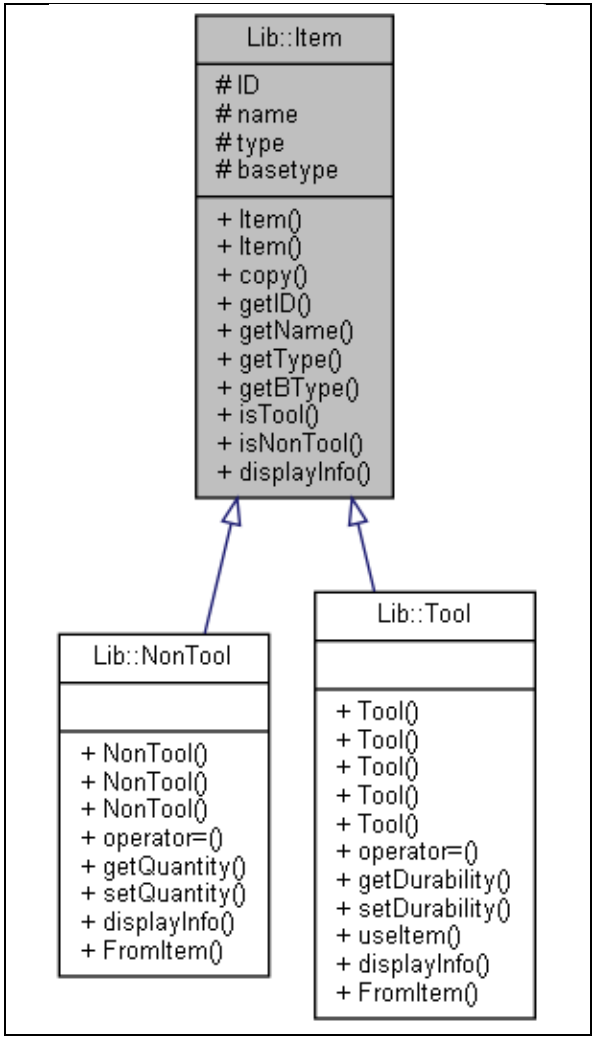
Kelas : 01
Nomor Kelompok : 03
Nama Kelompok : Kosan Benis
1. 13520028 / Timothy Stanley Setiawan
2. 13520091 / Andreas Indra Kurniawan
3. 13520103 / Amar Fadil
4. 13520124 / Owen Christian Wijaya
5. 13520139 / Fachry Dennis Heraldi
6. 13520157 / Thirafi Najwan Kurniatama
Asisten Pembimbing : Morgen Sudyanto

1. Diagram Kelas

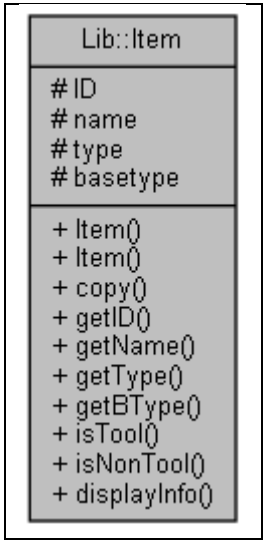
1.1. Modul Item

Alasan dipilihnya desain dibawah adalah karena adanya kesamaan perilaku dan tanggung jawab yang dimiliki oleh Kelas Tool dan Kelas NonTool terhadap kelas induknya yaitu Kelas Item. Pemilihan desain ini juga merujuk kepada definisi item untuk pembagian kategori yang dibagi menjadi dua, yaitu tool dan non-tool. Kelebihan dari pemilihan desain ini adalah kelas NonTool dan Kelas Tool dapat mengakses method yang dimiliki oleh kelas Item namun dapat memberi batasan juga, contohnya ada method yang hanya dimiliki oleh Kelas Tool saja atau Kelas NonTool saja. Kendala yang dialami dalam pemilihan desain OOP ini adalah menentukan method-method apa saja yang akan diwarisi oleh Kelas Tool dan Kelas NonTool dari Kelas Item

1.1.1. Diagram Inheritence Modul Item



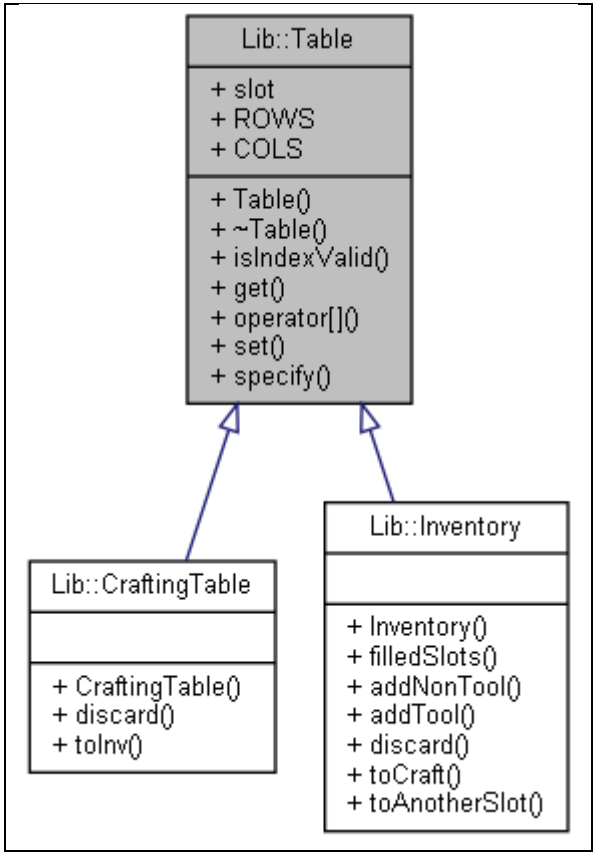
1.1.2. Diagram Collaboration Modul Item



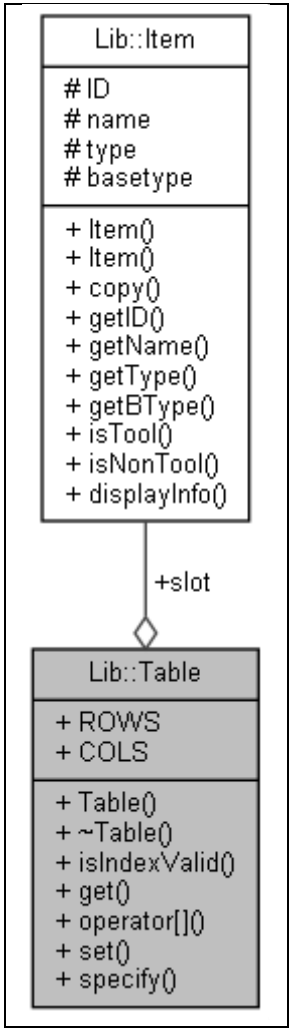
1.2. Modul Table

Alasan dipilihnya desain dibawah adalah karena adanya kesamaan perilaku dan tanggung jawab yang dimiliki oleh Kelas Inventory dan Kelas CraftingTable terhadap kelas induknya yaitu Kelas Table. Selain itu, Kelas Table juga memiliki agregasi dengan Kelas Item karena Kelas Item merupakan bagian dari Kelas Table. Kelebihan dari pemilihan desain ini adalah Kelas Inventory dan Kelas CraftingTable dapat mengakses method yang dimiliki oleh Kelas Table namun dapat memberi batasan juga, contohnya ada method yang hanya dimiliki oleh Kelas Inventory saja atau Kelas CraftingTable saja. Kendala yang dialami dalam pemilihan desain OOP ini adalah menentukan method-method apa saja yang akan diwarisi oleh Kelas Inventory dan Kelas CraftingTable dari Kelas Table.

1.2.1. Diagram Inheritance Modul Table



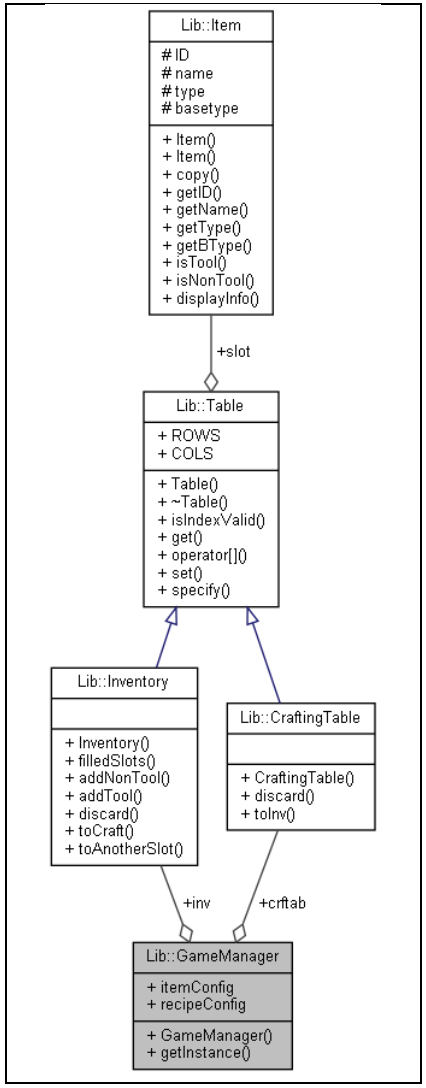
1.2.2. Diagram Collaboration Modul Table



1.3. Modul GameManager

Alasan dipilihnya desain dibawah agar kelas-kelas pada Modul Item dan Modul Table dapat terenkapsulasi dan diinisiasi oleh satu modul saja. Kelebihan dari pemilihan desain ini adalah hanya melalui satu modul saja, modul yang lain dapat terwakili perilaku dan tanggung jawabnya. Kendala yang dialami dalam pemilihan desain OOP ini adalah sedikit sulit dalam melakukan tracing kepada siapa-siapa saja suatu objek diberi tanggung jawabnya

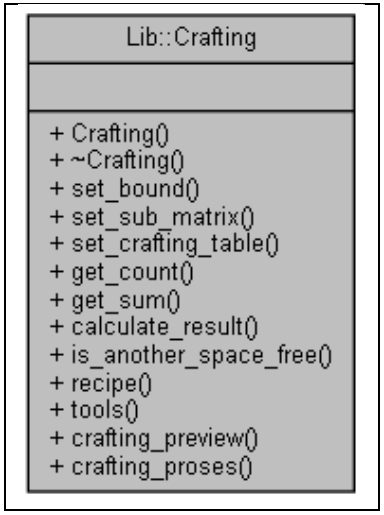
1.3.1. Diagram Collaboration Modul GameManager



1.4. Modul Crafting

Alasan dipilihnya desain dibawah ini karena modul ini digunakan sebagai handler bagi Modul Item dan Modul Table. Kelebihan dari desain ini adalah operasi-operasi dapat terenkapsulasi pada satu kelas dan dapat digunakan oleh kelas lain. Kendala yang dialami dalam pemilihan desain adalah menentukan apa-apa saja method/operasi yang diperlukan oleh Modul Item dan Modul Table.

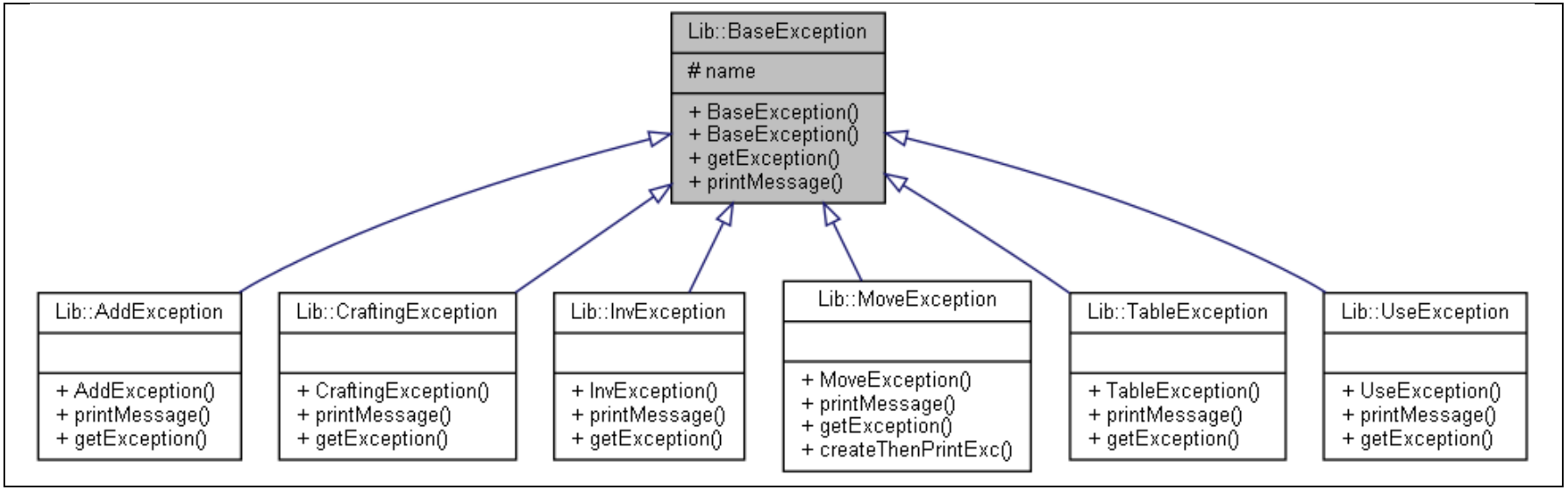
1.4.1. Diagram Collaboration Modul Crafting



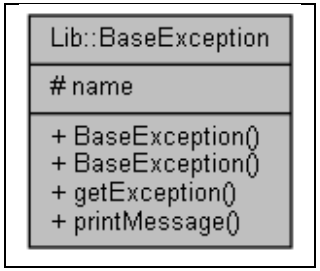
1.5. Modul BaseException

Alasan dipilihnya desain dibawah adalah diperlukannya kelas-kelas yang memiliki perilaku dan tanggung jawab yang sama terhadap suatu kelas induk. Kelebihan dari pemilihan desain ini adalah untuk operasi yang memilki kesamaan hanya perlu diimplementasikan sekali saja pada kelas induknya. Tiap kelas anak juga memiliki batasan dimana kelas anak lain tidak dapat mengakses method yang hanya diimplementasi di dirinya saja. Kendala yang dialami dalam pemilihan desain OOP ini adalah menentukan berapa banyak kelas yang dibutuhkan dan yang akan diwariskan.

1.5.1. Diagram Inheritance BaseException



1.5.2. Diagram Collaboration BaseException



2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

2.1.1. Kelas Table – Kelas Inventory / Crafting Table

Kelas Table adalah *base class* bagi kelas Inventory dan Crafting Table. Kedua kelas tersebut mempunyai kesamaan dalam hal penggunaan buffer, *setter item* di dalam buffer, implementasi *layout output* untuk CLI, dan validasi *boolean.* Oleh karena itu, kelas Table dijadikan basis bagi kedua kelas tersebut. Perbedaan antara Inventory dan Crafting Table terdapat pada penggunaan, ukuran, dan metode-metode yang dimiliki serta implementasi algoritma metode-metode tersebut. Metode pemindahan barang dan penggunaan barang juga diatur oleh kedua kelas tersebut.

Header kelas Table (Lib/core/components/headers/Table.hpp)

```
class Table
    {
    public:
        Item** slot;
        const int ROWS;
        const int COLS;
        Table(int, int);
        ~Table();
        bool isIndexValid(int);
        Item* get(int pos);
        Item* operator[](int);
        void set(int, Item*);
        void specify(int);
    };
```

Header kelas Inventory (Lib/core/components/headers/Table.hpp)

```
class Inventory: public Table
    {
    public:
```

```
        Inventory();
        friend ostream& operator<<(ostream& os, Inventory* inven);
        friend ostream& operator<<(ostream& os, Inventory& inven);
        int filledSlots();
        void addNonTool(NonTool* item, int start);
        void addTool(Tool* item, int quant);
        void discard(int quant, int slot);
        void toCraft(int slotSrc, int destSlot[], int N);
        void toAnotherSlot(int slotSrc, int destSlot[]);
    };
```

Header kelas CraftingTable (Lib/core/components/headers/Table.hpp)

```
class CraftingTable: public Table
    {
    public:
        CraftingTable();
        friend ostream& operator<<(ostream& os, CraftingTable* ct);
        friend ostream& operator<<(ostream& os, CraftingTable& ct);
        void discard(int quant, int slot);
        void toInv(int slotSrc, int destSlot[]);
    };
```

2.1.2. Kelas Item – Kelas Tool / NonTool

Kelas Item adalah *base class* bagi kelas Tool dan NonTool. Pembeda kelas Tool dan NonTool adalah metode penambahan (lebih lanjut di kelas Inventory) dan pencetakan informasi. Oleh karena itu, kelas Item mempunyai beberapa *virtual methods* yang dapat di-*override* oleh kedua kelas tersebut. Selain itu, implementasi *polymorphism* terdapat di penggunaan *typecasting*, sehingga kelas Item dapat diberlakukan sebagai kelas Tool/NonTool dan begitupula sebaliknya.

Header kelas Item (Lib/core/components/headers/Item.hpp)

```
class Item {
```

```
    protected:
        int ID;          // item ID (e.g. 24)
        string name;     // item name (e.g. "DIAMOND_SWORD")
        string type;     // item type (TOOL/NONTOOL)
        string basetype; // basetype (oak log basetypenya log)
    public:
        Item(int, string, string, string); // user-defined constructor
        Item(const Item& i); // copy constructor
        Item* copy(); // proper copy with inheritance
        int getID() const;
        string getName() const;
        string getType() const;
        string getBType() const;
        bool isTool();
        bool isNonTool();
        bool isUndef();
        virtual void displayInfo() const;
    };
```

Header kelas NonTool (Lib/core/components/headers/Item.hpp)

```
class NonTool : public Item {
    private:
        int quantity;
    public:
        NonTool(int, string, string, string, int);
        NonTool(const TupleItem&, int);
        NonTool(const NonTool& nt);
        static NonTool& FromItem(const Item*);
        NonTool& operator=(const NonTool& other);
        int getQuantity() const;
        void setQuantity(int);
```

```
        void displayInfo() const;
    };
```

Header Kelas Tool

```
class Tool : public Item {
    private:
        int durability; // durability of the tool
    public:
        Tool(int, string, int);
        Tool(int, string);
        Tool(const TupleItem&, int);
        Tool(const TupleItem&);
        Tool(const Tool&);
        static Tool& FromItem(const Item*);
        Tool& operator=(const Tool& other);
        int getDurability() const;
        void setDurability(int);
        void useItem();
        void displayInfo() const;
    };
```

2.2. Method/Operator Overloading

2.2.1. Overload operator = untuk Item NonTool/Tool

Overload operator = digunakan sebagai interface untuk *copy constructor* Tool dan NonTool. Overloading ini digunakan untuk mempermudah *assignment* suatu item baru tanpa harus memanggil *copy constructor* tiap kali ingin melakukan perubahan dari suatu variabel item. Dengan demikian, *source code* dari program yang dibuat lebih *readable* dan lebih mudah untuk di-*maintain*.

Implementasi overload = untuk Item NonTool

```
NonTool& NonTool::operator=(const NonTool& other) {
        this→ID = other.ID;
```

```
        this→name = other.name;
        this→type = other.type;
        this→basetype = other.basetype;
        this→quantity = other.quantity;
        return *this;
    }
```

Implementasi overload = untuk Item Tool

```
Tool& Tool::operator=(const Tool& other) {
        this→ID = other.ID;
        this→name = other.name;
        this→type = other.type;
        this→basetype = other.basetype;
        this→durability = other.durability;
        return *this;
    }
```

2.2.2. Ostream Overload untuk Layout

Overload ostream dipakai sehingga *layout* untuk Inventory dan Crafting Table dapat dipanggil menggunakan operator cout tanpa harus memanggil suatu method tertentu untuk melakukan penampilan.

Implementasi ostream overload untuk inventory

```
 ostream& operator<<(ostream& os, Inventory* inven) {
        int undef_count = INV_SIZE;
        os << "\n\n[INVENTORY DETAILS]" << endl;
        os << "Slot" << " | "
            << setw(NUMWIDTH) << "ID" << " | "
            << setw(WIDTH) << "Name" << " | "
            << setw(WIDTH) << "Type" << " | "
```

```
            << setw(WIDTH) << "Base Type" << endl;
        for (int i = 0; i < INV_SIZE; i++) {
            if (inven→slot[i]→getID() != UNDEFINED_ID) {
                undef_count--;
                os << setw(NUMWIDTH - to_string(i).length()) << "I" << i << " | ";
                inven→specify(i);
                os << endl;
            }
        }
        if (undef_count == INV_SIZE){
            os << "\nNo items in inventory :(" << endl;
        }
        return os;
    }
```

Implementasi ostream overload untuk crafting table

```
ostream& operator<<(ostream& os, CraftingTable& ct) {
        os << "\nCrafting Table : " << endl;
        for (int i = 0; i < CRAFT_SIZE; i++) {
            os << "[C" << i << " "
                << (ct.slot[i]→getID()) << " "
                << (ct.slot[i]→isTool() ?
                    ct.slot[i]→getDurability() :
                    ct.slot[i]→getQuantity()) << "] ";
                if ((i + 1) % CRAFT_COLS == 0) {
                    os << endl;
                }
        }
        return os;
    }
```

2.2.3. Overload Operator [ ] untuk Buffer

Overload operator [ ] sebenarnya adalah implementasi dari method get yang digunakan untuk mendapatkan elemen dari buffer. Overload digunakan untuk menyederhanakan sintaks yang dipakai, sehingga kode lebih *readable* dibandingkan menggunakan method get.

Implementasi method get dan overloading operator[]

```
Item* Table::get(int pos) {
        if (!isIndexValid(pos)) {
            throw new InvException("INVALID");
        }
        return this→slot[pos];
 }
 Item* Table::operator[](int pos) {
        return get(pos);
 }
```

2.3. Template & Generic Classes

2.3.1. Tuple

Tuple adalah template yang disediakan oleh standard library. Tuple digunakan dalam melakukan parsing item dan parsing resep dari crafting item. Konsep ini digunakan karena tuple dapat menyimpan koleksi tipe data yang berbeda-beda. Keuntungan dari tuple adalah dari segi penggunaan operasinya. Untuk membuat tuple dapat menggunakan method make_tuple(), sedangkan untuk mengambil elemen spesifik dari tuple dapat menggunakan method get().

Implementasi dari kelas GameManager

```
    tuple <string, string, string, string> GameManager::parseItem(string line) {
        istringstream iss(line);
        string ID;
        string nameTok, typeTok, btypeTok;
```

```
        getline(iss, ID, ' ');
        getline(iss, nameTok, ' ');
        getline(iss, typeTok, ' ');
        getline(iss, btypeTok, ' ');
        return make_tuple(ID, nameTok, typeTok, btypeTok);
    }
    tuple <tuple<int, int>, vector<string>, TupleItem, int> GameManager::parseRecipe(ifstream* file) {
        ...
        for (tuple tup : itemConfig) {
            if (get<1>(tup) == name) {
                ID = get<0>(tup);
                type = get<2>(tup);
                btype = get<3>(tup);
                break;
            }
        }
        return make_tuple(make_tuple(n, m), recipeList, make_tuple(ID, name, type, btype), quantity);
    }
```

2.3.2. Vector

Vector adalah template yang disediakan oleh standard library. Vector digunakan untuk menyimpan list dari resep bersamaan dengan penggunaan tuple yang telah dijelaskan sebelumnya. Vector juga digunakan untuk menyimpan informasi slot tujuan untuk operasi MOVE karena operasi ini dapat dilakukan untuk tujuan multislot. Vector sifatnya dinamis sehingga tidak perlu diinisiasi ukurannya. Keuntungan vector juga terletak pada cara *assignment value*-nya yang cukup mudah menggunakan method push_back().

Penggunaan vector pada GameManager.cpp

```
        ...
        vector<string> recipeList;
        ...
        for (int j = 0; j < m; j++) {
            string temp;
            getline(iss, temp, ' ');
```

```
            if (temp == "-") {
               recipeList.push_back("UNDEFINED");
            }
            else {
               recipeList.push_back(temp);
            }
         }
         ...
```

Penggunaan vector pada handler.cpp

```
        ...
        string slotSrc;
        int slotQty;

        cin >> slotSrc >> slotQty;
        vector<string> slotDestV;
        for (int i = 0; i < slotQty; i++) {
            string slotDest;
            cin >> slotDest;
            slotDestV.push_back(slotDest);
        }
        ...
```

2.4. Exception

2.4.1. Kelas BaseException

Kelas BaseException digunakan sebagai abstract class untuk jenis-jenis exception yang ada pada program. Kelas ini hanya digunakan untuk menyimpan nama dari exception karena fungsi getException dan printMessage merupakan fungsi pure virtual.

Header dari kelas BaseException

```
class BaseException {
    protected:
```

```
        string name;
    public:
        BaseException();
        BaseException(string);
        virtual string getException() = 0;
        virtual void printMessage() = 0;
    };
```

2.4.2. Kelas AddException

Kelas AddException digunakan untuk menampilkan error yang terdapat pada saat melakukan penambahan barang. Nama barang akan dicatat pada name di parent class BaseException sedangkan jenis error akan dicatat pada btype kelas addException. Hal ini akan memudahkan ketika melakukan print error dan menjaga kode tetap DRY.

Implementasi method dari kelas AddException

```
    string AddException::getException(){
        if (btype == "INVALID") {
            return "[ADD-EXC] Item " + name + " doesn't exist!\n";
        }
        else if (btype == "OVERCAP") {
            return "[ADD-EXC] Quantity exceeded current capacity!";
        }
    }

    AddException::AddException(string name, string btype) : BaseException(name) {
        this->btype = btype;
    }

    void AddException::printMessage() {
        cout << getException();
    }
```

2.4.3. Kelas CraftingException

Kelas CraftingException juga digunakan khusus untuk menyimpan tipe-tipe error ketika melakukan craft sehingga error yang terdapat ketika melakukan crafting lebih mudah untuk dicari.

Implementasi method kelas CraftingException

```
CraftingException::CraftingException(string btype) {
    this->btype = btype;
};

void CraftingException::printMessage() {
    cout << getException();
}
string CraftingException::getException(){
    string exc = "[CRF-EXC]";
    if (btype == "TOOL") {
        exc = "[CRF-EXC] Can't craft more than two tools!\n";
        return  exc;
    } else {
        exc = "[CRF-EXC] Recipe not found!\n";
        return exc;
    }
}
```

2.4.4. Kelas UseException

Kelas UseException digunakan untuk menyimpan error yang terdapat ketika melakukan use pada item sehingga jenis error ketika use dapat dicari dengan mudah.

Implementasi method kelas UseException

```
UseException::UseException(string name) : BaseException(name) {}

void UseException::printMessage() {
```

```
        cout << getException();
    }

    string UseException::getException(){
        string exc = "[USE-EXC] Couldn't use a Non-Tool item! (" + name + ")\nUse command 'DISCARD' to use nontool
items.\n";
        return exc;
    }
```

2.4.5. Kelas MoveException

Kelas MoveException digunakan untuk menyimpan error yang terdapat ketika melakukan pemindahan barang sehingga jenis error yang terjadi ketika memindahkan barang dapat dicari dengan mudah.

Implementasi constructor dan printMessage kelas MoveException

```
    MoveException::MoveException(string name) : BaseException(name) {}

    void MoveException::printMessage(){
        cout << getException();
    }

    string MoveException::getException(){
        string exc = "[MOV-EXC] ";
        if(this->name == "VOID"){
            return exc + "You are trying to move the void...\n";
        }else if(this->name == "INVALID"){
            return exc + "You move an invalid slot!\n";
        }else if(this->name == "INVALIDDEST"){
            return exc + "You entered an invalid destination slot!\n";
        }else if(this->name == "DOUBLETYPEDEST"){
            return exc + "You can only move the item to one type of slot!\n";
        }else if(this->name == "MOVETO2INV"){
            return exc + "You can only move this item to 1 inventory slot\n";
        }else if(this->name == "CRAFTTOCRAFT"){
```

```
            return exc + "You can't move item from crafting slot to another crafting slot.\n";
        }else if(this->name == "DIFFTYPE"){
            return exc + "The item you are trying to move to is not of the same type!\n";
        }else if(this->name == "FULL"){
            return exc + "The slot is full, you can't move it there!\n";
        }else if(this->name == "TOOL"){
            return exc + "There's a tool already here!\n";
        }else if(this->name == "NOTENOUGH"){
            return exc + "You don't have enough item to move\n";
        }else if(this->name == "INVALIDSLOT"){
            return exc + "Invalid slot count\n";
        }
        return exc;
    }
```

2.4.6. Kelas InvException

Kelas InvException digunakan untuk menyimpan error yang terdapat ketika melakukan operasi discard ataupun add item ke dalam Inventory.

Implementasi constructor dan printMessage kelas InvException

```
    InvException::InvException(string name) : BaseException(name) {}

    void InvException::printMessage() {
        cout << getException();
    }

    string InvException::getException(){
        string exc = "[INV-EXC] ";
        if (this->name == "INVALID") {
            return exc + "Invalid index! Index ranges from 0-26\n";
        }
        else if (this->name == "FULL") {
            return exc + "Inventory full!\n";
```

```
        }
        else if (this->name == "EMPTY") {
            return exc + "This slot is empty; couldn't discard anything!\n";
        }
        else if (this->name == "OVER") {
            return exc + "You're attempting to discard more than the available quantity!\n";
        }
    }
```

2.4.7. Kelas TableException

Kelas TableException digunakan untuk menyimpan error yang terjadi ketika melakukan operasi pada crafting table.

Implementasi constructor dan printMessage kelas InvException

```
    TableException::TableException(string name) : BaseException(name) {}

    void TableException::printMessage() {
        cout << getException();
    }

    string TableException::getException(){
        string exc = "[TAB-EXC] ";
        if (this->name == "INVALID") {
            return exc + "Invalid index! Index ranges from 0-9\n";
        }
        else if (this->name == "OCCUPIED") {
            return exc + "This slot is already occupied!\n";
        }
        else if (this->name == "FULL") {
            return exc + "This slot has reached its maximum capacity!\n";
        }
        else if (this->name == "EMPTY") {
            return exc + "This slot is empty; couldn't discard anything!\n";
        }
```

```
        return "";
    }
```

2.5. C++ Standard Template Library

2.5.1. Tuple

Tuple digunakan dalam melakukan parsing item dan parsing resep dari crafting item. Konsep ini digunakan karena tuple dapat menyimpan koleksi tipe data yang berbeda-beda. Keuntungan dari tuple adalah dari segi penggunaan operasinuya. Untuk membuat tuple dapat menggunakan method make_tuple(), sedangkan untuk mengambil elemen spesifik dari tuple dapat menggunakan method get().

Implementasi dari kelas GameManager

```
    tuple <string, string, string, string> GameManager::parseItem(string line) {
        istringstream iss(line);
        string ID;
        string nameTok, typeTok, btypeTok;
        getline(iss, ID, ' ');
        getline(iss, nameTok, ' ');
        getline(iss, typeTok, ' ');
        getline(iss, btypeTok, ' ');
        return make_tuple(ID, nameTok, typeTok, btypeTok);
    }
    tuple <tuple<int, int>, vector<string>, TupleItem, int> GameManager::parseRecipe(ifstream* file) {
        ...
        for (tuple tup : itemConfig) {
            if (get<1>(tup) == name) {
                ID = get<0>(tup);
                type = get<2>(tup);
                btype = get<3>(tup);
                break;
            }
```

```
        }
        return make_tuple(make_tuple(n, m), recipeList, make_tuple(ID, name, type, btype), quantity);
    }
```

2.5.2. Vector

Vector digunakan untuk menyimpan list dari resep bersamaan dengan penggunaan tuple yang telah dijelaskan sebelumnya. Vector juga digunakan untuk menyimpan informasi slot tujuan untuk operasi MOVE karena operasi ini dapat dilakukan untuk tujuan multislot. Vector sifatnya dinamis sehingga tidak perlu diinisiasi ukurannya. Keuntungan vector juga terletak pada cara *assignment value*-nya yang cukup mudah menggunakan method push_back().

Penggunaan vector pada GameManager.cpp

```
        ...
        vector<string> recipeList;
        ...
        for (int j = 0; j < m; j++) {
            string temp;
            getline(iss, temp, ' ');
            if (temp == "-") {
               recipeList.push_back("UNDEFINED");
            }
            else {
               recipeList.push_back(temp);
            }
         }
         ...
```

Penggunaan vector pada handler.cpp

```
        ...
        string slotSrc;
        int slotQty;

        cin >> slotSrc >> slotQty;
        vector<string> slotDestV;
        for (int i = 0; i < slotQty; i++) {
```

```
            string slotDest;
            cin >> slotDest;
            slotDestV.push_back(slotDest);
        }
        ...
```

2.6. Konsep OOP lain

2.6.1. Abstract Class dan Virtual Function

Implementasi *abstract class* terdapat pada kelas *Item* dan kelas *Tool/Non-Tool*. Kelas *Item* mempunyai metode-metode virtual yang dapat di-override sehingga dapat dijadikan *abstract class* bagi kelas turunan *Tool* dan *Non-Tool*. Hal serupa juga dapat dilihat pada implementasi *Table* dan kelas *Inventory/Crafting Table*, dimana *Inventory* dan *Crafting Table* melakukan override terhadap metode-metode yang ada pada kelas *Table*. Selain itu, implementasi *exception* juga memanfaatkan konsep *abstract class*, di mana kelas BaseException menjadi *base class* bagi kelas-kelas *exception* turunannya.

2.6.2. Aggregation

Aggregation digunakan antara kelas Item dengan kelas Table dengan mendefinisikan objek bertipe data objek Item pada kelas Table sehingga kelas Item dengan kelas Table memiliki relasi HAS-A. Penggunaan aggregation ini dikarenakan pada kelas Table memiliki buffer untuk merepresentasikan slot yang perilaku dan tanggung jawabnya didefinisikan oleh kelas Item. Dengan demikian, ketika kelas Table didefinisikan dan digunakan, secara otomatis kelas Item juga terdefinisi dan dapat digunakan pula untuk kelas Table.

Header kelas Item (Lib/core/components/headers/Item.hpp)

```
class Item {
    protected:
        int ID;          // item ID (e.g. 24)
        string name;     // item name (e.g. "DIAMOND_SWORD")
        string type;     // item type (TOOL/NONTOOL)
```

```
        string basetype; // basetype (oak log basetypenya log)
    public:
        Item(int, string, string, string); // user-defined constructor
        Item(const Item& i); // copy constructor
        Item* copy(); // proper copy with inheritance
        int getID() const;
        string getName() const;
        string getType() const;
        string getBType() const;
        bool isTool();
        bool isNonTool();
        bool isUndef();
        virtual void displayInfo() const;
    };
```

Header kelas Table (Lib/core/components/headers/Table.hpp)

```
class Table
    {
    public:
        Item** slot;
        const int ROWS;
        const int COLS;
        Table(int, int);
        ~Table();
        bool isIndexValid(int);
        Item* get(int pos);
        Item* operator[](int);
        void set(int, Item*);
        void specify(int);
    };
```

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Pembuatan GUI

Implementasi GUI dibuat dalam folder GUI. GUI dibuat menggunakan C++/CLI yang memanfaatkan sintaks C++ yang disesuaikan sehingga dapat digunakan untuk membangun *interface* dengan komponen-komponen serupa C#. GUI dapat dioperasikan serupa *interface* Minecraft, dengan implementasi penambahan/pengurangan item berupa *form* terpisah. Pengguna dapat melakukan *drag-and-drop* item dan melakukan *crafting*.

Implementasi file Main dari library GUI

```
namespace GUI {
    Main::Main() {...}
    Main::~Main() {...}
    void Main::itemAdd_Click(Object^ sender, EventArgs^ e) {...}
    void Main::itemMenu_Opening(Object^ sender, CancelEventArgs^ e) {...}
    void Main::itemDiscard_Click(Object^ sender, EventArgs^ e) {...}
    void Main::itemUse_Click(Object^ sender, EventArgs^ e) {...}
    void Main::systemExport_Click(Object^ sender, EventArgs^ e) {...}
}
```

GUI dari Minecraft Crafting Table

3.1.2. Multiple Crafting

Multiple crafting memungkinkan pengguna untuk melakukan *crafting* menggunakan lebih dari satu material. Proses *multiple crafting* akan menghasilkan *item* dengan kelipatan sesuai material yang terdapat pada tool (mis. melakukan crafting *stick* dengan masing-masing slot terisi dua *plank* akan menghasilkan 8 buah stick). Hasil *crafting* untuk non-tool akan di-*stack* dan disimpan dalam satu slot, sementara hasil *multiple crafting* untuk tool tidak akan distack dan akan mengisi slot inventory yang berbeda-beda. Apabila ada jumlah item berlebih di *crafting table* setelah crafting, maka item tersebut masih tetap tersedia di *crafting table* (mis. melakukan crafting *stick* dengan slot pertama terisi tiga *plank* dan dua *plank* menghasilkan 8 buah *stick* dengan satu *plank* tersisa di *crafting table*).

Implementasi method crafting_proses()

```
void Crafting::crafting_proses() {
    Crafting crf;
    Item* item = crf.crafting_preview();
    int count = crf.get_count();
```

```
    if (count == -1) {
        throw new CraftingException("TOOL1");
    }
    else if (count == -2) {
        throw new CraftingException("TOOL2");
    }
    else if (item == nullptr) {
        throw new CraftingException("RECIPE");
    }
    int sum = crf.get_sum();
    if (item->isNonTool()) {
        cout << "Item " << item->getName() << " successfully crafted! (Quantity: "
            << sum << ") " << endl;
        gm.inv.addNonTool((NonTool*)item, 0);
    } else {
        cout << "Tool " << item->getName() << " successfully crafted! (Durability: "
            << Tool::FromItem(item).getDurability() << ") " << endl;
        gm.inv.addTool((Tool*)item, sum);
    }
    crf.set_crafting_table(count);
}
```

3.1.3. Item dan Tool Baru

Item baru yang diimplementasikan adalah *tools* dan material Gold. Material *gold* berupa *gold nugget* dan *gold ingot*. *Gold ingot* dapat di-craft dari sembilan buah *gold nugget*, dan begitupula sebaliknya. *Tools* yang dapat di-craft antara lain *pickaxe*, *axe*, *hoe*, dan *sword*. Untuk implementasi GUI, pengguna dapat menambahkan dan men-craft item tersebut dengan resep serupa dengan *tools* lainnya. Resep dan data item telah ditambahkan dalam folder config. Untuk GUI sendiri, terdapat *icons* yang telah ditambahkan terkait item-item tersebut.

Konfigurasi file item.txt

```
1 OAK_LOG LOG NONTOOL
2 SPRUCE_LOG LOG NONTOOL
3 BIRCH_LOG LOG NONTOOL
```

```
4 OAK_PLANK PLANK NONTOOL
5 SPRUCE_PLANK PLANK NONTOOL
6 BIRCH_PLANK PLANK NONTOOL
7 STICK - NONTOOL
8 COBBLESTONE STONE NONTOOL
9 BLACKSTONE STONE NONTOOL
10 IRON_INGOT - NONTOOL
11 IRON_NUGGET - NONTOOL
12 GOLD_INGOT - NONTOOL
13 GOLD_NUGGET - NONTOOL
14 DIAMOND - NONTOOL
15 WOODEN_PICKAXE - TOOL
16 STONE_PICKAXE - TOOL
17 IRON_PICKAXE - TOOL
18 GOLDEN_PICKAXE - TOOL
19 DIAMOND_PICKAXE - TOOL
20 WOODEN_AXE - TOOL
21 STONE_AXE - TOOL
22 IRON_AXE - TOOL
23 GOLDEN_AXE - TOOL
24 DIAMOND_AXE - TOOL
25 WOODEN_SWORD - TOOL
26 STONE_SWORD - TOOL
27 IRON_SWORD - TOOL
28 GOLDEN_SWORD - TOOL
29 DIAMOND_SWORD - TOOL
30 WOODEN_HOE - TOOL
31 STONE_HOE - TOOL
32 IRON_HOE - TOOL
33 GOLDEN_HOE - TOOL
34 DIAMOND_HOE - TOOL
```

3.1.4. Unit Testing Implementation

Unit Testing Implementation menggunakan gtest untuk menguji kebenaran program.

Implementasi beberapa unit testing

```
TEST(CRAFTING, EMPTY_SLOT) {
    EXPECT_NO_THROW(Lib::CraftingHandler());
}

TEST(CRAFTING, BIRCH_PLANK) {
    Lib::GiveHandler("BIRCH_LOG", 1);
    std::vector<std::string> sd = { "C0" };
    Lib::MoveHandler("I0", 1, sd);
    EXPECT_NO_THROW(Lib::CraftingHandler());
    std::string slot = "I0";
    Lib::DiscardHandler(stoi(slot.substr(1, slot.length() - 1)), 4);
}

TEST(CRAFTING, DIAMOND_AXE) {
    Lib::GiveHandler("DIAMOND", 3);
    Lib::GiveHandler("STICK", 2);
    std::vector<std::string> sd = { "C0", "C1", "C3"};
    Lib::MoveHandler("I0", 3, sd);
    sd = { "C4", "C7" };
    Lib::MoveHandler("I1", 2, sd);
    EXPECT_NO_THROW(Lib::CraftingHandler());
    std::string slot = "I0";
    Lib::DiscardHandler(stoi(slot.substr(1, slot.length() - 1)), 1);
}
...
```

3.2. Bonus Kreasi Mandiri

3.2.1. Kompresi PNG dan Penggunaan Font untuk GUI

Untuk memasukkan *icon* per item dalam GUI, dibutuhkan proses kompresi dari masing-masing PNG *icon* yang digunakan. Selain itu, font yang digunakan dalam GUI adalah font bertipe *Minecraft*.

4. Pembagian Tugas

Modul (dalam poin spek)	**Implementer**	**Tester**
Mekanisme: Item	13520124	13520103, 13520028, 13520157
Mekanisme: Inventory	13520124, 13520139	13520157, 13520103
Mekanisme: Crafting	13520028, 13520139	13520103
Operasi dan Command: SHOW	13520124, 13520028	13520103, 13520139
Operasi dan Command: GIVE	13520124	13520091, 13520139, 13520157
Operasi dan Command: DISCARD	13520124	13520139, 13520157
Operasi dan Command: MOVE	13520091	13520091, 13520157
Operasi dan Command: USE	13520157	13520157, 13520124, 13520091

Operasi dan Command: CRAFT	13520028	13520124, 13520103, 13520091
Operasi dan Command: EXPORT	13520139	13520124
Konfigurasi Item	13520103	13520157
Konfigurasi Resep	13520103	13520157
Bonus: Pembuatan GUI	13520103	13520124
Bonus: Multiple Crafting	13520028	13520124, 13520103
Bonus: Item dan Tool Baru	13520157	13520124
Bonus: Unit Testing	13520157	13520103, 13520124