void CetakPeternakanHelper();
        void CetakPeternakan();

        bool isFull(); //Mengecek apakah penyimpanan sudah penuh atau belum

        bool isEmpty();

        void BeriMakan(Produk*);
```

```
    Grid<Hewan*> getKotak() const;

    bool isSiapPanen(Hewan*); //Mengecek apakah suatu hewan siap panen atau tidak
};
```

Penggunaan konsep *inheritance* & *polymorphism* juga diterapkan pada kelas Grid dan turunannya, kelas Inventory, Ladang, dan Peternakan. Hal ini mempertimbangkan pola konsep Inventory, Ladang, dan Peternakan yang mirip, dengan petak-petak yang menyimpan sebuah objek. Hanya saja, Inventory menyimpan pointer Item, Ladang menyimpan pointer Tanaman dan Peternakan menyimpan pointer Hewan.

2.2. Method/Operator Overloading

2.2.1 Method Overloading

Method overloading adalah penggunaan nama metode yang sama, namun memiliki parameter dan cara penggunaan yang berbeda. Method overloading kami gunakan pada kelas Inventory, Pemain, Peternak, dan Petani. Fungsi penggunaan method overloading pada SimpanBarang inventory penting karena adanya 2 tipe penyimpanan. SimpanBarang dengan 1 parameter pointer item adalah penyimpanan otomatis, atau mencari lokasi yang kosong pertama pada matriks penyimpan, sedangkan SimpanBarang kedua dengan 2 parameter pointer item dan string lokasi, untuk meletakkan item ke dalam inventory slot lokasi. Pada kelas Pemain, konsep method overloading digunakan pada constructor pemain, antara constructor pemain status default (starter uang 50 dan berat 40), serta constructor pemain user defined (uang dan berat ditentukan sendiri). Hal ini berdampak pada pembentukan metode konstruktor kelas turunan Pemain, kelas Pekerja, kelas Petani, dan kelas Peternak.

```
// FILE INVENTORY.HPP
class Inventory: public Grid<Item*>{
   private:
```

```
        ...

    public:

        void SimpanBarang(Item*); //Auto

        void SimpanBarang(Item*, string); //Simpan barang manual dengan lokasi tertentu


// FILE PEMAIN.HPP
class Pemain {
    protected:
        ...
    public:
        Pemain(string username, string peran);
        Pemain(string username, string peran, int uang, int beratBadan);

// FILE PEKERJA.HPP
class Pekerja : public Pemain {
    protected:
        int KTKP;
    public:
        /* User-Defined Constructor */
        Pekerja(string username, string peran, int KTKP);
        Pekerja(string username, string peran, int uang, int beratBadan, int KTKP);


// FILE PETANI.HPP
class Petani : public Pekerja {
    private:
        ...
    public:
        Petani(string username);
        Petani(string username, int uang, int beratBadan);
```

```
// FILE PETERNAK.HPP
class Peternak : public Pekerja {
    private:
        ...
    public:
        Peternak(string username);
        Peternak(string username, int uang, int beratBadan);
```

2.2.2 Operator Overloading

Operator overloading adalah mengubah cara kerja operator seperti +, -, *, /, ==, <, >, etc untuk melakukan proses pada objek bentukan dari user, sebab bawaan operator tersebut hanya bisa melakukan proses pada tipe data umum seperti int, string, atau float. Operator overloading kami gunakan pada kelas Pemain, Item, dan Inventory. Alasan penggunaan ini adalah untuk memberikan fleksibilitas metode mengikuti jenis parameter atau jumlah parameter.

```
// FILE PEMAIN.HPP
class Pemain {
    protected:
        ...
    public:
        /* Operator == untuk membandingkan pointer pemain */
        bool operator==(const Pemain* other) const;
};
```

Penggunaan operator overloading dapat dilihat sebagai berikut:

```
// FILE LISTPEMAIN.CPP
int Game::get_idxinalist(Pemain* x){
    vector<Pemain*> pemain = listPemain.get_ArrPemain();

    for (int i = 0; i<(pemain.size()); i++){
        if (*pemain[i] == x){    //menggunakan operator overloading == untuk mengecek apakah pemain ke-i sama dengan x.
            return i;
        }
    }
    return -1;  //tidak ketemu
}
```

2.3. Template & Generic Classes

Kelas template adalah kelas yang memiliki atribut atau method dengan tipe data bebas tergantung oleh kelas turunannya. Tipe data tersebut dapat berupa sebuah tipe objek bentukan dari user maupun tipe-tipe yang sudah sering kita jumpai seperti integer. Kelas Grid adalah kelas template yang memiliki atribut sebuah map key lokasi dan value sebuah objek user defined, sehingga setiap kelas turunan Grid memiliki atribut map yang berbeda-beda. Pada Inventory, map tersebut diisi dengan value bertipe Item, yaitu objek yang meliputi Bangunan, Hewan, Tanaman, dan Produk. Alasan penggunaan dari konsep template ini karena kami melihat bahwa inventory, ladang, dan peternakan sama-sama menggunakan konsep petak/grid yang sama, dengan cara pengaksesan yang sama, tetapi dengan isi perpetaknya yang berbeda jenis. Maka dari itu, kami memutuskan untuk menggunakan template untuk memudahkan akses dan pembangunan struktur ketiga kelas sekaligus.

```
// GRID.HPP
template<class T>
class Grid{
    private:
        map<string, T> element;
        int numOfEl;
    public:
```

```
        Grid(){};
        ~Grid(){};
        bool isPresent(string k){};
        auto iteratorOf(string k){};
        T value(string k){};
        void add(string k, T val){};
        auto firstptr(){};
        auto endptr(){};
        void del(string k){};
        int howMuchElement(){};
        map<string, T> getElMap() const {};
};

// FILE INVENTORY.HPP
class Inventory: public Grid<Item*>{
    private:
        int baris;
        int kolom;
        int charBarisMaksimal;
        int charKolomMaksimal;

    public:
        ...
};

// FILE LADANG.HPP
class Ladang: public Grid<Tanaman*>{
    private:
        int baris;
        int kolom;
        int charBarisMaksimal;
        int charKolomMaksimal;

    public:
```

```
        ...
};

// FILE PETERNAKAN.HPP
class Peternakan: public Grid<Hewan*>{
    private:
        int baris;
        int kolom;
        int charBarisMaksimal;
        int charKolomMaksimal;

    public:
        ...
};
```

2.4. Exception

```
// ABC Exception
class Exception {
    public:
        virtual string what() = 0;
};

class NoAnimalFood : public Exception
{/*...*/};

class NoMoney : public Exception
{/*...*/};

class NoInventorySpace : public Exception
```

```
{/*...*/};

class InvalidRole : public Exception
{/*...*/};

class LadangFull : public Exception
{/*...*/};

class noTanamaninInv : public Exception
{/*...*/};

#endif
```

Tabel 2.4.1 Exception dan Penjelasan

Cuplikan Kode	Penjelasan

`// ABC Exception`
`class Exception {`
`  public:`
`  virtual string what() = 0;`
`};`
	Exception abstract base class yang method what() akan diturunkan untuk melakukan return error message masing-masing

`// Tidak ada makanan yang cocok/tidak ada makanan untuk hewan pada KASIH PANGAN`
`class NoAnimalFood : public Exception {`
`  public:`
`  string what() override {`
`  return "Tidak ada makanan untuk hewan ternak.";`
`  }`
	Exception tidak ada makanan hewan sama sekali di inventory ketika akan melakukan command KASIH_MAKAN, sehingga otomatis tidak ada yang bisa dilakukan.

`};`	

`class NoMoney : public Exception {`
`  public:`
`  string what() override {`
`  return "Gulden kamu kurang.";`
`  }`
`};`
	Exception gulden kurang dalam command TAMBAH_PEMAIN sehingga command tidak bisa dilanjutkan.

`class NoInventorySpace : public Exception {`
`  public:`
`  string what() override {`
`  return "Inventorymu penuh.";`
`  }`
`};`
	Exception ruang inventory penuh pada contoh command PANEN atau BELI yang membutuhkan minimal 1 slot inventory kosong.

`class InvalidRole : public Exception {`
`  public:`
`  string what() override {`
`  return "Peranmu tidak bisa menggunakan command ini.";`
`  }`
`};`
	Exception peran invalid karena peran currentpemain tidak memenuhi kriteria untuk baca command, sehingga langsung throw error peran invalid.

Dari contoh beberapa cuplikan kode di atas, kita melihat bahwa exception penting untuk handle kasus-kasus tidak mungkin yang mengharuskan/lebih baik untuk kita melakukan terminate program dan mengeluarkan pesan kesalahan. Exception ini penting dalam melakukan handle kesalahan dan berbagai macam exception ini akan memberi informasi alasan program *terminate abnormally.* Adanya exception juga membantu menjaga control flow program supaya kita bisa langsung melompat dan skip menuju kode lain yang tidak terkait dengan error tersebut (penggunaan try catch).

Penggunaan exception dapat dilihat pada command-command dan try catch pada Game.cpp.

Cuplikan Kode	Penjelasan

// FILE GAME.CPP
void Game::start_game(){
...
while (!finish){
while (true){
…
try {
if (command == "NEXT") {
next_turn();
break;
}
else if (command == "CETAK_PENYIMPANAN") {
printPenyimpanan(currentpemain);
}
else if (command == "PUNGUT_PAJAK") {
pungutPajak(currentpemain,listPemain);
}
else if (command == "CETAK_LADANG") {
printLadang(currentpemain);
}
else if (command == "CETAK_PETERNAKAN") {
printPeternakan(currentpemain);
}
else if (command == "TANAM") {
tanam(currentpemain);
}
else if (command == "TERNAK") {
ternak(currentpemain);
}
else if (command == "BANGUN") {
bangunBangunan(currentpemain);
}
else if (command == "MAKAN") {
makan(currentpemain);
}
else if (command == "KASIH_MAKAN") {
kasihMakan(currentpemain);
	Game.cpp akan *catch* semua error yang di-*throw* oleh command-command. Untuk mempermudah tanpa mencantumkan exception yang dilempar secara spesifik, *catcher* exception adalah pointer base class Exception yang otomatis akan menyesuaikan dengan exception yang dilempar dan melakukan printing pesan error.

```
            }
            else if (command == "BELI") {
                beli(currentpemain, toko);
            }
            else if (command == "JUAL") {
                jual(currentpemain, toko);
            }
            else if (command == "PANEN") {
                if (currentpemain->getPeran() == "Petani") {
                    panen_petani(currentpemain);
                }
                else if (currentpemain->getPeran() == "Peternak") {
                    panen_peternak(currentpemain);
                }
            }
            else if (command == "SIMPAN") {
                // Handle SIMPAN command
            }
            else if (command == "TAMBAH PEMAIN") {
                addPemain(listPemain, currentpemain);
            }
            else {
                cout << "Masukan tidak valid, silakan ulangi lagi." << endl;
            }

            ...
            }
        }
        catch (Exception& e) {
            cout << e.what() << endl;
        }
    }

  }
  ...
}
```

```
// FILE TANAM.CPP
void tanam(Pemain* p){
  if (p->getPeran() !="Petani"){
    throw InvalidRole();
```

Command tanam akan throw invalid role jika peran pemain sekarang bukan petani. Exception LadangFull akan

<table>
<tr>
<td>

```
    }

    Petani* petani = dynamic_cast<Petani*>(p);
    // Ladang Full
    if (petani->getLadang().isFull()){
        throw LadangFull();
    }

    // no food
    if (!(p->getInventory().isThereTanaman())){
        throw noTanamaninInv();
    }
}
```

</td>
<td>di-throw jika ladang sudah penuh sehingga tidak memungkinkan untuk menanam tanaman lagi. Exception noTanamaninInv akan di-throw jika tidak terdapat tanaman di inventory yang bisa ditanam oleh pemain.</td>
</tr>
<tr>
<td>

```
// FILE PANEN.CPP
void panen_petani(Pemain* p) {

    ...

    if (ListSiapPanen.empty()){
        throw TidakAdaPanen();
    }
    …
    if (nomor_tanaman < 1 || nomor_tanaman >= i ){
        throw InvalidNomorPanen();
    }
    …
    if (jumlah_petak < 1 || jumlah_petak > maks_jumlah_panen){
        throw InvalidJumlahPanen();
    }
    if (jumlah_petak > p->getInventory().SlotTersisa()){
        throw PenyimpananTidakCukup();
    }
}
```

</td>
<td>

- Command panen_petani (atau PANEN) akan melakukan *throw* TidakAdaPanen exception apabila tidak ada tanaman yang siap dipanen (dilakukan pengecekan kekosongan list yang direturn metode ListSiapPanen).
- Exception InvalidNomorPanen() mengatasi input user yang melebihi jumlah tanaman yang bisa dipanen (counter i) atau lebih kecil dari 1 (invalid).
- Exception InvalidJumlahPanen() akan *handle* masalah masukan jumlah petak yang melebihi tanaman yang bisa dipanen (maks_jumlah_panen) atau kurang dari 1 (invalid).
- Exception PenyimpananTidakCukup melakukan validasi jumlah petak yang kosong pada inventory

</td>
</tr>
</table>

	muat untuk menampung seluruh produk yang dihasilkan dari panen.

2.5. C++ Standard Template Library

2.5.1 Vector

Penggunaan STL vector sebagai container pada kelas ListPemain. Pada kelas ListPemain, vector digunakan untuk menyimpan kumpulan pointer objek Pemain yang melambangkan list pemain yang ada di dalam permainan. Penggunaan vector memudahkan proses insert Pemain baru, sifatnya yang dinamis juga memudahkan kita untuk mengatur penambahan Pemain tanpa harus memperhatikan ukuran *container* sekarang. Daftar objek pemain akan dimasukkan ke dalam atribut ArrPemain dalam kelas ListPemain. Dengan menggunakan vector, penambahan pemain hanya perlu dilakukan dengan *push_back*, lalu akan dilakukan sorting urutan main sesuai dengan leksikografis.

```
// FILE LISTPEMAIN.HPP
class ListPemain{
   private:
      vector<Pemain*> ArrPemain;
```

2.5.2 Map

Penggunaan STL map menyimpan pasangan key dan value. Hal ini memudahkan kita untuk melakukan akses value berdasarkan key, melakukan penghapusan dengan erase(key), dan berbagai fungsi-fungsi lain yang terkait pada STL map. Pada kelas Bangunan, STL map listofResep akan menyimpan key string nama bahan (ex: TEAK_WOOD) dan value berupa jumlah bahan tersebut yang dibutuhkan untuk membangun bangunan tersebut. Sekadar mengingatkan, nama, ID, dan atribut lainnya terdapat pada kelas Item, sehingga pada kelas Bangunan hanya terdapat map resep bahan yang dibutuhkan. Begitu pula pada kelas Grid, STL map digunakan untuk menyimpan key string lokasi slot pada grid dan value berupa objek template (bebas). Map lebih dipilih pada Grid sebab urutan pemasukan elemen tidak selalu berurutan dari slot terkecil, sehingga tidak diperlukan penyimpanan yang *ordered*. Sedangkan pada kelas Toko, container STL map digunakan untuk container menyimpan key Item (barang yang ada di Toko) dan value berupa stok barang tersebut. Pada ConfigLoader, map digunakan untuk memetakan ID sebagai key dan value adalah kelas config masing-masing sesuai dengan apa yang ingin di-*load.* ID setiap tanaman

berbeda-beda dan selalu sama (bukan *auto-increment*). Misal, kita ingin melakukan load config Tanaman, dengan menggunakan ID sebagai key dan value objek constructor TanamanConfig.

```
// FILE BANGUNAN.HPP
class Bangunan : public Item {
    private:
        map<string, int> listOfResep;


// FILE GRIDMAP.HPP
template<class T>
class Grid{
    private:
        map<string, T> element;
        int numOfEl;


// FILE TOKO.HPP
class Toko {
    private:
        map<Item, int> inventory;


// FILE CONFIGLOADER.HPP
class BangunanConfig {
public:
    string code;
    string name;
    int price;
    map<string, int> resep;


class ConfigLoader {
private:
```

```
    ConfigLoader() {}
    ConfigLoader(const ConfigLoader&) = delete;
    ConfigLoader& operator=(const ConfigLoader&) = delete;
public:
    static ConfigLoader& getInstance(){
        static ConfigLoader instance;
        return instance;
    }
    /* Atribut */
    int uangMenang;
    int beratMenang;
    pair<int, int> ukuranInventory;
    pair<int, int> ukuranLahan;
    pair<int, int> ukuranPeternakan;
    map<int, TanamanConfig> tanamanConfigs;
    map<int, HewanConfig> hewanConfigs;
    map<int, ProdukConfig> produkConfigs;
    map<int, BangunanConfig> bangunanConfigs;
```

2.5.3 Pair

STL Pair merepresentasikan sepasang objek yang bisa berbeda tipe. Pada kelas ConfigLoader, STL Pair digunakan untuk menyimpan nilai panjang dan lebar dari inventory, lahan, dan peternakan.

```
// FILE CONFIGLOADER.HPP
class ConfigLoader {
private:
    ConfigLoader() {}
    ConfigLoader(const ConfigLoader&) = delete;
    ConfigLoader& operator=(const ConfigLoader&) = delete;
public:
```

```
    static ConfigLoader& getInstance(){
        static ConfigLoader instance;
        return instance;
    }
    /* Atribut */
    int uangMenang;
    int beratMenang;
    pair<int, int> ukuranInventory;
    pair<int, int> ukuranLahan;
    pair<int, int> ukuranPeternakan;
    map<int, TanamanConfig> tanamanConfigs;
    map<int, HewanConfig> hewanConfigs;
    map<int, ProdukConfig> produkConfigs;
    map<int, BangunanConfig> bangunanConfigs;
```

2.5.4 Sorting

Built-in function sorting dari C++, sort() digunakan oleh ListPemain untuk sort daftar Pemain secara leksikografis berdasarkan username. Pemain.Fungsi compareNames merupakan sebuah fungsi pembanding yang digunakan untuk mengurutkan vektor ArrPemain berdasarkan username dari objek Pemain. Fungsi add_Pemain bertanggung jawab untuk menambahkan pemain baru (objek Pemain) ke dalam vektor ArrPemain setelah memeriksa duplikat. Jika username pemain baru belum ada dalam vektor, maka pemain tersebut akan ditambahkan ke vektor dan vektor akan diurutkan berdasarkan username.

```
// FILE LISTPEMAIN.CPP

bool ListPemain::compareNames(const Pemain* a, const Pemain* b) {
    return a->getUsername() < b->getUsername();
}
```

```
void ListPemain::add_Pemain(Pemain* pemain){
    try{
        check_Dupe(pemain->getUsername());
        ArrPemain.push_back(pemain);
        sort(ArrPemain.begin(), ArrPemain.end(), compareNames);
        cout << "Pemain baru ditambahkan!" << endl;
        cout << "Selamat datang \"" << pemain->getUsername() << "\" di kota ini!" << endl;
    }
    catch(DupeName e){
        cout << e.what() <<endl;
    }

}
```

2.6. Abstract Class dan Virtual Function

2.6.1 Abstract Class: Exception

Abstract base class adalah kelas yang memiliki minimal 1 metode pure virtual. Exception dijadikan abstract class agar setiap class turunannya dapat memiliki implementasinya sendiri terkait kalimat apa yang akan dithrow ketika suatu eror dihadapi. Pada penggunaan

```
// FILE EXCEPTION.HPP
// ABC
class Exception {

    public:

        virtual string what() = 0;

};
```

.

Pengguna

2.6.2 Abstract Class: Hewan

Method CanEat dijadikan pure virtual sebab hewan herbivora, karnivora, dan omnivora, memiliki perbedaan dalam makanan yang dapat dimakan. Maka dari itu, implementasi fungsi ini haruslah berada pada child class, sehingga class Hewan dijadikan abstract class. Pada fungsi ini, parameter pointer of produk digunakan. Dengan demikian, Hewan memiliki asosiasi dengan Class Produk. Selain itu, Hewan juga merupakan Inheritance dari class Item.

```
// FILE HEWAN.HPP
class Hewan : public Item {

private:

        string type;

        int harvestweight;

        int actualweight;

        int age;

 public:

        static int DEADAGE;

        Hewan(int id);

        Hewan(string name);
```

```
        ~Hewan();
        void set_type(string type);
        string get_type();
        void set_age(int age);
        int get_age();
        void set_harvestweight(int harvestweight);
        int get_harvestweight() const;
        void set_actualweight(int actualweight);
        int get_actualweight() const;
        bool siapPanen();
        bool is_Dead();
        string getType() const;
        void setType(string type);
        virtual bool CanEat(Produk* makanan) = 0;
};
```

2.6.3 Abstract Class: Produk

Method isEdibleBy() dijadikan pure virtual sebab hewan herbivora, karnivora, dan omnivora, memiliki perbedaan dalam makanan yang dapat dimakan. Maka dari itu, implementasi fungsi ini haruslah berada pada child class, sehingga class Produk dijadikan abstract class. Selain itu, Produk juga merupakan Inheritance dari class Item.

```
// FILE PRODUK.HPP
class Produk : public Item {

        private:

        string tipe;

        string origin;

        int addedWeight;

        public:

        Produk(int id);

        Produk(string name);

        virtual bool isEdibleBy() = 0;

        int getAddedWeight() const;

        string getType() const;

        void setType(string type);

        void setOrigin(string origin);

        void setAddedWeight(int weight);

};
```

3. Bonus Yang dikerjakan

3.1. Diagram Sistem

3.2. Bonus Kreasi Mandiri

Untuk mempermudah debugging dan testing program, dibuat fungsi testing checkStatus untuk pemain yang akan menampilkan uang dan berat badan sekarang.

```
#include "command.hpp"
#include "../header/Pemain/Pemain.hpp"

void checkStatus(Pemain* pemain){
    cout << "<<<<<<<<<< STATUS PEMAIN >>>>>>>>>>" << endl;
    cout << "🧑👧👵👶👩‍🦰👸🤴👩👮‍♂️👩‍🚒🧕👳‍♀️👩‍🔬" << endl;
    cout << "Nama: " << pemain->getUsername() << endl;
    cout << "Peran: " << pemain->getPeran() << endl;
    cout << "Uang: " << pemain->getUang() << " Gulden" << endl;
    cout << "Berat Badan: " << pemain->getBeratBadan() << endl;
}
```

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Kelas Pemain dan Turunannya	13522096	13522096
Kelas Hewan	13522012	13522012

Kelas Tanaman	13522104	13522012
Kelas Grid dan Turunannya	13522040	13522040
Kelas Bangunan, Toko, Barang	13522046	13522046
Kelas GameManager	13522012	13522012, 13522096
Muat	13522096	13522096
Simpan	13522046	13522046
Makefile	13522104	13522096
Next	13522012	13522012, 13522096
Command Cetak Penyimpanan	13522012, 13522040	13522040, 13522104
Command Pungut Pajak	13522096	13522096
Command Cetak Ladang dan Peternakan	13522012, 13522040	13522012
Command Tanam	13522012	13522040, 13522012
Command Ternak	13522104	13522012
Command Bangun Bangunan	13522046, 13522096	13522046
Command Makan	13522096	13522096
Command Memberi Pangan	13522040	13522012
Command Membeli	13522046, 13522096	13522096
Command Menjual	13522046	13522046

Command Memanen	13522104	13522012
Command Tambah Pemain	13522012	13522012, 13522096
Diagram Kelas Laporan	13522104	-
Laporan (exc: Diagram)	13522012, 13522040	

5. Link Repository

https://github.com/slntkllr01/IF2210_TB1_OOPsie