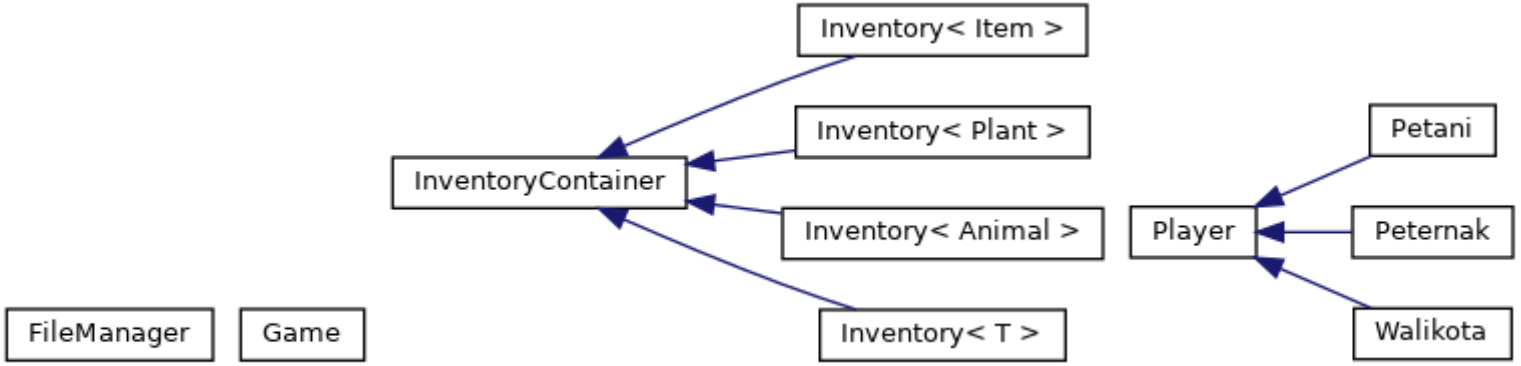
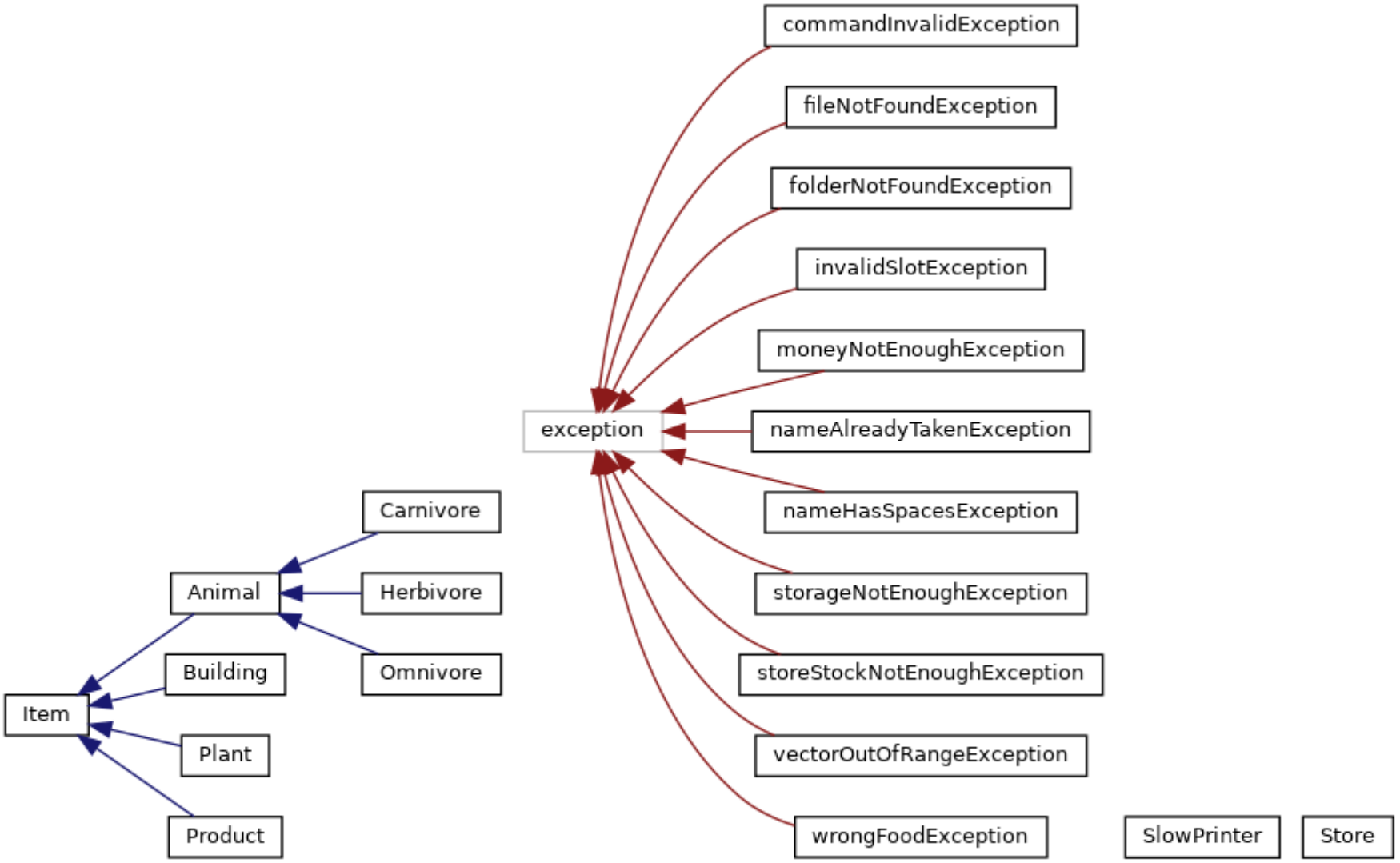
Kode Kelompok : BTC
Nama Kelompok : pembenci motor matic
1. 13522126 / Rizqika Mulia Pratama
2. 13522131 / Owen Tobias Sinurat
3. 13522143 / Muhammad Fatihul Irhab
4. 13522147 / Ikhwan Al Hakim
5. 13522158 / Muhammad Rasheed Qais Tandjung
Asisten Pembimbing : Malik Akbar Hashemi Rafsanjani

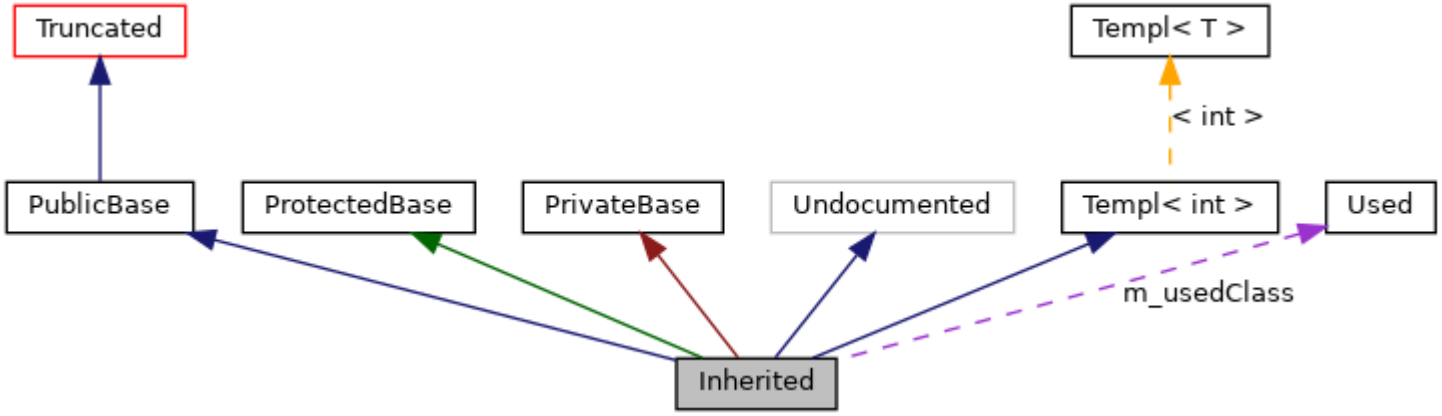
1. Diagram Kelas

a. Class Hierarchy





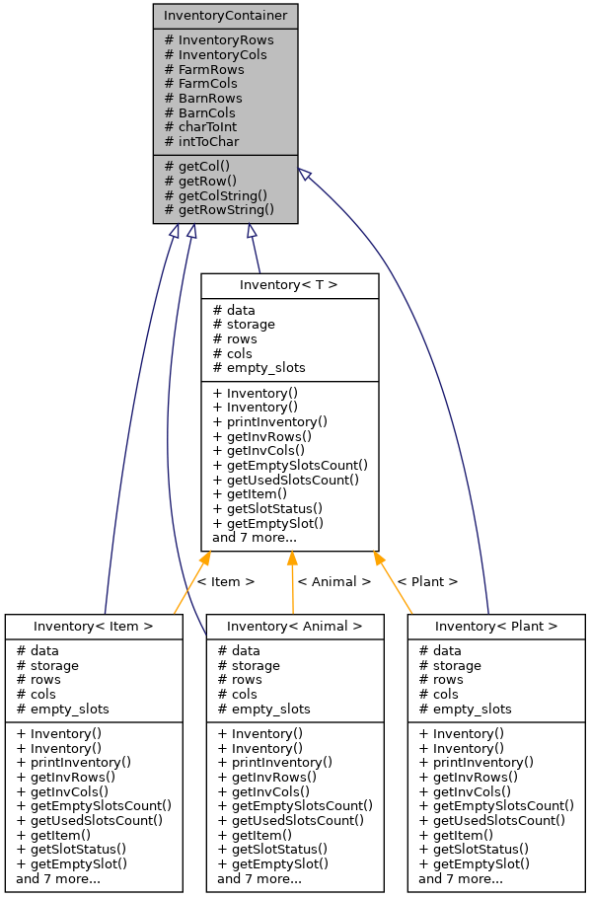
b. Class Diagram



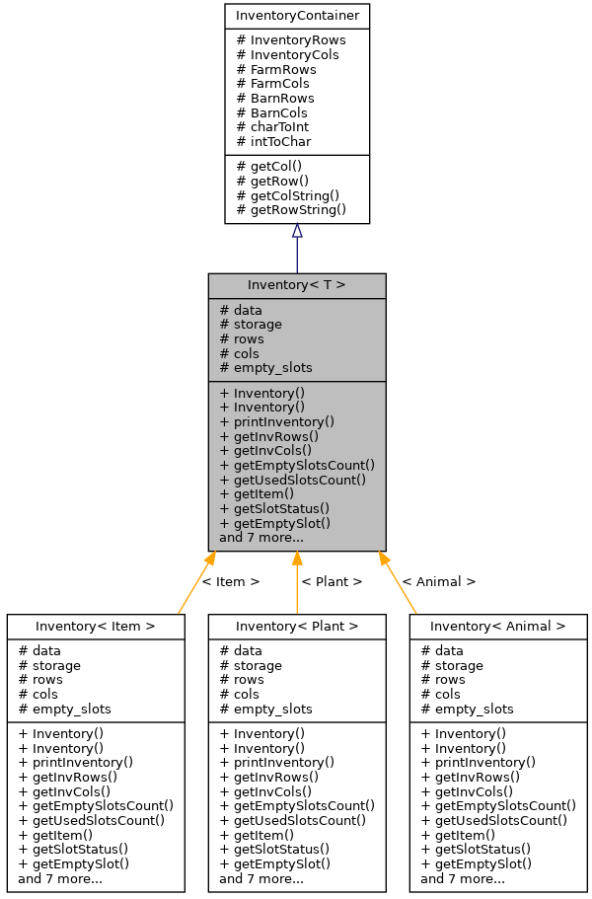
Legenda

- Kotak
  - Kotak abu-abu: struktur atau kelas untuk yang diagram tersebut dihasilkan.
  - Kotak batas hitam: struktur atau kelas yang didokumentasikan.
  - Kotak batas abu-abu: struktur atau kelas yang tidak didokumentasikan.
  - Kotak batas merah: struktur atau kelas yang didokumentasikan tetapi tidak semua hubungan warisan/kepemilikan ditunjukkan.
- Panah
  - Panah biru: memvisualisasikan hubungan public inheritance antara dua kelas.
  - Panah hijau: memvisualisasikan hubungan protected inheritance antara dua kelas.
  - Panah merah: memvisualisasikan hubungan private inheritance antara dua kelas.
  - Panah ungu: digunakan jika sebuah kelas diandalkan atau digunakan oleh kelas lain. Panah dilabeli dengan variabel yang melalui kelas atau struktur yang ditunjuk.
  - Panah kuning: menunjukkan hubungan antara instance template dan kelas template dari mana ia diinstansiasi. Panah dilabeli dengan parameter-template dari instance.

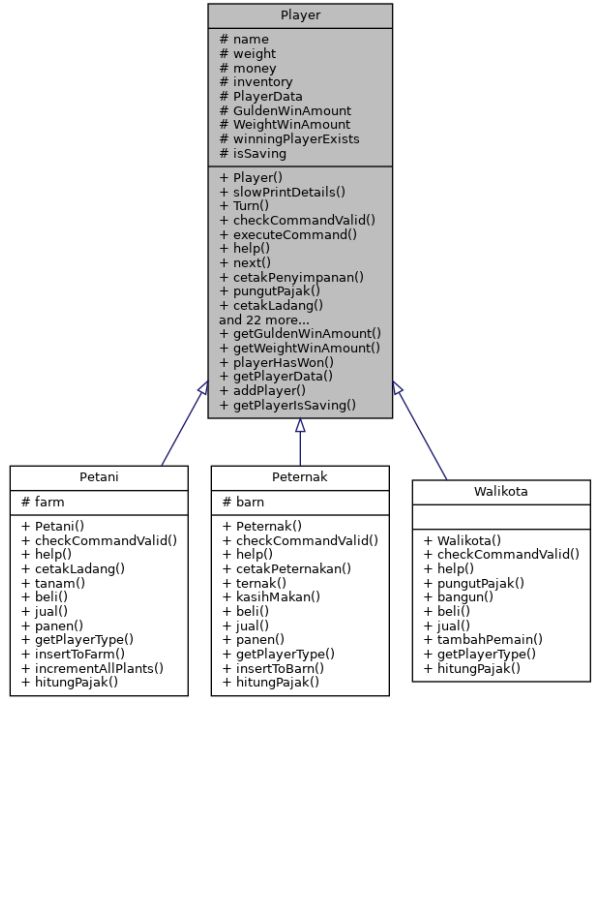
Class InventoryContainer Inheritance Diagram	**Class Inventory<T> Inheritance Diagram**	**Class Player Inheritance Diagram**



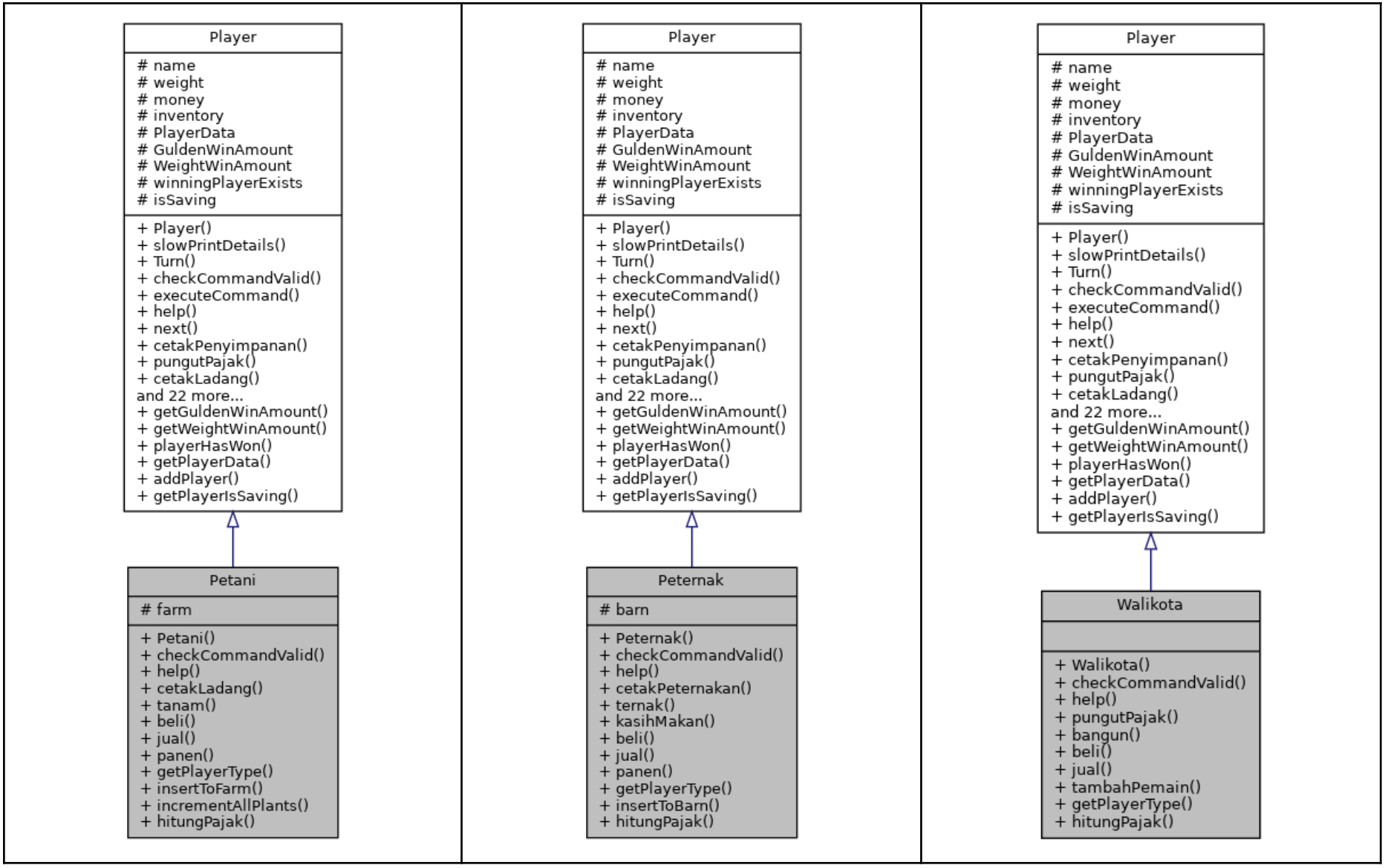
Class Petani
Inheritance Diagram



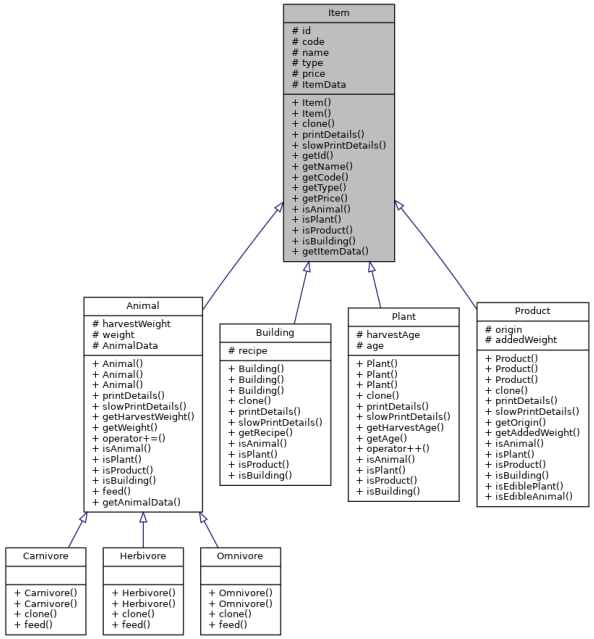
Class Peternak
Inheritance Diagram



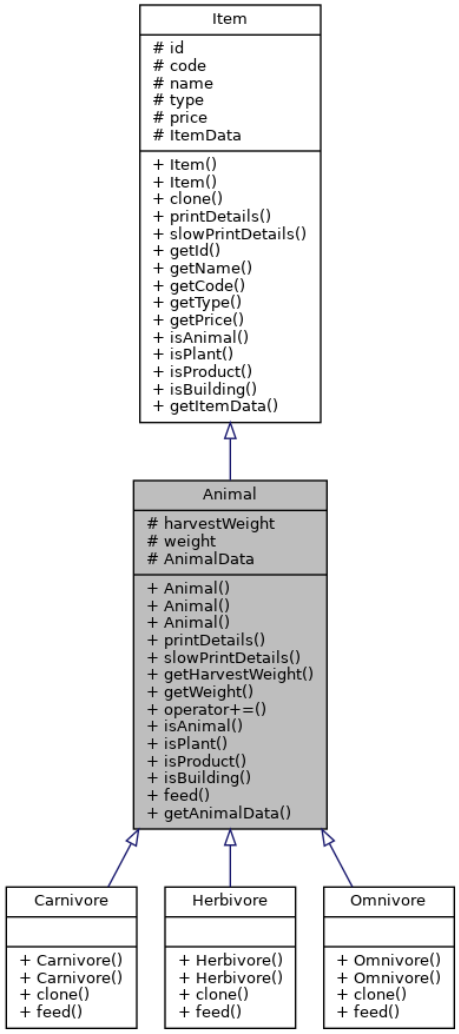
Class Walikota
Inheritance Diagram


Player
name
weight
money
inventory
PlayerData
GuldenWinAmount
WeightWinAmount
winningPlayerExists
isSaving
+ Player()
+ slowPrintDetails()
+ Turn()
+ checkCommandValid()
+ executeCommand()
+ help()
+ next()
+ cetakPenyimpanan()
+ pungutPajak()
+ cetakLadang()
and 22 more...
+ getGuldenWinAmount()
+ getWeightWinAmount()
+ playerHasWon()
+ getPlayerData()
+ addPlayer()
+ getPlayerIsSaving()
Petani
farm
+ Petani()
+ checkCommandValid()
+ help()
+ cetakLadang()
+ tanam()
+ beli()
+ jual()
+ panen()
+ getPlayerType()
+ insertToFarm()
+ incrementAllPlants()
+ hitungPajak()
Player
name
weight
money
inventory
PlayerData
GuldenWinAmount
WeightWinAmount
winningPlayerExists
isSaving
+ Player()
+ slowPrintDetails()
+ Turn()
+ checkCommandValid()
+ executeCommand()
+ help()
+ next()
+ cetakPenyimpanan()
+ pungutPajak()
+ cetakLadang()
and 22 more...
+ getGuldenWinAmount()
+ getWeightWinAmount()
+ playerHasWon()
+ getPlayerData()
+ addPlayer()
+ getPlayerIsSaving()
Peternak
barn
+ Peternak()
+ checkCommandValid()
+ help()
+ cetakPeternakan()
+ ternak()
+ kasihMakan()
+ beli()
+ jual()
+ panen()
+ getPlayerType()
+ insertToBarn()
+ hitungPajak()
Player
name
weight
money
inventory
PlayerData
GuldenWinAmount
WeightWinAmount
winningPlayerExists
isSaving
+ Player()
+ slowPrintDetails()
+ Turn()
+ checkCommandValid()
+ executeCommand()
+ help()
+ next()
+ cetakPenyimpanan()
+ pungutPajak()
+ cetakLadang()
and 22 more...
+ getGuldenWinAmount()
+ getWeightWinAmount()
+ playerHasWon()
+ getPlayerData()
+ addPlayer()
+ getPlayerIsSaving()
Walikota
+ Walikota()
+ checkCommandValid()
+ help()
+ pungutPajak()
+ bangun()
+ beli()
+ jual()
+ tambahPemain()
+ getPlayerType()
+ hitungPajak()

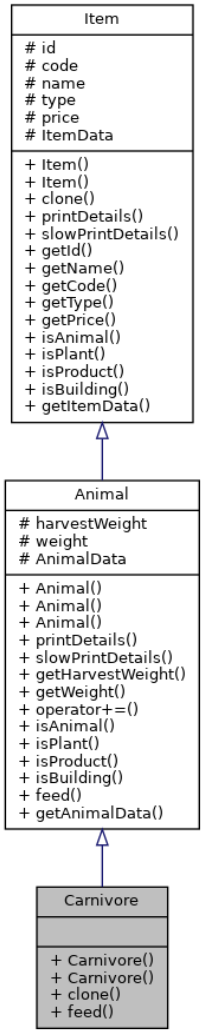
Class Item
Inheritance Diagram



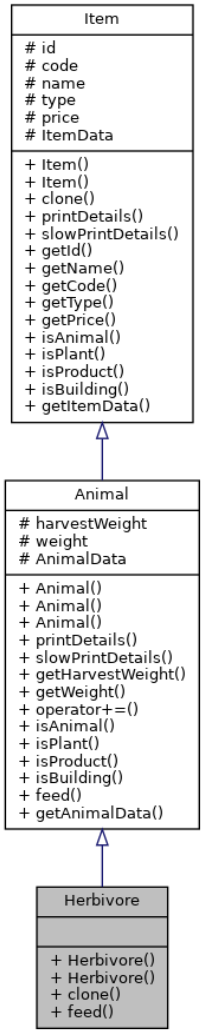
Class Animal
Inheritance Diagram



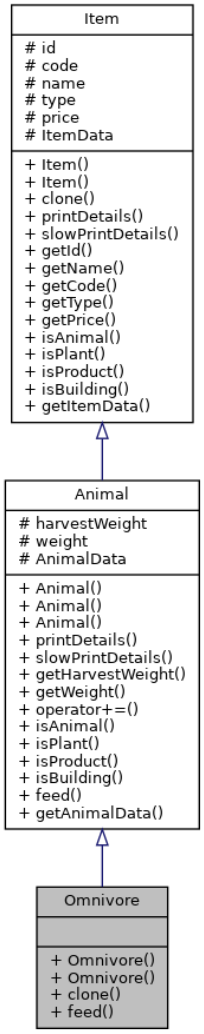
Class Carnivore
Inheritance Diagram



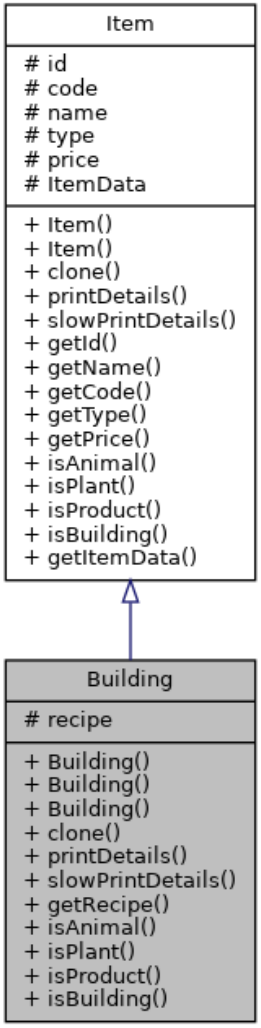
Class Herbivore Inheritance Diagram



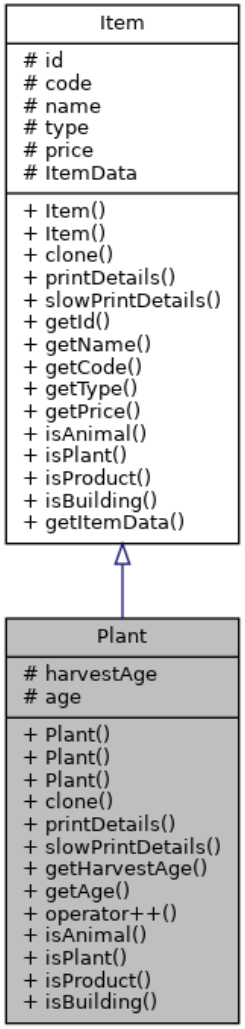
Class Omnivore Inheritance Diagram



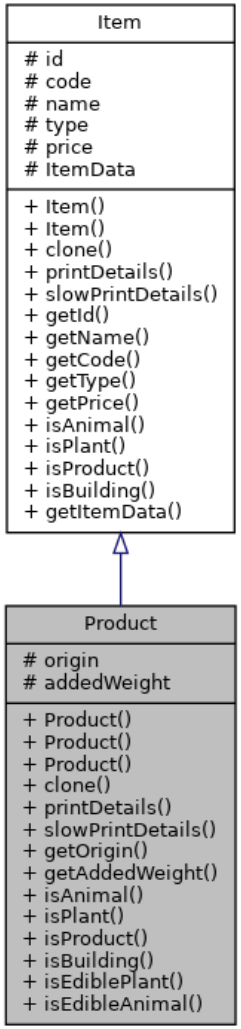
Class Building Inheritance Diagram



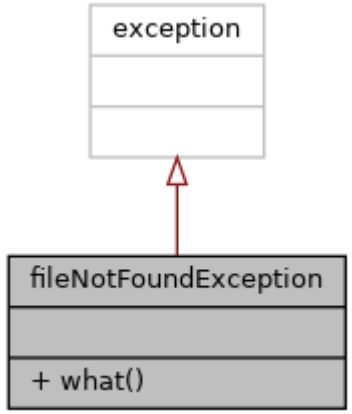
Class Plant
Inheritance Diagram



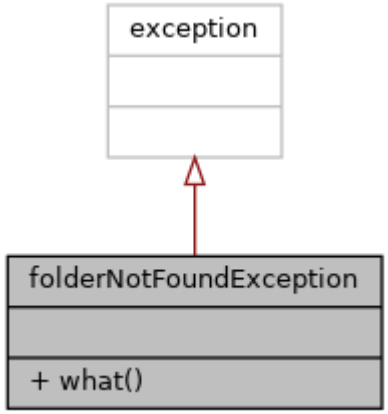
Class Product
Inheritance Diagram



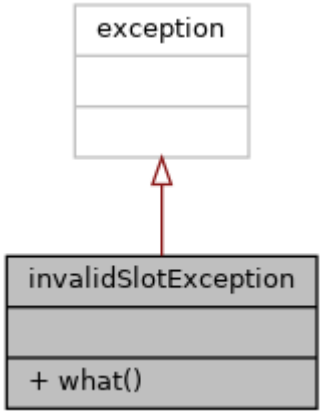
Class fileNotFoundException
Inheritance Diagram



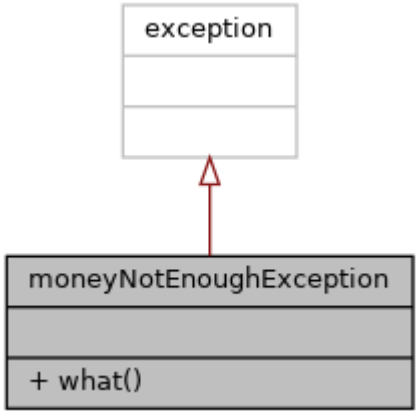
Class folderNotFoundException Inheritance Diagram



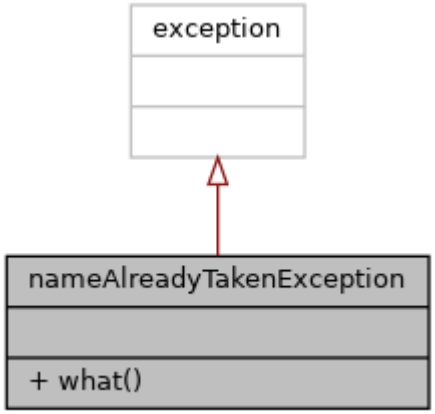
Class invalidSlotException Inheritance Diagram



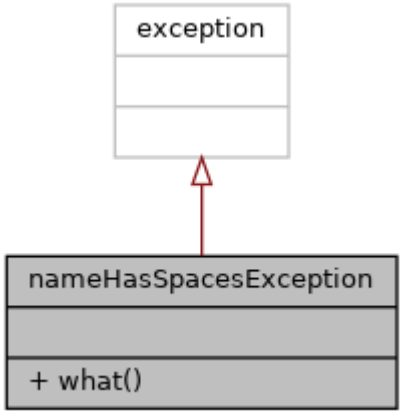
Class moneyNotEnoughException Inheritance Diagram



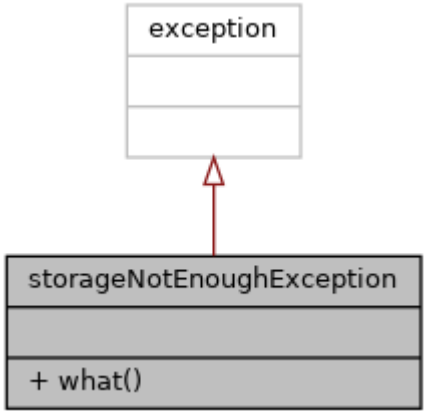
Class nameAlreadyTakenException Inheritance Diagram



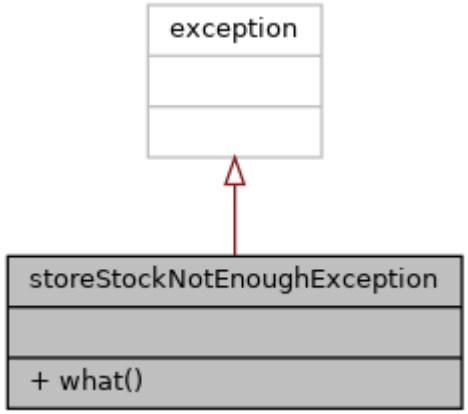
Class nameHasSpacesException Inheritance Diagram



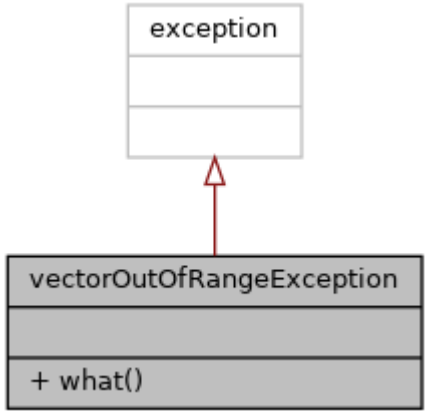
Class storageNotEnoughException Inheritance Diagram



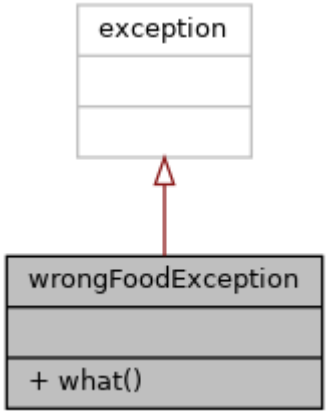
Class storeStockNotEnoughException Inheritance Diagram



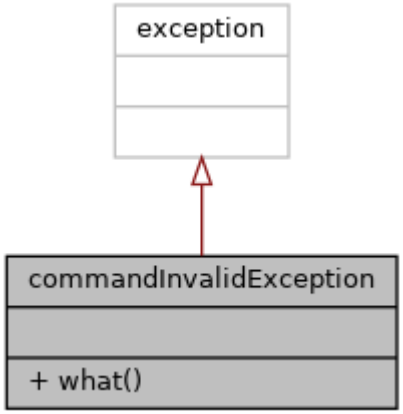
Class vectorOutOfRangeException Inheritance Diagram



Class wrongFoodException Inheritance Diagram



Class commandInvalidException Inheritance Diagram



c. Class Design

1. Item

Kelas item banyak menerapkan inheritance, polymorphism, operator overloading, dan function overloading dalam strukturnya. Inheritance digunakan untuk memperluas fungsionalitas ke child class seperti Plant, Animal, Building, dan Product. Polymorphism memungkinkan penggunaan objek dengan tipe Item untuk menampilkan rincian yang sesuai dengan jenisnya, seperti halnya dengan metode printDetails() dan slowPrintDetails(). Operator overloading dimanfaatkan untuk menambah fungsionalitas tambahan, seperti pada operator<< untuk output stream dan operator+=(int) untuk menambahkan berat hewan dan umur tumbuhan. Function overloading juga digunakan untuk menyediakan fungsionalitas tambahan dalam beberapa konteks, seperti operator<< untuk SlowPrinter.

Kelebihan dari desain kelas Item ini adalah kemampuannya untuk dengan mudah mengelola koleksi berbagai jenis Item dalam Inventory tanpa perlu membuat tipe koleksi yang berbeda untuk setiap jenis. Hal ini memungkinkan fleksibilitas yang besar dalam pengelolaan inventory dalam aplikasi. Namun, kelemahannya adalah rawannya terjadi pengecualian secara abnormal karena beberapa metode dapat melemparkan exception jika digunakan secara tidak teliti.Oleh karena itu, dibutuhkan penanganan exception yang cermat untuk menggunakan kelas Item ini secara efektif.

2. Player

Kelas Player dibuat sebagai kelas dasar yang menggambarkan entitas pemain dalam program. Desain ini dipilih untuk memberikan fleksibilitas dan modularitas dalam pengembangan permainan dengan menerapkan prinsip-prinsip pemrograman berorientasi objek. Dengan menggunakan konsep inheritance dan polymorphism, kelas ini memungkinkan berbagai jenis pemain seperti Walikota, Petani, dan Peternak untuk memiliki perilaku dan atribut yang berbeda sesuai dengan peran masing-masing.

Salah satu alasan utama dalam pemilihan desain ini adalah untuk meminimalkan duplikasi kode dan memperjelas struktur hirarki kelas. Dengan mewarisi atribut dan metode dasar dari kelas Player, kelas-kelas turunan dapat dengan mudah menambahkan perilaku khusus yang sesuai dengan perannya dalam permainan. Selain itu, struktur data seperti map digunakan untuk menyimpan data pemain secara global, sehingga memungkinkan akses yang cepat dan efisien ke informasi pemain.

Meskipun desain ini memiliki banyak kelebihan, seperti fleksibilitas dan kode yang reusable, ada juga beberapa kekurangan yang perlu diperhatikan. Salah satunya adalah ketergantungan antara kelas dasar dan kelas-kelas turunan, yang dapat membuat perubahan di kelas dasar sulit untuk dikelola dan dapat memengaruhi banyak bagian dari kode. Hal ini dapat meningkatkan kompleksitas pemeliharaan dan pengembangan program secara keseluruhan.

3. Inventory

Kelas Inventory merupakan kelas yang bertanggung jawab untuk mengelola inventaris dalam program. Desain ini dipilih untuk memberikan fleksibilitas dalam menyimpan berbagai jenis item seperti Item, Plant, dan Animal dalam suatu struktur data yang terorganisir. Melalui penggunaan template, kelas ini dapat digunakan untuk membuat inventaris dengan tipe data yang berbeda-beda tanpa perlu menulis ulang kode.

Salah satu alasan utama dalam pemilihan desain ini adalah untuk memungkinkan penggunaan ulang kode dan penyimpanan item yang dinamis. Dengan menggunakan struktur data seperti vector dan map, kelas ini dapat mengelola item-item dengan efisien dan mudah diakses. Selain itu, penggunaan template memungkinkan kelas ini untuk digunakan dalam berbagai konteks tanpa perlu menulis ulang kode.

Meskipun desain ini memiliki banyak kelebihan, seperti fleksibilitas dan penggunaan ulang kode, ada juga beberapa kekurangan yang perlu diperhatikan. Salah satunya adalah kompleksitas dalam pengelolaan dan pemeliharaan inventaris yang besar. Hal ini dapat mengakibatkan kesulitan dalam debugging dan pengembangan fitur-fitur baru yang melibatkan inventory.

4. File Manager

Kelas FileManager bertanggung jawab untuk mengelola operasi baca dan tulis data dari dan ke file eksternal. Desain kelas ini didasarkan pada prinsip kejelasan dan pemisahan tanggung jawab. Semua metode dalam kelas ini didefinisikan sebagai metode statis karena kelas ini bertindak sebagai utilitas dan tidak memerlukan pembuatan objek khusus untuk digunakan.

Kelebihan dari desain ini adalah kesederhanaannya. Dengan hanya menggunakan metode statis, kelas ini menjadi sederhana dan mudah dimengerti. Selain itu, desain ini juga memberikan modularitas, memisahkan logika operasi file ke dalam kelas terpisah sehingga mudah pemeliharaan dan pengembangan di masa depan.

Namun, ada beberapa kekurangan yang mungkin dimiliki desain ini. Ketergantungan yang kuat pada metode statis dapat membuat pengujian dan pengembangan menjadi lebih rumit, terutama dalam hal pengujian. Selain itu, kelas ini mungkin

memiliki keterbatasan dalam fleksibilitas karena semua metodenya didefinisikan sebagai metode statis, sehingga membuat perubahan besar dalam logika atau struktur kelas menjadi lebih sulit dilakukan.

5. Store

Kelas Store bertanggung jawab untuk menyimpan dan mengelola inventaris barang di toko dalam program. Desain kelas ini didasarkan pada prinsip kejelasan dan kohesi. Dengan hanya memiliki satu tanggung jawab, yaitu mengelola data inventaris toko, kelas ini tetap sederhana dan mudah dimengerti.

Kelebihan dari desain ini adalah kesederhanaan dan fokus yang jelas pada tanggung jawab tunggalnya. Hal ini membuat kelas ini mudah dipahami dan dikelola. Penggunaan variabel statis memungkinkan data inventaris toko untuk diakses dan dimodifikasi secara global dalam program, yang merupakan keuntungan dalam hal konsistensi dan kemudahan pengelolaan data.

Namun, ada kekurangan potensial dalam desain ini. Ketergantungan pada variabel statis dapat membuat pengujian dan pengembangan menjadi lebih rumit, terutama dalam hal pengujian unit dan pengujian integrasi. Selain itu, desain ini mungkin memiliki keterbatasan dalam hal fleksibilitas, karena variabel statis hanya dapat diakses melalui metode statis, yang membuat perubahan besar dalam logika atau struktur kelas menjadi lebih sulit dilakukan.

Kelas ini memiliki beberapa metode statis untuk menam dan mengurangi inventaris toko. Metode addStoreData digunakan untuk menambahkan jumlah barang tertentu ke inventaris toko, sedangkan metode reduceStoredata digunakan untuk mengurangi jumlah barang tertentu dari inventaris toko. Variabel statis StoreData digunakan untuk menyimpan jumlah barang yang tersedia di toko.

6. Slow Printer

Kelas SlowPrinter bertanggung jawab untuk mencetak teks ke output dengan jeda waktu tertentu antara setiap karakter atau baris teks. Desain kelas ini dipilih untuk memberikan fleksibilitas dalam mencetak teks dengan kecepatan yang dapat disesuaikan.

Kelebihan dari desain ini adalah kemampuannya untuk mengontrol kecepatan pencetakan teks dengan mudah melalui pengaturan jeda waktu. Ini memberikan pengguna kemampuan untuk mengatur kecepatan pencetakan sesuai dengan preferensi atau kebutuhan mereka. Selain itu, dengan menggunakan operasi overload pada operator<<, kelas ini dapat digunakan dengan mudah dan intuitif seperti penggunaan objek ‘cout’ dari STL C++.

Namun, ada kekurangan potensial dalam desain ini. Salah satunya adalah bahwa pengguna bahwa pengguna harus secara eksplisit membuat objek ‘SlowPrinter’ untuk mencetak teks dengan kecepatan tertentu. Ini mungkin membutuhkan lebih banyak kode daripada menggunakan metode cetak bawaan seperti ‘cout’. Selain itu, jika pengguna lupa untuk mengembalikan kecepatan pencetakan ke nilai default setelah mengatur nilai yang berbeda, hal itu dapat menyebabkan inkonsistensi dalam program.

Kelas ini memiliki beberapa metode untuk mengatur dan mengontrol kecepatan pencetakan teks. Metode operator<< digunakan untuk mencetak teks dengan kecepatan tertentu, sedangakn metode getDelay, getmult, setDelay, setMult, resetDelay, dan resetMult digunakan untuk mengatur dan mengontrol jeda waktu antara karakter atau baris teks yang dicetak.

7. Game

Kelas Game bertanggung jawab untuk mengatur dan menjalankan permainan “Harvest Bitcoin Gulden”. Desain kelas ini dipilih untuk memisahkan tanggung jawab pengaturan awal permainan dan pengelolaan jalannya permainan.

Kelas ini memiliki beberapa kelebihan. Pertama, dengan memisahkan inisialisasi permainan dan perulangan permainan ke dalam metode Initialize dan Start secara terpisah, kelas Game menjadi lebih mudah dipahami dan terorganisir. Ini juga memungkinkan untuk memisahkan kode-kode yang berbeda ke dalam metode-metode terpisah, sehingga memudahkan pengembangan dan pemeliharaan kode. Selain itu, dengan menggunakan kelas FileManager untuk membaca dan menulis data permainan, kelas Game tidak terlalu bergantung pada operasi file secara langsung, sehingga memudahkan pengujian dan penggantian implementasi file sistem jika diperlukan.

Namun, ada beberapa potensi kekurangan dalam desain ini. Salah satunya adalah bahwa perulangan permainan di dalam metode Start tidak memiliki batasan waktu, yang dapat menyebabkan permainan terjebak dalam loop tak berujung jika tidak ada pemain yang memenangkan permainan. Selain itu, kelas Game memiliki ketergantungan terhadap kelas Player dan SlowPrinter, yang dapat membuatnya sulit untuk diuji secara terpisah.

Kelas ini memiliki dua metode utama, yaitu Initialize dan Start, yang bertanggung jawab untuk mengatur awal permainan dan menjalankan permainan. Metode Initialize bertanggung jawab untuk mengatur awal permainan dan menjalankan permainan. Metode Initialize bertanggung jawab untuk membaca data permainan dan mengatur setup awal, sementara metode Start bertanggung jawab untuk memulai dan menjalankan perulangan permainan hingga ada pemain yang memenangkan permainan.

8. Exception

Kelas Exception ini menerapkan konsep inheritance dan polymorphism dalam implementasinya. Dengan menggunakan inheritance, kelas-kelas exception spesifik seperti vectorOutOfRangeException, moneyNotEnoughException, storageNotEnoughException, dan lainnya mengambil sifat-sifat dari kelas dasar Exception. Ini memungkinkan kita untuk menangkap exception dengan hanya menggunakan kelas dasarnya, tanpa perlu menyebutkan semua kemungkinan tipe exception yang terjadi.

Kelebihan dari implementasi ini adalah kemudahan dalam menangani exception, karena cukup dengan menangkap exception berdasarkan tipe dasarnya saja pada blok try-catch. Misalnya, ketika menangkap Exception, kita sudah dapat menangkap semua exception spesifik yang diwarisi darinya, tanpa perlu menuliskan catch block untuk setipa tipe exception spesifik secara terpisah. Selain itu, implementasi ini memudahkan dalam penggunaan inheritance untuk membuat kelas exception yang lebih spesifik dalam penggunaannya, seperti ItemException.

Namun, ada kekurangan dari implementasi ini. Salah satunya adalah sulitnya melacak pesan error yang dihasilkan ketika exception dilemparkan. Karena pesan error yang dihasilkan ketika exception dilemparkan. Karena pesan error yang dihasilkan terdapat dalam metode what() pada setiap kelas exception, sulit untuk langsung mengetahui sumber error yang tepat tanpa melihat implementasi dari masing-masing kelas exception.

Meskipun demikian, dengan menerapkan konsep inheritance dan polymorphism, kelas Exception ini menjadi lebih fleksibel dan mudah digunakan dalam menangani error yang mungkin terjadi dalam program.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Dalam konteks program ini, konsep pemrograman berorientasi objek, yaitu inheritance dan polymorphism, diterapkan secara luas untuk mengorganisir dan memodularisasi kode dengan baik

Pertama, pada kelas Player, inheritance digunakan untuk membuat hubungan hierarki antara kelas Player sebagai kelas induk dan kelas turunan seperti Walikota, Petani, dan Peternak. Ini memungkinkan kelas turunan untuk mewarisi atribut dan perilaku yang sama dari kelas induk, seperti nama, uang, dan inventaris. Dengan cara ini, kode yang redundan dapat dihindari karena sifat yang sama dapat diwariskan dari kelas induk.

Selain itu, konsep polymorphism digunakan di kelas Player dengan mendefinisikan metode-metode virtual yang kemudian diimplementasikan ulang di kelas turunan. Misalnya, metode hitungPajak dideklarasikan sebagai metode virtual murni di kelas Player dan diimplementasikan ulang

di kelas Walikota, Petani, dan Peternak. Ini memungkinkan pemanggilan metode yang sesuai dengan tipe objek pada saat runtime, memberikan fleksibilitas dalam penggunaan kelas-kelas tersebut.

Sementara itu, kelas Store tidak menggunakan inheritance atau polymorphism karena fungsinya yang sederhana untuk menyimpan data inventaris toko. Tidak ada kebutuhan untuk hierarki kelas atau perilaku yang berbeda-beda berdasarkan tipe objek.

Di sisi lain, kelas SlowPrinter menggunakan polymorphism dengan mendefinisikan berbagai operator << untuk menangani berbagai jenis data dengan cara yang seragam. Dengan cara ini, kelas SlowPrinter dapat digunakan dengan mudah untuk mencetak berbagai jenis data dengan kecepatan yang disesuaikan, meningkatkan fleksibilitas penggunaannya.

Secara keseluruhan, penggunaan inheritance dan polymorphism membantu mengorganisir kode dengan baik, mengurangi duplikasi, dan meningkatkan fleksibilitas dalam penggunaan kelas-kelas tersebut.

```
class vectorOutOfRangeException {
    public:
        const char* what() {
            return "Indeks invalid";
        };
};

class moneyNotEnoughException {
    public:
        const char* what() {
            return "Uang Anda tidak mencukupi";
        };
};

class storageNotEnoughException {
    public:
        const char* what() {
            return "Kapasitas Anda tidak mencukupi";
        };
};
```

```
class Walikota : public Player {
    friend class FileManager;
    protected:
        // Static variables

        // Instance variables

    public:
        // Constructors
        Walikota(string, int, int);

        // Walikota command methods
        bool checkCommandValid(string);

        // Walikota commands
        void help();
        void pungutPajak();
        void bangun();
        void beli();
        void jual();
        void tambahPemain();

        // Getters
        string getPlayerType();

        // Setters

        // Instance methods
        int hitungPajak();
};
```

```
class Petani : public Player {
    friend class FileManager;
    protected:
        // Static variables

        // Instance variables
        Inventory<Plant> farm;

    public:
        // Constructors
        Petani(string, int, int);

        // Petani command methods
        bool checkCommandValid(string);

        // Petani commands
        void help();
        void cetakLadang();
        void tanam();
        void beli();
        void jual();
        void panen();

        // Getters
        string getPlayerType();

        // Setters
        void insertToFarm(Plant *, string);
        void incrementAllPlants();

        // Instance methods
        int hitungPajak();
};
```

2.2. Method/Operator Overloading

Dalam konteks program ini, konsep pemrograman berorientasi objek, yaitu method/operator overloading, digunakan untuk memperluas kemampuan objek untuk berinteraksi dengan operator atau fungsi bawaan bahasa C++.

Misalnya, dalam kelas SlowPrinter, terdapat method/operator overloading untuk operator ‘<<’ yang memungkinkan objek SlowPrinter untuk digunakan dengan sintaks mirip dengan ‘cout’. Method/operator overloading ini memungkinkan objek SlowPrinter menerima berbagai jenis data dan mencetaknya dengan kecepatan tertentu yang telah ditentukan. Dengan demikian, penggunaan SlowPrinter menjadi lebih intuitif dan serupa dengan penggunaan ‘cout’, sehingga meningkatkan kejelasan dan kegunaannya.

Di kelas Player, terdapat juga method/operator overloading untuk operator ‘<<’, yang memungkinkan objek ‘Player’ dicetak menggunakan SlowPrinter. Ini meningkatkan fleksibilitas dalam cara objek Player dapat diakses dan dicetak.

Penggunaan konsep method/operator overloading pada kelas-kelas tersebut menghasilkan kode yang lebih mudah dimengerti, lebih seragam, dan lebih mudah digunakan. Ini membantu meningkatkan ekspresivitas kode, mengurangi kebingungan, dan memungkinkan penggunaan objek dengan sintaks yang lebih alami dan intuitif.

```cpp
SlowPrinter& SlowPrinter::operator<<(const char* str) {
    for (size_t i = 0; i < strlen(str); ++i) {
        out << str[i] << std::flush;
        this_thread::sleep_for(chrono::milliseconds(delay));
    }
    this_thread::sleep_for(chrono::milliseconds(delay * delay_multiplier));
    return *this;
}

SlowPrinter& SlowPrinter::operator<<(const std::string& str) {
    for (const auto& c : str) {
        out << c << std::flush;
        std::this_thread::sleep_for(std::chrono::milliseconds(delay));
    }
    this_thread::sleep_for(chrono::milliseconds(delay * delay_multiplier));
    return *this;
}

SlowPrinter& SlowPrinter::operator<<(ostream& (*manipulator)(ostream&)) {
    manipulator(out);
    return *this;
}
```

2.3. Template & Generic Classes

Dari kelas-kelas yang kami gunakan, terdapat penerapan konsep template dan generic class pada kelas Inventory. Kelas Inventory adalah contoh dari penggunaan template untuk membuat struktur data umum yang dapat digunakan dengan berbagai jenis objek, dalam kasus ini Item, Plant, dan Animal.

Dengan menggunakan template, kelas Inventory dapat diparameterisasi dengan tipe data apa pun. Ini memungkinkan kelas Inventory untuk digunakan dengan berbagai jenis objek tanpa perlu menuliskan kelas yang berbeda untuk setiap jenisnya. Sebagai contoh, dengan menggunakan kelas Inventory, kita dapat membuat inventaris untuk menyimpan Item, Plant, atau Animal dengan cara yang serupa.

Keuntungan utama dari penggunaan konsep template dan generic class pada kelas Inventory adalah peningkatan reusability dan fleksibilitas kode. Karena kelas Inventory adalah templatized, ia dapat digunakan dengan berbagai jenis objek tanpa perlu menulis ulang kode atau membuat kelas yang serupa untuk setiap jenis objek. Hal ini memungkinkan pengembang untuk menulis kode yang lebih efisien dan mudah dikelola.

Selain itu, penggunaan konsep template dan generic class juga meningkatkan ekspresivitas kode. Dengan menggunakan template, kita dapat membuat kelas yang lebih umum dan abstrak, yang memungkinkan kita untuk menyampaikan tujuan dan fungsionalitas kelas dengan cara yang lebih jelas dan mudah di mengerti.

```cpp
template <class T>
void Inventory<T>::printInventory(){
    SlowPrinter& sc = *(SlowPrinter::getSlowPrinter());
    sc.setMult(0);
    sc.setDelay(sc.getDelay() - 12);

    int lineLength = this->cols * 6;
    string title = " Penyimpanan ";
    int totalPadding = lineLength - title.length();
    int leftPadding = totalPadding / 2;
    int rightPadding = totalPadding - leftPadding;

    title = string(4, ' ') + string(leftPadding, '=') + title + string(rightPadding, '=');
    sc << title << endl;

    sc << " ";
    for (int i = 0; i < this->cols; i++){
        sc << "      " << getColString(i);
    }
    sc << "    " << endl;

    for (int i = 0; i < this->rows; i++) {
        sc << "    +-----";
        for (int j = 0; j < this->cols - 1; ++j) {
            sc << "+-----";
        }
        sc << "+" << endl;

        cout << setw(2) << setfill('0') << i + 1;
        cout << " |";

        for (int j = 0; j < this->cols; ++j) {
            string key = getColString(j) + getRowString(i);
            if (storage.find(key) != storage.end()) {
                cout << " " << BOLD CYAN << storage[key]->getCode() << BOLD YELLOW << " |";
            } else {
                cout << "      |";
            }
        }
        cout << endl;
    }

    sc << "    +-----";
    for (int j = 0; j < this->cols - 1; ++j) {
        sc << "+-----";
    }
    sc << "+" << endl;

    sc.resetMult();
    sc.resetDelay();

    int emptySlots = getEmptySlotsCount();
    sc << BOLD CYAN << "Total slot kosong: " << BOLD YELLOW << emptySlots << endl;
}
```

2.4. Exception

Implementasi *exception* kami gunakan untuk *error handling.* Hal ini mengurangi secara drastis kemungkinan program *crash* atau berhenti secara paksa karena kesalahan input atau *error* yang tidak ter-*handle*. *Exception* juga mengurangi pengulangan kode, karena satu *exception* bisa dipakai oleh beberapa fitur yang memiliki *error* serupa contohnya *index out of range.* Implementasi *exception* membatasi penggunaan *conditional statements* untuk *error handling*, yang dapat mengurangi kompleksitas kode dan membuat kode lebih mudah dibaca.

2.5. C++ Standard Template Library

Dalam tugas ini, kami menggunakan dua STL, yaitu vector dan map. Vector kami gunakan untuk menyimpan data ketika ukurannya tidak diketahui secara pasti. Contoh penggunaannya adalah untuk menyimpan input slot saat perintah jual dan beli. Pada kasus tersebut, belum diketahui secara pasti berapa slot yang akan dimasukkan pemain untuk dilakukan perintah jual dan beli. Karena itu, kami menggunakan vector untuk menyimpan masukan slot-slot ini.

Sedangkan penggunaan map dipakai secara spesifik untuk menyimpan pasangan kunci-nilai. Contohnya adalah untuk menyimpan data barang yang dijual di toko serta jumlah stok barang tersebut, kami menggunakan pasangan kunci-nilai string dan integer. Alasan kami menggunakannya adalah karena aksesnya yang efisien. Untuk mendapatkan jumlah stok dari suatu barang, kami hanya perlu untuk menggunakan nama barang tersebut pada map.

```cpp
void Walikota::jual() {
    // Initialize slowprinter
    SlowPrinter& sc = *(SlowPrinter::getSlowPrinter());

    // Run command
    sc << BOLD CYAN << "Berikut merupakan penyimpanan Anda" << RESET << endl;
    this->cetakPenyimpanan();

    vector<string> slots;
    string slotTemp, slotInput;
    getline(cin, slotInput);

petak:
    int profit = 0;
    try {
    bool valid = false;
    while (!valid) {
        sc << BOLD GREEN << "Silahkan pilih petak yang ingin Anda jual!" << endl;
        sc << "Petak: " << RESET;
        getline(cin, slotInput);
        stringstream ss(slotInput);
        while (std::getline(ss, slotTemp, ',')) {
            slotTemp.erase(0, slotTemp.find_first_not_of(' '));
            slotTemp.erase(slotTemp.find_last_not_of(' ') + 1);
            slots.push_back(slotTemp);
        }
        bool empty = false;

        for (int i = 0; i < int(slots.size()); i++) {
            if (this->inventory.getSlotStatus(slots.at(i)) == NULL) {
                sc << BOLD RED << "Slot " << slots.at(i) << " kosong" << endl;
                empty = true;
                while (!slots.empty()) {
                    slots.pop_back();
                }
                break;
            }
        }
        if (empty == false) {
            valid = true;
        }
    }

    Item* itemTemp;

        for (int i = 0; i < int(slots.size()); i++) {
            itemTemp = this->inventory.getItem(slots.at(i));
            this->inventory.DeleteItemAt(slots.at(i));
            Store::addStoreData(itemTemp->getName());
            this->money += itemTemp->getPrice();
            profit += itemTemp->getPrice();
        }
    }
    catch (invalidSlotException e) {
        sc << e.what() << endl;
        while (!slots.empty()) {
            slots.pop_back();
        }
        goto petak;
    }
    catch (invalid_argument &e) {
        sc << BOLD RED << "Slot yang Anda masukkan tidak valid" << endl;
        while (!slots.empty()) {
            slots.pop_back();
        }
        goto petak;
    }

    sc << BOLD MAGENTA << "Barang Anda berhasil dijual! Uang Anda bertambah " << BOLD YELLOW << profit << BOLD MAGENTA
<< " gulden!" << RESET << endl;
}
```

```cpp
void Walikota::beli() {
    // Initialize slowprinter
    SlowPrinter& sc = *(SlowPrinter::getSlowPrinter());

    // Run command
    sc << BOLD MAGENTA << "Selamat datang di toko!!" << endl; sc.setMult(0);
    sc << BOLD GREEN << "Berikut merupakan benda yang dapat Anda Beli" << endl;
    vector<string> itemList;
    map<string, int>::iterator itStore;
    map<string, int> storeTemp = Store::getStoreData();
    Item *item;
    int num = 1;
    sc.setMult(0);
    sc.setDelay(sc.getDelay() - 15);
    for (itStore = storeTemp.begin(); itStore != storeTemp.end(); itStore++) {
        if (itStore->second == 0 || Item::getItemData()[itStore->first]->isBuilding()) {
            continue;
        }
        item = Item::getItemData()[itStore->first];
        sc << BOLD CYAN << num << ". " << BOLD YELLOW << itStore->first << " - " << BOLD GREEN << item->getPrice() <<
BOLD CYAN << " (" << itStore->second << ")" << endl;
        itemList.push_back(itStore->first);
        num++;
    }

    map<string, Item*>::iterator itItem;
    map<string, Item*> itemTemp = Item::getItemData();
    for (itItem = itemTemp.begin(); itItem != itemTemp.end(); itItem++) {
        sc << BOLD CYAN << num << ". " << BOLD YELLOW << itItem->first << " - " << BOLD GREEN << itItem->second-
>getPrice() << endl;
        itemList.push_back(itItem->first);
        num++;
    }
    sc.resetMult();
    sc.resetDelay();

    sc << endl << BOLD MAGENTA << "Uang Anda: " << BOLD YELLOW << this->money << endl;
    sc << BOLD MAGENTA << "Slot penyimpanan tersedia: " << BOLD YELLOW << this->inventory.getEmptySlotsCount() << endl
<< endl;

    int buy;
    int q;
    bool valid = false;

    while (!valid) {
        sc << BOLD GREEN << "Barang yang ingin dibeli: " << RESET;
        cin >> buy;
        sc << BOLD GREEN << "Kuantitas: " << RESET;
        cin >> q;

        try {
            if (buy <= 0 || buy >= int(itemList.size()-1)) {
                throw vectorOutOfRangeException();
            }
            else if (this->money < Item::getItemData()[itemList.at(buy-1)]->getPrice() * q) {
                throw moneyNotEnoughException();
            }
            else if (this->inventory.getEmptySlotsCount() < q) {
                throw storageNotEnoughException();
            }
            else if (storeTemp[itemList.at(buy-1)] < q) {
                throw storeStockNotEnoughException();
            }
            valid = true;
        }
        catch (vectorOutOfRangeException e) {
            sc << e.what() << endl;
            sc << BOLD RED << "Masukkan barang yang ingin dibeli diantara" << BOLD YELLOW << "1" << BOLD RED << "
sampai " << BOLD YELLOW << itemList.size() << RESET << endl;
        }
        catch (moneyNotEnoughException e) {
            sc << e.what() << endl;
        }
        catch (storageNotEnoughException e) {
            sc << e.what() << endl;
        }
        catch (storeStockNotEnoughException e) {
            sc << e.what() << endl;
        }
    }

    this->money -= Item::getItemData()[itemList.at(buy-1)]->getPrice();
```

2.6. Konsep OOP lain

Pertama, Abstract Base Class digunakan dalam kelas Player, yang bertindak sebagai kerangka dasar untuk berbagai tipe permian seperti Walikota, Petani, dan Peternak. Player memiliki metode virtual murni checkCommandValid, yang memungkinkan kelas turunannya untuk mengimplementasikan perilaku sesuai kebutuhan mereka sendiri

Kedua, aggregation digunakan dalam kelas Inventory<Item> di dalam kelas Player. Di sini, Player memiliki akses ke Inventory, tetapi tidak sepenuhnya memiliki objek tersebut. Ini memungkinkan untuk mengelompokkan item dalam inventory tanpa harus sepenuhnya menggabungkan struktur data inventory ke dalam kelas Player.

Ketiga, composition terlibat dalam penggunaan Inventory<Plant> dan Inventory<Animal> di dalam kelas Petani dan Peternak. Objek inventory ini menjadi bagian intrinsik dari objek Petani dan Peternak, sehingga ketika objek pemain dihapus, inventory juga ikut terhapus.

Penerapan konsep Abstract Base Class, aggregation, dan composition ini membantu dalam membuat desain yang modular dan terorganisir dengan baik. Ini memisahkan tanggung jawab dengan jelas, memungkinkan untuk mengelola hubungan antar objek dengan lebih baik, dan membuat kode lebih mudah dipelihara dan dimodifikasi di masa mendatang.

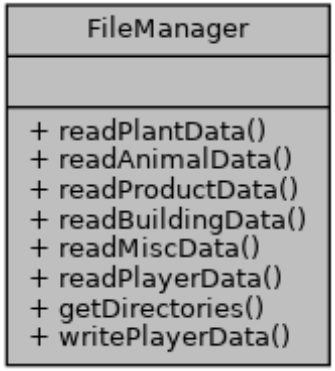
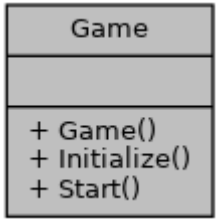
3. Bonus Yang dikerjakan

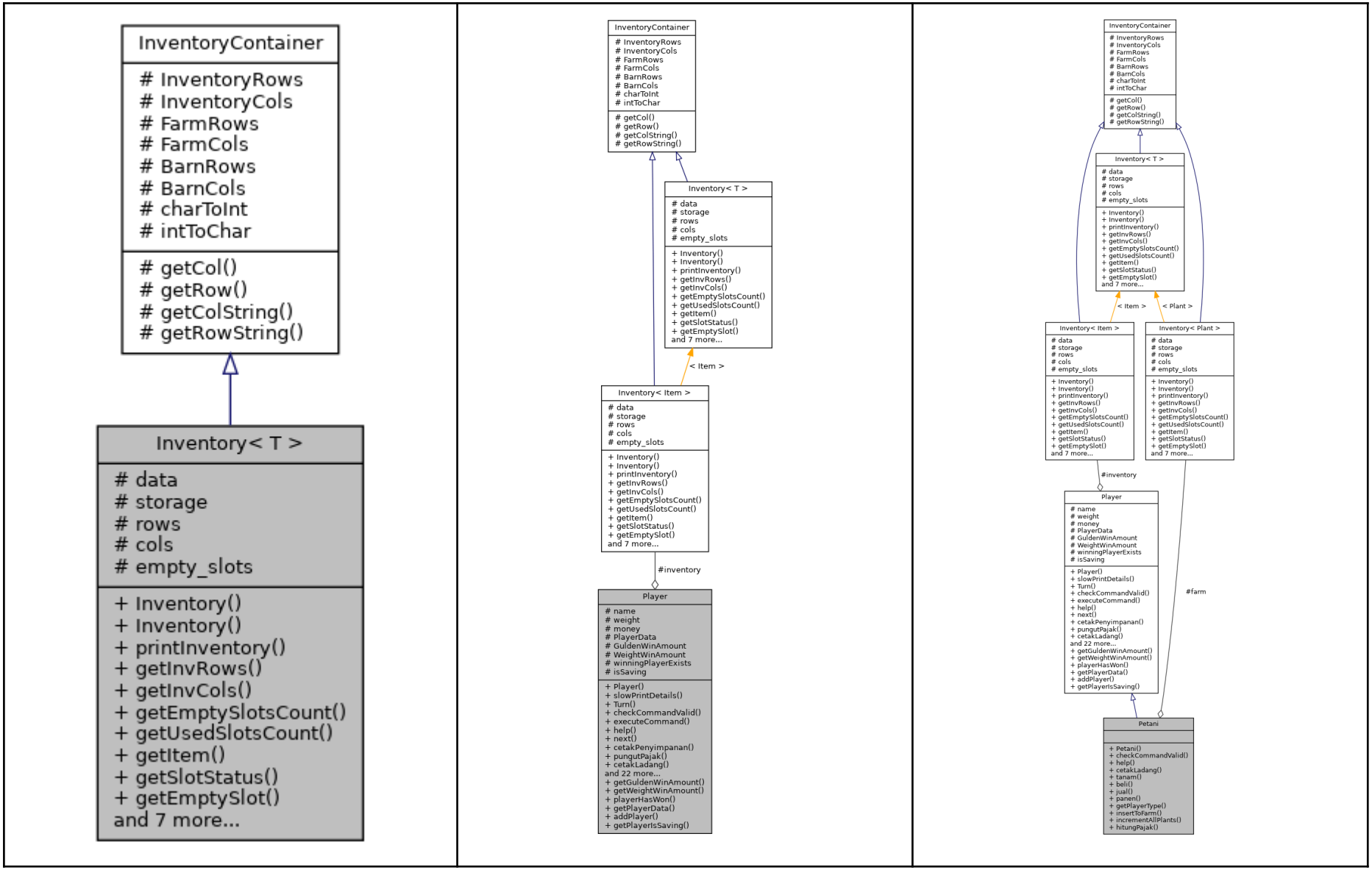
3.1. Bonus yang diusulkan oleh spek

3.1.1. Diagram Sistem

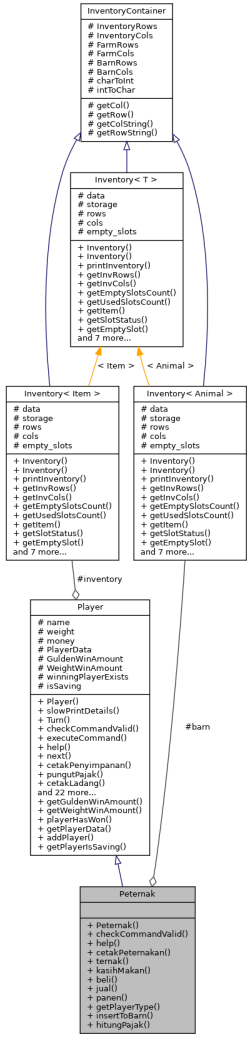
Selain Inheritance diagram, kami juga membuat Collaboration diagram. Perbedaannya dengan Inheritance diagram adalah Inheritance diagram menggambarkan hierarki kelas dan hubungan pewarisan antara kelas-kelas, dengan panah menunjukkan arah pewarisan dari kelas induk ke kelas turunan. Diagram ini membantu memvisualisasikan bagaimana kelas turunan mewarisi perilaku dan atribut dari kelas induk. Di sisi lain, Collaboration diagram menunjukkan interaksi antara objek-objek dan kelas-kelas dalam sistem untuk mencapai suatu tujuan. Diagram ini terdiri dari objek-objek atau kelas-kelas, dengan panah menunjukkan pesan atau interaksi antara objek-objek tersebut. Collaboration diagram membantu memahami bagaimana objek-objek bekerja sama dalam situasi tertentu, membantu menggambarkan aliran kontrol dan data dalam sistem secara lebih terperinci.

Class FileManager **Collaboration Diagram**	**Class Game** **Collaboration Diagram**	**Class InventoryContainer** **Collaboration Diagram**

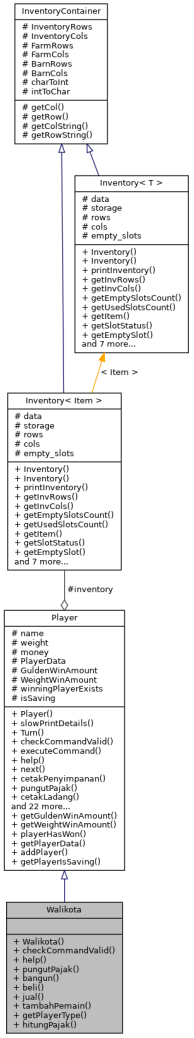
		
Class Inventory<T> Collaboration Diagram	**Class Player Collaboration Diagram**	**Class Petani Collaboration Diagram**


InventoryContainer
InventoryRows
InventoryCols
FarmRows
FarmCols
BarnRows
BarnCols
charToInt
intToChar
getCol()
getRow()
getColString()
getRowString()
Inventory< T >
data
storage
rows
cols
empty_slots
+ Inventory()
+ Inventory()
+ printInventory()
+ getInvRows()
+ getInvCols()
+ getEmptySlotsCount()
+ getUsedSlotsCount()
+ getItem()
+ getSlotStatus()
+ getEmptySlot()
and 7 more...
< Item >
Inventory< Item >
Player
name
weight
money
PlayerData
GuldenWinAmount
WeightWinAmount
winningPlayerExists
isSaving
+ Player()
+ slowPrintDetails()
+ Turn()
+ checkCommandValid()
+ executeCommand()
+ help()
+ next()
+ cetakPenyimpanan()
+ pungutPajak()
+ cetakLadang()
and 22 more...
+ getGuldenWinAmount()
+ getWeightWinAmount()
+ playerHasWon()
+ getPlayerData()
+ addPlayer()
+ getPlayerIsSaving()
#inventory
< Plant >
Inventory< Plant >
#farm
Petani
+ Petani()
+ checkCommandValid()
+ help()
+ cetakLadang()
+ tanam()
+ beli()
+ jual()
+ panen()
+ getPlayerType()
+ insertToFarm()
+ incrementAllPlants()
+ hitungPajak()

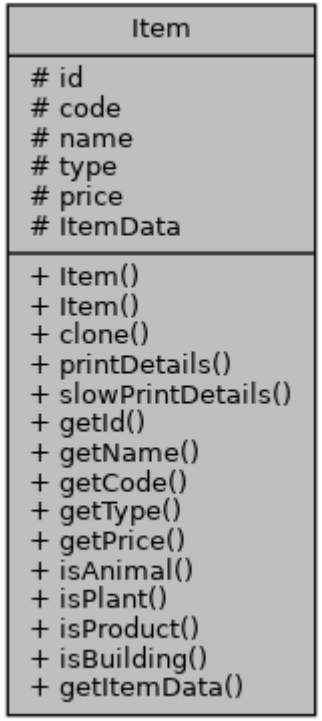
Class Peternak Collaboration Diagram



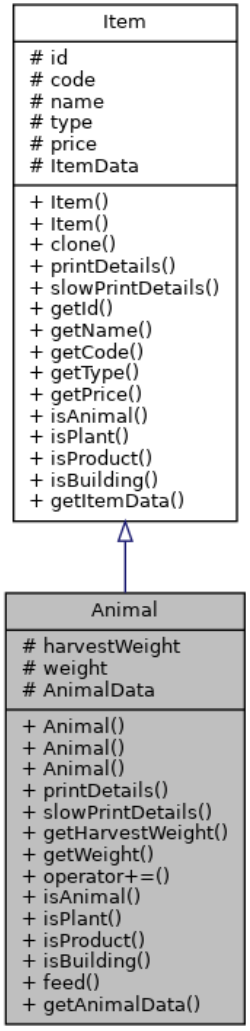
Class Walikota Collaboration Diagram



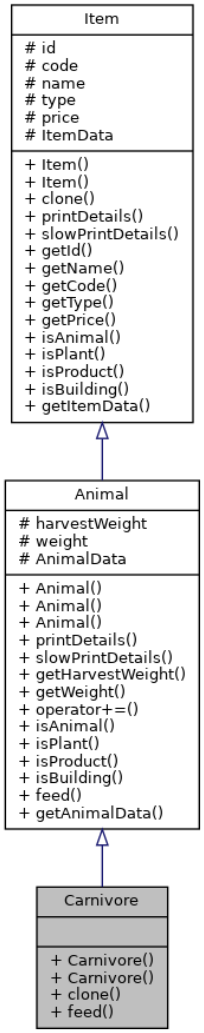
Class Item Collaboration Diagram



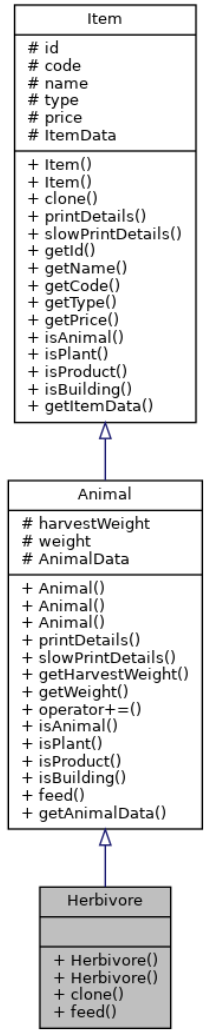
Class Animal
Collaboration Diagram



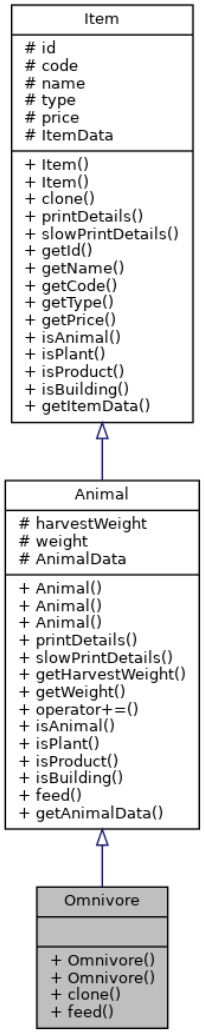
Class Carnivore
Collaboration Diagram



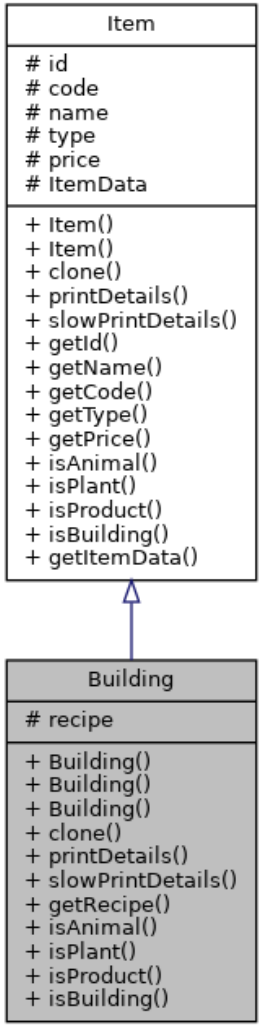
Class Herbivore
Collaboration Diagram



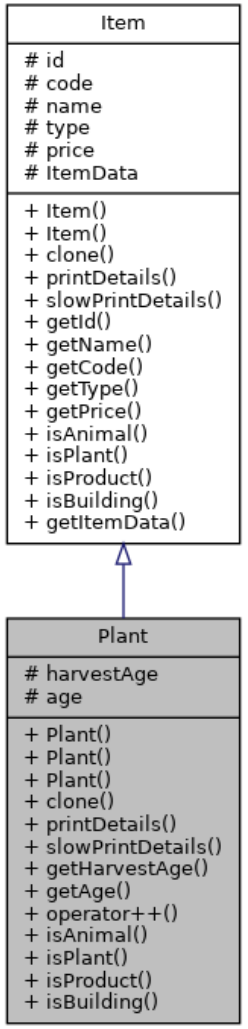
Class Omnivore Collaboration Diagram



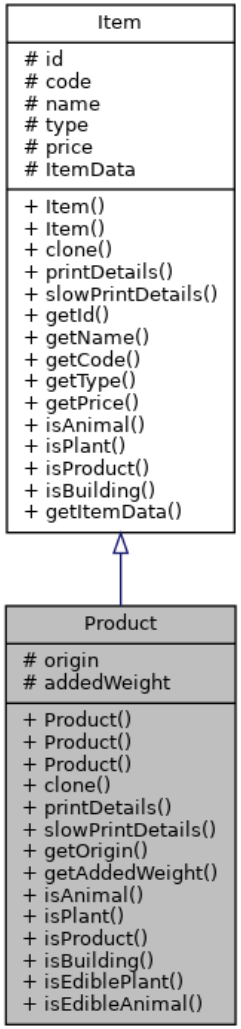
Class Building Collaboration Diagram



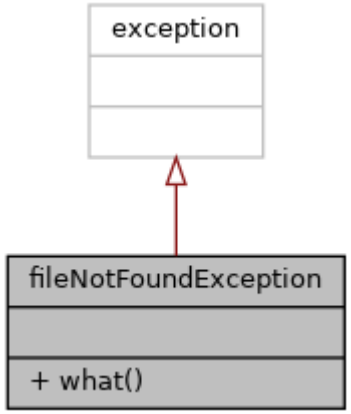
Class Plant Collaboration Diagram



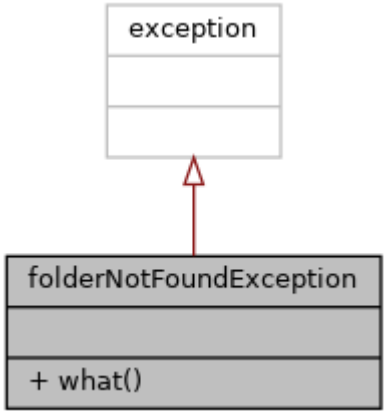
Class Product
Collaboration Diagram



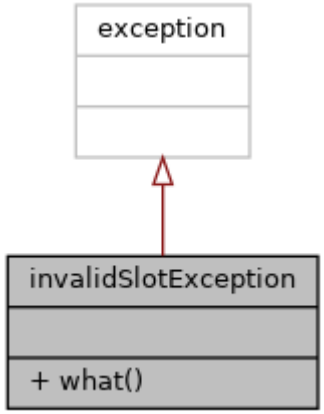
Class fileNotFoundException
Collaboration Diagram



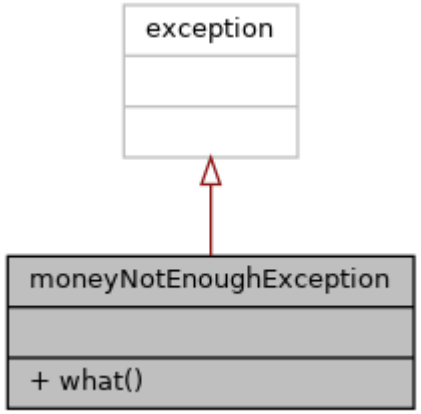
Class folderNotFoundException
Collaboration Diagram



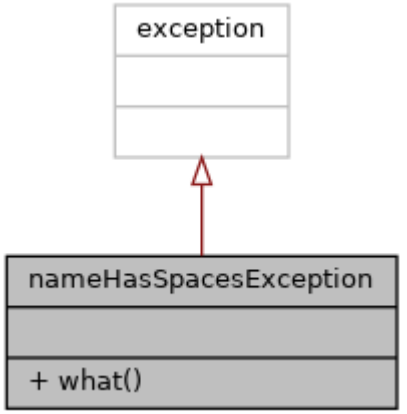
Class invalidSlotException Collaboration Diagram



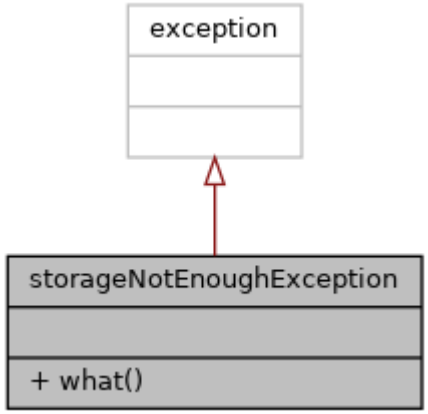
Class moneyNotEnoughException Collaboration Diagram



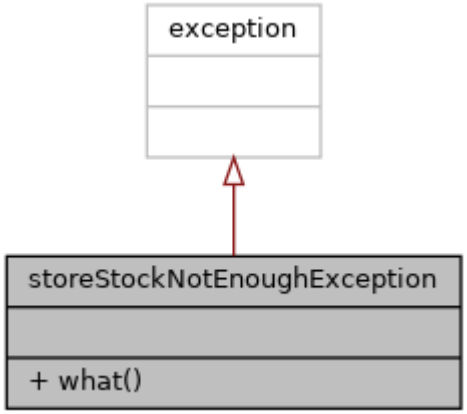
Class nameHasSpacesException Collaboration Diagram



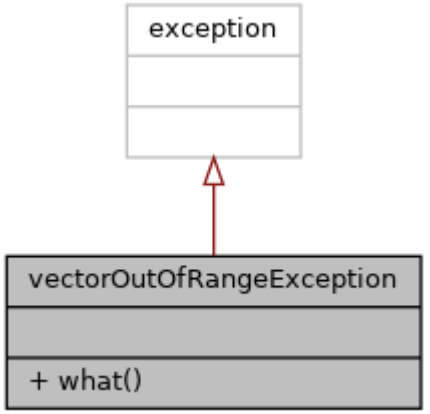
Class storageNotEnoughException Collaboration Diagram



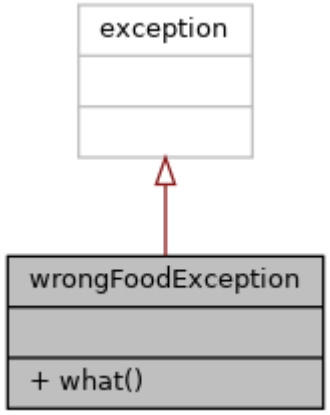
Class storeStockNotEnoughException Collaboration Diagram



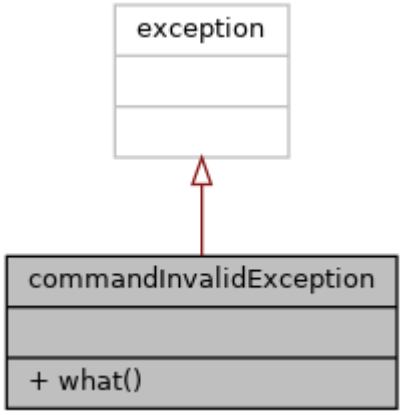
Class vectorOutOfRangeException Collaboration Diagram



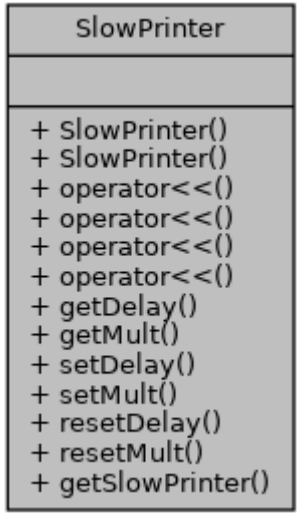
Class wrongFoodException
Collaboration Diagram



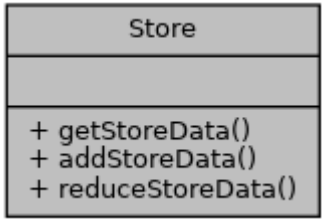
Class commandInvalidException
Collaboration Diagram



Class SlowPrinter
Collaboration Diagram



Class Store
Collaboration Diagram



3.2. Bonus Kreasi Mandiri

3.2.1. Slow Printer

Fitur ini memberikan animasi mencetak pada CLI yang kami terapkan menggunakan konsep OOP. Tujuan kami mengimplementasikan fitur ini adalah untuk memperbagus pencetakan pada permainan karena pencetakan bawaan pada terminal terlihat sangat hambar dan biasa.

```hpp
#ifndef __SLOWPRINTER_HPP
#define __SLOWPRINTER_HPP

#include <iostream>
#include <chrono>
#include <thread>
#include <cstring>

using namespace std;

class SlowPrinter {
    private:
        static SlowPrinter* slowcout;

        ostream& out;
        int delay;
        int delay_multiplier;

        int default_delay;
        int default_mult;


    public:
        SlowPrinter(ostream&, int, int);
        SlowPrinter(const SlowPrinter&);

        static SlowPrinter* getSlowPrinter() {
            return slowcout;
        }

        template<class T>
        SlowPrinter& operator<<(const T& value) {
            for (const auto& c : to_string(value)) {
                out << c << flush;
                this_thread::sleep_for(chrono::milliseconds(delay));
            }
            this_thread::sleep_for(chrono::milliseconds(delay * delay_multiplier));
            return *this;
        }

        SlowPrinter& operator<<(const std::string& str);
        SlowPrinter& operator<<(const char* str);
        SlowPrinter& operator<<(ostream& (*manipulator)(ostream&));

        int getDelay();
        int getMult();

        void setDelay(int);
        void setMult(int);

        void resetDelay();
        void resetMult();

};

#endif
```

```cpp
#include "lib/SlowPrinter.hpp"

SlowPrinter* SlowPrinter::slowcout = new SlowPrinter(cout, 20, 10);

SlowPrinter::SlowPrinter(ostream& out, int delay, int mult) : out(out), delay(delay), delay_multiplier(mult),
default_delay(delay), default_mult(mult) {}

SlowPrinter::SlowPrinter(const SlowPrinter& other) : out(other.out), delay(other.delay),
delay_multiplier(other.delay_multiplier) {}

SlowPrinter& SlowPrinter::operator<<(const char* str) {
    for (size_t i = 0; i < strlen(str); ++i) {
        out << str[i] << std::flush;
        this_thread::sleep_for(chrono::milliseconds(delay));
    }
    this_thread::sleep_for(chrono::milliseconds(delay * delay_multiplier));
    return *this;
}

SlowPrinter& SlowPrinter::operator<<(const std::string& str) {
    for (const auto& c : str) {
        out << c << std::flush;
        std::this_thread::sleep_for(std::chrono::milliseconds(delay));
    }
    this_thread::sleep_for(chrono::milliseconds(delay * delay_multiplier));
    return *this;
}

SlowPrinter& SlowPrinter::operator<<(ostream& (*manipulator)(ostream&)) {
    manipulator(out);
    return *this;
}

int SlowPrinter::getDelay() {
    return this->delay;
}

int SlowPrinter::getMult() {
    return this->delay_multiplier;
}

void SlowPrinter::setDelay(int delay) {
    this->delay = delay;
}

void SlowPrinter::setMult(int mult) {
    this->delay_multiplier = mult;
}

void SlowPrinter::resetDelay() {
    this->delay = default_delay;
}

void SlowPrinter::resetMult() {
    this->delay_multiplier = default_mult;
}
```

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Arsitektur Class	13522158	13522126
Next	13522158	13522147
Cetak Penyimpanan	13522126	13522131
Pungut Pajak	13522147	13522143
Cetak Ladang dan Peternakan	13522126	13522131
Tanam	13522143	13522131
Ternak	13522143	13522131
Bangun Bangunan	13522131	13522147
Makan	13522131	13522126
Memberi Pangan	13522131	13522143
Membeli	13522147	13522126
Menjual	13522147	13522126
Memanen	13522143	13522126
Muat	13522158	13522143
Simpan	13522158	13522147

Tambah Pemain	13522158	13522147