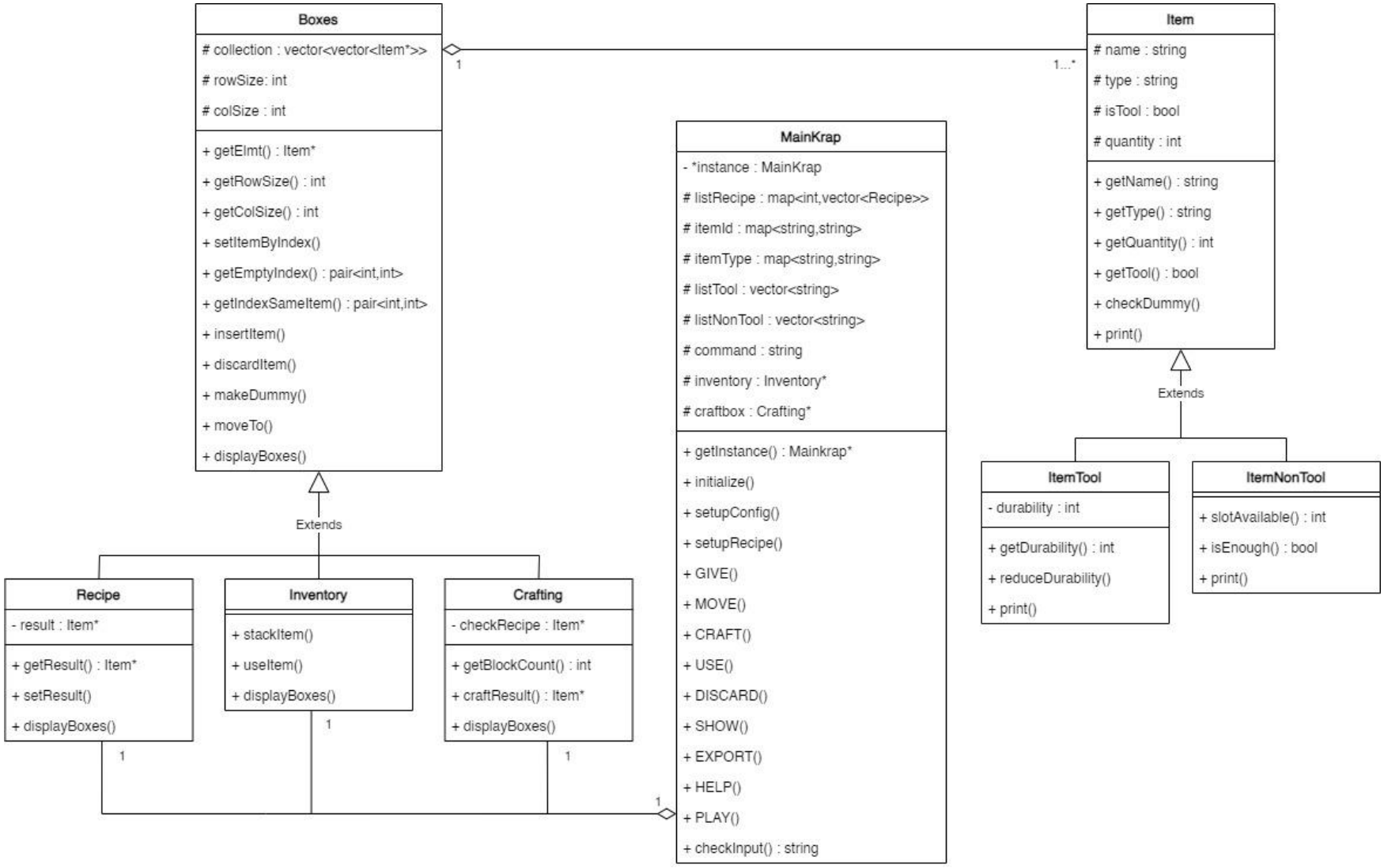
Kelas : 03
Nomor Kelompok : 07
Nama Kelompok : DummyName

1. 13520051 / Flavia Beatrix Leoni A. S.
2. 13520099 / Vincent Prasetiya Atmadja
3. 13520108 / Muhammad Rakha Athaya
4. 13520138 / Gerald Abraham Sianturi
5. 13520159 / Atabik Muhammad Azfa Shofi

Asisten Pembimbing : Matthew Kevin Amadeus

1. Diagram Kelas



Alasan desain kelas-kelas seperti gambar di atas adalah terdapat kesamaan atribut dan metode yang dibutuhkan oleh masing-masing kebutuhan kelas pada spesifikasi tugas, misalnya Recipe, Inventory, dan Crafting, ketiganya merupakan kumpulan item yang hanya dibedakan berdasarkan rowSize dan colSize.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

2.1.1. Kelas Item - Kelas ItemTool, Kelas ItemNonTool

Kelas ItemTool dan ItemNonTool diturunkan dari kelas Item karena Item dapat dibedakan menjadi dua kategori, yaitu Tool dan NonTool yang memiliki perbedaan karakteristik. Namun, ItemTool memiliki atribut yang sama seperti atribut ItemNonTool dengan tambahan atribut durability.

```
class Item{
protected:
    string name;
    string type;
    bool isTool;
    int quantity;
…
};
```

```
class ItemTool: public Item{
private:
    int durability;
…
};
```

```
class ItemNonTool: public Item{
...
};
```

2.1.2. Kelas Boxes - Kelas Recipe, Kelas Inventory, Kelas Crafting

Kelas Recipe, Inventory, dan Crafting diturunkan dari kelas Boxes karena ketiga kelas tersebut sama-sama terbentuk dari suatu matrix of Item dengan ukuran matrix yang berbeda. Kelas Boxes digunakan untuk menyimpan matrix of item dan ukuran baris serta kolomnya.

```
class Boxes {
    protected:
        //Array for saving item
        vector<vector<Item*>>collection;
        const int rowSize;
        const int colSize;
…
};
```

```
class Recipe : public Boxes {
    private:
        Item* result;
…
};
```

```
class Inventory : public Boxes {
…
```

```
};
```

```
class Crafting : public Boxes {
…
};
```

2.2. Method/Operator Overloading

2.2.1. Kelas Item

Pada kelas Item, terdapat operator overloading untuk operator '==', '&', dan '<<'. Operator '==' digunakan untuk mengecek apakah tipe dari kedua Item sama. Operator '&' digunakan untuk mengecek kedua Item merupakan item yang sama, yaitu memiliki nama dan tipe yang sama. Sedangkan operator '<<' digunakan untuk menuliskan item ke layar dengan format tertentu.

```
    // * Operator overloading

    // * Mengecek apakah tipe nya sama
    friend bool operator==(const Item& item1, const Item& item2);
    // * Mengecek apakah benda yang sama (tipe dan nama nya sama)
    friend bool operator&(const Item& item1, const Item& item2);

    friend ostream& operator<<(ostream& out, const Item& item);
```

2.2.2. Kelas ItemNonTool

Pada kelas ItemNonTool, terdapat operator overloading untuk operator '+=' dan '-='. Operator '+=' di-overload untuk menumpuk dua ItemNonTool menjadi satu dan untuk menambah quantity dari suatu ItemNonTool. Operator '-=' digunakan untuk mengurangi quantity dari suatu ItemNonTool.

```
// * Operator overloading
// * Menambahkan item
ItemNonTool& operator+=(const ItemNonTool& item);
ItemNonTool& operator+=(const int& quantity);
ItemNonTool& operator-=(const int& quantity);
```

2.2.3. Kelas Boxes

Pada kelas Boxes, terdapat operator overloading untuk operator '()' yang digunakan untuk mengakses boxes pada indeks baris dan kolom tertentu.

```
        // * Operator overloading
        // * Accessing element
        Item* operator()(int indexRow, int indexCol);
```

2.3. Template & Generic Classes

Konsep ini tidak digunakan karena seluruh kelas yang dibuat hanya perlu mengimplementasikan satu jenis template saja, sehingga penggunaan generic class dianggap tidak perlu.

2.4. Exception

2.4.1. Kelas BaseException

Kelas BaseException merupakan Abstract Base Class untuk menangani kesalahan atau error yang terjadi.

```
class BaseException{
public:
    virtual void printMessage() =0;
};
```

2.4.2. Kelas InvalidTypeException

Kelas InvalidTypeException merupakan turunan dari kelas BaseException yang digunakan untuk menangani kasus menggunakan Item dengan tipe yang salah.

```
class InvalidTypeException: public BaseException{
private:
    bool isTool;
public:
    InvalidTypeException(bool isTool);
    void printMessage();
};
```

2.4.3. Kelas NotSameException

Kelas NotSameException merupakan turunan dari kelas BaseException yang digunakan untuk menangani kasus melakukan stack Item dengan nama dan tipe yang berbeda.

```
class NotSameException: public BaseException{
public:
    NotSameException();
    void printMessage();
};
```

2.4.4. Kelas FullCollectionException

Kelas FullCollectionException merupakan turunan dari kelas BaseException yang digunakan untuk menangani kasus collection penuh ketika menambahkan Item.

```
class FullCollectionException: public BaseException{
public:
    FullCollectionException();
    void printMessage();
};
```

2.4.5. Kelas ArithmeticException

Kelas ArithmeticException merupakan turunan dari kelas BaseException yang digunakan untuk menangani kasus angka yang tidak valid ketika menambahkan atau mengurangi suatu atribut.

```
class ArithmeticException: public BaseException{
private:
    //Given -> Angka yang diberikan
```

```
    //Need -> Angka maksimal (need > 0) atau minimal (need <0) yang tersedia
    int given;
    int need;
public:
    ArithmeticException(int given, int need);
    void printMessage();
};
```

2.4.6. Kelas OperationFailedException

Kelas OperationFailedException merupakan turunan dari kelas BaseException yang digunakan untuk menangani kesalahan atau error yang belum ditangani oleh kelas lain.

```
class OperationFailedException: public BaseException{
private:
    BaseException* exc;
public:
    OperationFailedException(BaseException* exc);

    void printMessage();
};
```

2.5. C++ Standard Template Library

2.5.1. Map

Kelas MainKrap mengimplementasikan STL map untuk mempermudah dalam pencarian elemen berdasarkan kata kunci tertentu. Map digunakan untuk menyimpan banyak block atau slot yang digunakan dalam suatu resep dengan suatu vector of Recipe serta menyimpan nama Item dengan Id dan typenya.

```
class MainKrap{
private:
    static MainKrap *instance;
protected:
    map<int,vector<Recipe>> listRecipe;
    map<string,string> itemId;
    map<string,string> itemType;
    vector<string> listTool;
    vector<string> listNonTool;
    string command;
…
};
```

2.5.2. Vector

Kelas Boxes dan turunannya mengimplementasikan STL vector. STL ini digunakan untuk menyimpan Item karena memiliki container yang telah menangani penyimpanan secara otomatis. Item dapat dimasukkan dan diakses dengan mudah menggunakan metode-metode yang tersedia. Vector digunakan untuk menyimpan Item dalam bentuk vector of vector atau matrix.

```
class Boxes {
```

```
    protected:
        //Array for saving item
        vector<vector<Item*>>collection;
…
};
```

Kelas MainKrap juga mengimplementasikan STL vector untuk menyimpan nama Item yang termasuk dalam ItemTool dan nama Item yang termasuk dalam ItemNonTool.

```
class MainKrap{
private:
    static MainKrap *instance;
protected:
    map<int,vector<Recipe>> listRecipe;
    map<string,string> itemId;
    map<string,string> itemType;
    vector<string> listTool;
    vector<string> listNonTool;
    string command;
…
};
```

2.5.3. Pair

Kelas Boxes dan turunannya mengimplementasikan STL pair untuk memudahkan dalam menyimpan dua objek menjadi satu yaitu indeks baris dan kolom dari suatu vector of vector. Pair digunakan sebagai parameter dan return value dari method yang ada untuk mempermudah penggunaan indeks vector of vector dalam proses selanjutnya.

```
class Boxes {
…
        // * Method
        // * Get empty cell with lowest index,
        // Lowest index dicek berdasarkan column nya dulu
        pair<int, int> getEmptyIndex();

        // * get index of item in the box if item in the box same as the parameter
        pair<int, int> getIndexSameItem(Item* item);
…
        void moveTo(Boxes& target, pair<int, int>indexSrc, vector<pair<int,int>> indexDst);

…
};
```

2.6. Konsep OOP lain

2.6.1. Abstract Base Class

Kelas Boxes merupakan abstract base class karena memiliki sebuah pure virtual method function, yaitu fungsi displayBoxes.

```
class Boxes {
    protected:
        …
        virtual void displayBoxes()=0; // display all Item di Boxes
};
```

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Item dan Tool Baru

Kami membuat beberapa item dan tool baru, yaitu GOLD_INGOT, GOLD_NUGGET, GOLDEN_AXE, GOLDEN_PICKAXE, dan GOLDEN_SWORD. Berikut merupakan salah satu resep dari item baru yang kami buat, yaitu GOLDEN_PICKAXE.

```
3 3
GOLD_INGOT GOLD_INGOT GOLD_INGOT
- STICK -
- STICK -
GOLDEN_PICKAXE 1
```

3.1.2. Unit Testing Implementation

Berikut merupakan beberapa testing yang kami implementasikan. Implementasi testing yang lebih lengkap dapat dilihat pada file testItem.cpp dan testBox.cpp.

```cpp
#include "./Item.hpp"
#include <iostream>
using namespace std;

int main(){
    freopen("testresult.txt", "w", stdout);

    // Test Constructor
    // Dummy Item
    Item* dummyItem = new Item;
    ItemNonTool* dummyItemNT = new ItemNonTool;
    ItemTool* dummyItemT = new ItemTool;

    // Non Dummy Item
    ItemNonTool* itemNT = new ItemNonTool("birch_log", "log", 3);
    ItemNonTool* itemNT1 = new ItemNonTool("birch_log", "log", 4);
    ItemNonTool* itemNT2 = new ItemNonTool("wood_log", "log", 2);
    ItemNonTool* itemNT3 = new ItemNonTool("wood_plank", "plank", 2);

    ItemTool* itemT = new ItemTool("wood_axe", "axe");
    ItemTool* itemT1 = new ItemTool("iron_axe", "axe");
    ItemTool* itemT2 = new ItemTool("wood_axe", "axe");
    ItemTool* itemT3 = new ItemTool("wood_rope", "rope");

    cout << "Try operator==" << endl;
```

```
    cout << "Hasil yang diharapkan: YES NO NO YES YES NO YES YES NO" << endl;
    cout << ((*dummyItem) == (*dummyItemNT) ? "YES" : "NO") << endl;
    cout << ((*dummyItem) == (*itemNT) ? "YES" : "NO") << endl;
    cout << ((*itemNT) == (*itemT) ? "YES" : "NO") << endl;
    cout << ((*itemNT) == (*itemNT1) ? "YES" : "NO") << endl;
    cout << ((*itemNT2) == (*itemNT1) ? "YES" : "NO") << endl;
    cout << ((*itemNT1) == (*itemNT3) ? "YES" : "NO") << endl;
    cout << ((*itemT) == (*itemT1) ? "YES" : "NO") << endl;
    cout << ((*itemT2) == (*itemT1) ? "YES" : "NO") << endl;
    cout << ((*itemT1) == (*itemT3) ? "YES" : "NO") << endl;

    cout << endl;

    cout << "Try operator&&" << endl;
    cout << "Hasil yang diharapkan: YES NO NO YES NO NO NO YES NO" << endl;
    cout << ((*dummyItem) & (*dummyItemNT) ? "YES" : "NO") << endl;
    cout << ((*dummyItem) & (*itemNT) ? "YES" : "NO") << endl;
    cout << ((*itemNT) & (*itemT) ? "YES" : "NO") << endl;
    cout << ((*itemNT) & (*itemNT1) ? "YES" : "NO") << endl;
    cout << ((*itemNT2) & (*itemNT1) ? "YES" : "NO") << endl;
    cout << ((*itemNT1) & (*itemNT3) ? "YES" : "NO") << endl;
    cout << ((*itemT) & (*itemT1) ? "YES" : "NO") << endl;
    cout << ((*itemT2) & (*itemT) ? "YES" : "NO") << endl;
    cout << ((*itemT1) & (*itemT3) ? "YES" : "NO") << endl;
```

```
    cout << endl;
    cout << "Try Getter" << endl << endl;

    cout << "Try getName" << endl;
    cout << "Hasil yang diharapkan: - birch_log iron_axe" << endl;
    cout << dummyItemNT->getName() << endl;
    cout << itemNT1->getName() << endl;
    cout << itemT1->getName() << endl;
    …
```

```
#include "./Boxes.hpp"
#include <iostream>

int main(){
    freopen("testresult.txt", "w", stdout);

    Inventory* inventory = new Inventory;

    cout << "Try Getter" << endl;

    cout << endl;

    cout << "Try getRowSize" << endl;
    cout << "Hasil yang diharapkan 3" << endl;
    cout << inventory->getRowSize() << endl;
```

```
cout << endl;

cout << "Try getColSize" << endl;
cout << "Hasil yang diharapkan 9" << endl;
cout << inventory->getColSize() << endl;

cout << endl;

cout << "Try getEmpyIndex" << endl;
pair<int,int> eIndex = inventory->getEmptyIndex();
cout << "Hasil yang diharapkan 0 0" << endl;
cout << eIndex.first << " " << eIndex.second << endl;


//Create Item for testing
ItemNonTool* itemNT = new ItemNonTool("birch_log", "log", 3);
ItemNonTool* itemNT1 = new ItemNonTool("birch_log", "log", 4);
ItemNonTool* itemNT2 = new ItemNonTool("wood_log", "log", 2);
ItemNonTool* itemNT3 = new ItemNonTool("wood_plank", "plank", 2);

ItemTool* itemT = new ItemTool("wood_axe", "axe");
ItemTool* itemT1 = new ItemTool("iron_axe", "axe");
ItemTool* itemT2 = new ItemTool("wood_axe", "axe");
ItemTool* itemT3 = new ItemTool("wood_rope", "rope");
```

```
cout << endl;

cout << "Try InsertItem" << endl;
cout << "Hasil yang diharapkan (berupa empty index) 0 7 " << endl;

inventory->insertItem(itemNT); // 0 0
inventory->insertItem(itemNT1); // 0 0
inventory->insertItem(itemNT2); // 0 1
inventory->insertItem(itemNT3); // 0 2

inventory->insertItem(itemT); // 0 3
inventory->insertItem(itemT1); // 0 4
inventory->insertItem(itemT2); // 0 5
inventory->insertItem(itemT3); // 0 6

eIndex = inventory->getEmptyIndex();
cout << eIndex.first << " " << eIndex.second << endl;

cout << endl;

cout << "Try Operator()" << endl;
cout << "Hasil yang diharapkan birch_log 2 10 rope" << endl;

ItemNonTool* checkNT1 = static_cast<ItemNonTool*>((*inventory)(0, 0));
```

```
ItemNonTool* checkNT2 = static_cast<ItemNonTool*>((*inventory)(0, 2));

ItemTool* check1 = static_cast<ItemTool*>((*inventory)(0, 4));
ItemTool* check2 = static_cast<ItemTool*>((*inventory)(0, 6));

cout << checkNT1->getName() << endl;
cout << checkNT2->getQuantity() << endl;
cout << check1->getDurability() << endl;
cout << check2->getType() << endl;
...
```

4. Pembagian Tugas

Modul (dalam poin spek)	Designer	Implementer
Mekanisme item	13520051, 13520099	13520108, 13520138, 13520159
Mekanisme inventory	13520108, 13520138	13520051, 13520099, 13520159
Mekanisme crafting	13520099, 13520108, 13520138	13520051, 13520159
Konfigurasi item	13520051, 13520159	13520099, 13520108, 13520138
Konfigurasi resep	13520099, 13520159	13520051, 13520108, 13520138
Unit testing	13520051, 13520108	13520099, 13520138, 13520159