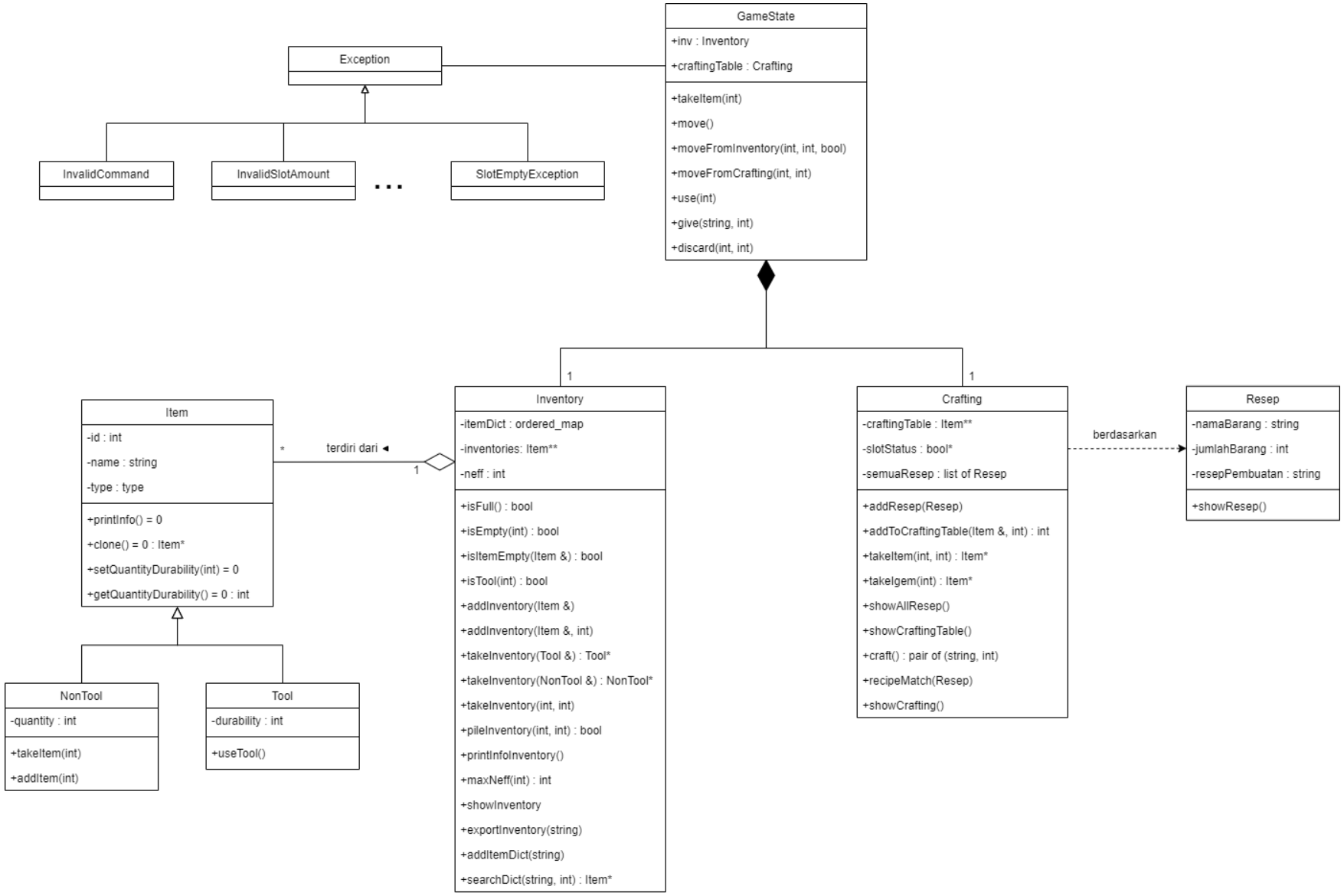
Kelas : 01
Nomor Kelompok : 09
Nama Kelompok : ngOOP
1. 13520019 / Maharani Ayu Putri Irawan
2. 13520034 / Bryan Bernigen
3. 13520040 / Ng Kyle
4. 13520088 / Rio Alexander Audino
5. 13520115 / Maria Khelli
Asisten Pembimbing : Jonathan Yudi Gunawan

1. Diagram Kelas

Pada diagram kelas ini, kami tidak memasukkan 4 sekawan (constructor, destructor, copy constructor, dan operator assignment=), setter, dan getter karena sebagian besar sifatnya merupakan implementasi secara umum yang sudah diketahui. Hanya saja, perlu diperhatikan bahwa kami hanya memberi cctor dan operator assignment bila diperlukan. Misalnya, pada kelas yang tidak memiliki atribut free store object, kami tidak mengimplementasikan cctor dan opperator assignment.

Kelebihan dari implementasi diagram kelas kami adalah kami menggunakan sebuah kelas besar yang menampung state game (skeleton), yaitu GameState yang instancenya hanya satu. Dengan demikian, abstraksi tercipta pada program main.cpp karena seluruh command ditangani di dalam objek GameState yang dibuat. Namun, kekurangan dari implementasi kami adalah inkonsistensi penggunaan reference dan pointer. Banyak fungsi yang mengembalikan pointer, tetapi beberapa fungsi masih memerlukan argumen berbentuk reference sehingga tidak sesuai. Dengan demikian, diperlukan cara lain dalam implementasi yang berpotensi membuat logika program sulit dipahami oleh programmer lain.

Kami memilih desain seperti ini setelah beberapa iterasi memodifikasi kelas. Kami memutuskan untuk membuat pointer dengan kata kunci *new* saat membuat objek. Hal ini dilakukan agar objek tidak hilang bersama siklus hidup fungsi/method yang dijalankan. Namun, karena implementasinya demikian, manajemen memori untuk melakukan *delete* menjadi sulit sehingga memungkinkan terciptanya zombie memory. Diagram kelas yang dibuat akan ditampilkan di halaman selanjutnya.



Gambar 1. Diagram Kelas

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

```
class Tool : public Item
```

Gambar 2. Inheritance pada kelas Tool

```
class NonTool : public Item
```

Gambar 3. Inheritance pada kelas NonTool

Dalam program ini, konsep *inheritance* dapat dilihat pada kelas Item, Tool, dan NonTool. Dapat dilihat pada diagram kelas bahwa NonTool dan Tool merupakan kelas anak dari Item. Kami mendesain demikian karena Tool dan NonTool memiliki karakteristik (atribut dan fungsi anggota) yang sama. Contohnya, setiap Tool dan NonTool harus memiliki id, nama, dan tipe. Agar tidak terjadi redundansi, diperlukan kelas orang tua untuk menggeneralisasi kelas anak.

Di sisi lain, konsep *polymorphism* diterapkan saat objek Item (yang berbentuk Tool/NonTool tetapi tidak diketahui saat *compile-time*) memanggil fungsi anggota yang bersifat pure virtual. Dengan demikian, fungsi anggota yang dieksekusi adalah fungsi anggota dari kelas anak Item, yaitu Tool/NonTool, bergantung pada jenis atau tipe objek yang diketahui pada saat *run-time* (*run-time polymorphism*).

Keuntungan kedua konsep ini adalah, programmer tidak harus mendefinisikan secara eksak tipe kelas anak pada kode program (apakah tipenya Tool/NonTool) sehingga dapat membiarkan tipe objek terdefinisi saat masuk ke tahap *run-time* (*late binding/extremely late binding*).

2.2. Method/Operator Overloading

Program ini menggunakan konsep method overloading dalam modul Inventory dan Crafting. Berikut merupakan contoh pendefinisian method overloading yang digunakan.

```
// Add item to inventory
// 1. add item, unknown slotID
void addInventory(Item&);
// 2. add item to specific inventory ID
void addInventory(Item&, int);
```

Gambar 4. addInventory pada modul inventory

```
// Take item from inventory
// 1. take Tool from Inventory
Tool* takeInventory(Tool&);
// 2. take NonTool from Inventory
NonTool* takeInventory(NonTool&);
```

Gambar 5. takeInventory pada modul inventory

```
// 3. take item from inventory
Item* takeInventory(Item&);
// 4. take item with specific slot id and quantity
Item* takeInventory(int, int);
```

Gambar 6. takeInventory pada modul inventory

```
Item *takeItem(int pos, int quantity);
Item *takeItem(int pos);
```

Gambar 7. takeItem pada modul Crafting

Dalam potongan kode sumber di atas, dapat terlihat bahwa *method* yang didefinisikan memiliki nama yang sama namun tipe data parameter masukan yang berbeda. Overloading perlu dilakukan karena untuk sebuah *method* dapat memiliki nama yang sama namun tipe data parameter masukan yang berbeda, ataupun jumlah parameter masukan yang berbeda. Untuk tiap tipe data yang berbeda, operasi yang dilakukan berbeda pula. Misalnya, mengambil item dari inventory dengan *method* takeInventory untuk item yang berjenis Tool dan NonTool pasti berbeda. Pada pengambilan Tool, berapapun durability-nya, sebuah Tool pasti akan terambil seluruhnya. Sedangkan pada pengambilan NonTool, jumlah item NonTool yang diambil dapat bervariasi tergantung keinginan pengguna. Oleh karena itu, konsep *object-oriented programming* yang paling tepat didefinisikan adalah *method overloading.*

Sementara itu, operator overloading digunakan dalam modul Inventory untuk meng-overload operator assignment (=). Berikut merupakan tangkapan layar program yang telah dibuat.

```
inventory &inventory::operator=(const inventory &inv)
{
    delete[] this->inventories;
    this->neff = inv.neff;
    this->inventories = new Item *[27];
    for (int i = 0; i < 27; i++)
    {
        this->inventories[i] = inv.inventories[i];
    }

    return *this;
}
```

Gambar 8. Penggunaan operator overloading

Dalam kasus ini, operator overloading digunakan untnuk mengganti makna simbol =. Dengan adanya operator assignment yang dibuat ini, suatu inventory dapat disalin sementara untuk pemrosesan tertentu.

2.3. Template & Generic Classes

Dalam program yang dibuat, tidak digunakan konsep template dan generic class di manapun. Hal ini dikarenakan, tipe data yang digunakan dalam program ini tidak terlampau bervariasi. Untuk beberapa cuplikan kode yang mirip, tidak pula tepat untuk digunakan template dan generic class karena operasi yang dilakukan berbeda. Sehingga, akan lebih cocok untuk menggunakan method overloading. Hal ini kami pandang lebih menguntungkan karena tidak memaksakan penggunaan konsep *object-oriented programming* pada program kami.

2.4. Exception

Di dalam pemrograman berbasis objek, kita juga mengenal konsep exceptions. Exception merupakan sebuah mekanisme penanganan error dan kasus khusus yang terpusat (*centralized proccessing of errors and exceptional situations).* Penggunaan exceptions akan lebih menguntungkan dari metode prosedural biasa karena proses penanganan error yang terpusat dan dapat menangani beberapa error secara sekaligus.

```
class InvalidCommand : public exception
{
public:
    const char* what() const throw()
    {
        return "We dont know what are you saying?! Please try it again.\n";
    }
};
```

Gambar 9. Contoh exception

Dalam program ini, contoh penerapan konsep exception dapat dilihat pada file exception.hpp. Dapat dilihat penanganan error ketika command yang salah di-handle oleh sebuah kelas InvalidCommand. Keuntungan dari design penanganan error yang terpusat adalah sangat menguntungkan proses pengerjaan , debugging, dan eksekusi program. Keterbacaan kode program semakin meningkat dan proses pengembangan kode menjadi lebih mudah.

2.5. C++ Standard Template Library

```
//Mendapatkan list of inventory/crafting yang akan didistrirbusikan
getline(cin, namaSlot);
namaSlot.erase(0,1);
for(auto ch : namaSlot)
{
    if (ch == ' ')
    {
        slots.push_back(temp);
        temp = "";
        continue;
    }
    temp += ch;

}
slots.push_back(temp);
```

Gambar 10. Penggunaan STL Vector slots

```
for (auto& ptr : this->semuaresep)
{
    bool found = true;
    if (craftingstring.find(ptr.getResep()) != string::npos)
    {
        string::iterator it;
        string str = craftingstring;
        str.replace(str.find(ptr.getResep()), ptr.getResep().length(), "-");
        for (it = str.begin(); it != str.end(); it++)
```

Gambar 11. Penggunaan STL List semuaresep

Konsep C++ Standard Template Library adalah salah satu konsep dalam pemrograman C++ yang memudahkan code dalam membangun suatu program. Dalam suatu STL, terdapat berabagai elemen seperti struktur data, fungsi, dan kelas yang bisa mempercepat pengerjaan program. Dalam program ini, contoh penerapan konsep C++ Standard Template Library dapat dilihat di beberapa bagian program, seperti proses pembacaan item dari config file, proses pembacaan resep, dan proses crafting. Gambar di atas merupakan potongan kode dari proses command move (kiri) dan proses pembacaan resep (kanan). Bisa dilihat terdapat menggunakan STL sebagai iterator yang mengiterasi tiap element dalam string. Keuntungan dari penggunaan STL pada program kami adalah tidak memerlukan implementasi dari struktur data, variable yang lebih dinamis, dan method sudah banyak yang terdefinisi.

2.6. Konsep OOP lain

Pada tugas ini, kami mengimplementasikan abstract base class, aggregation, dan composition.

Pertama, konsep abstract base class (ABC) diterapkan pada kelas Item yang memiliki anak Tool dan NonTool. Kelas abstrak ini diperlukan agar memaksa kelas anak untuk mengimplementasikan beberapa method yang harus ada, misalnya, printInfo(), clone(), get dan setDurabilityQuantity(). Dengan demikian, jika kelas Inventory memiliki atribut matrix of Item, kelas tersebut bisa langsung menangani Item tersebut tanpa harus mempedulikan apakah Item tersebut berjenis Tool atau NonTool (penanganan secara umum). Kami tidak mengimplementasikan Item sebagai kelas nonabstrak karena kata penanganan method yang disebutkan berbeda untuk kelas Tool dan NonTool sehingga kata kunci abstrak diperlukan. Sebagai contoh, pada printInfo, kelas Tool memberikan informasi durability, sedangkan kelas NonTool memberikan informasi quantity.

Kedua, agregasi dipakai dalam hubungan kelas Inventory dan Item. Kelas Inventory terdiri dari beberapa (atau tidak sama sekali) kelas Item. Karena lifetime Inventory tidak bergantung pada keberadaan instansi kelas Item, dan sebaliknya, implementasi ini dikategorikan sebagai agregasi.

Ketiga, berbeda dengan poin kedua, implementasi komposisi mengharuskan keberadaan suatu kelas bergantung pada kelas lain. Pada program yang kami buat, kelas GameState memiliki relasi “has a” dengan kelas Inventory dan Crafting. Maka, lifetime Inventory dan Crafting bergantung pada keberadaan kelas GameState. Jika objek kelas GameState dihilangkan, otomatis objek Inventory dan Crafting yang ada di dalamnya ikut hilang.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Multiple Crafting

Pada program ini, *multiple crafting* diimplementasikan pada modul crafting, tepatnya pada *method* craft.

```
pair<string, int> Crafting::craft()
```

Gambar 12. *Method* craft()

Method ini mengembalikan nama item hasil *crafting* dan jumlahnya. Pendekatan yang digunakan adalah sebagai berikut:

```
for (int i = 0; i < 9 ; i++ ){
    if(this->slotStatus[i]){
        if(this->craftingtable[i]->getType() == "TOOL"){
            Tools++;
        }
        else{
            NonTools++;
        }
    }
}
if (Tools == 1){
    throw new NoneCreated();
```

Gambar 13. Pengecekan isi slot crafting

Pertama-tama, dilakukan pengecekan isi slot crafting untuk menghitung jumlah item tool dan NonTool yang tersedia. Apabila tool hanya ada 1, tidak dilakukan crafting karena crafting untuk meningkatkan durability tool hanya dapat dilakukan dengan 2 tool.

```
string craftingstring = "";
string flippedcraftingstring = "";
for (int i = 0; i < 9; i++)
{
    if (!this->slotStatus[i])
    {
        craftingstring += "-";
    }
    else
    {
        if (this->craftingtable[i]->getQuantityDurability() < minQnt){
            minQnt = this->craftingtable[i]->getQuantityDurability();
        }
        craftingstring += this->craftingtable[i]->getType();
    }
}
```

```
for (int i = 0; i < 3; i++)
{
    for (int j = 2; j >= 0; j--)
    {
        if (!this->slotStatus[i * 3 + j])
        {
            flippedcraftingstring += "-";
        }
        else
        {
            flippedcraftingstring += craftingtable[i * 3 + j]->getType();
        }
    }
}
```

Gambar 14 dan 15. Pencarian jumlah item yang bisa di-craft dan string konfigurasi crafting table

Cuplikan kode tersebut mencari kuantitas terkecil item yang terdapat pada crafting table sekaligus menyimpan nama item pada crafting table sebagai string untuk mengecek konfigurasi crafting. Cuplikan kode di bagian kanan digunakan untuk menukar item pada kolom paling kiri dan paling kanan dalam crafting table untuk translasi pada sumbu y.

```
for (auto& ptr : this->semuaresep)
{
    bool found = true;
    if (craftingstring.find(ptr.getResep()) != string::npos)
    {
        string::iterator it;
        string str = craftingstring;
        str.replace(str.find(ptr.getResep()), ptr.getResep().length(), "-");
        for (it = str.begin(); it != str.end(); it++)
        {
            if (*it != '-')
            {
                found = false;
                break;
            }
        }
        if (found)
        {
            takeAll(minQnt);
            return (make_pair(ptr.getNamaBarang(), ptr.getJumlah()*minQnt));
        }
    }
```

```
    if (flippedcraftingstring.find(ptr.getResep()) != string::npos)
    {
        bool found = true;
        string::iterator it;
        string str = flippedcraftingstring;
        str.replace(str.find(ptr.getResep()), ptr.getResep().length(), "-");
        for (it = str.begin(); it != str.end(); it++)
        {
            // Print current character
            if (*it != '-')
            {
                found = false;
                break;
            }
        }
        if (found)
        {
            takeAll(minQnt);
            return (make_pair(ptr.getNamaBarang(), ptr.getJumlah()*minQnt));
        }
    }
}
```

Gambar 16 dan 17. Crafting sesuai recipe dengan pendekatan *string matching*.

Untuk setiap resep akan dicek apakah konfigurasi crafting saat ini dapat di-craft sesuai resep tersebut dengan *string matching*. Pengecekan akan dilakukan baik pada konfigurasi crafting table yang awal maupun ditukar. Jika item yang dibutuhkan ditemukan, crafting resep tersebut akan berhasil. Jika tidak, akan dilanjutkan pada resep selanjutnya. Apabila sampai akhir tidak ditemukan resep yang dapat dibuat, dikemballikan exception sebagai berikut.

```
throw new NoneCreated();
```

Gambar 18. *Exception* yang dilempar jika tidak bisa melakukan crafting

3.1.2. Item dan Tool Baru

Dalam program ini, selain item dan tool yang didefinisikan pada spesifikasi tugas besar, ditambahkan pula beberapa tool lain, antara lalin:

```
25 IRON_HELMETS - TOOL
26 IRON_BOOTS - TOOL
27 IRON_CHESTPLATES - TOOL
28 IRON_LEGGINGS - TOOL
```

Gambar 19. Tool baru yang ditambahkan pada program

Berikut merupakan recipe yang dibuat untuk setiap tool baru yang ditambahkan:

```
3 3
IRON_INGOT IRON_INGOT IRON_INGOT
IRON_INGOT - IRON_INGOT
- - -
IRON_HELMETS 1
```

```
2 3
IRON_INGOT - IRON_INGOT
IRON_INGOT - IRON_INGOT
IRON_BOOTS 1
```

```
3 3
IRON_INGOT - IRON_INGOT
IRON_INGOT IRON_INGOT IRON_INGOT
IRON_INGOT IRON_INGOT IRON_INGOT
IRON_CHESTPLATES 1
```

```
3 3
IRON_INGOT IRON_INGOT IRON_INGOT
IRON_INGOT - IRON_INGOT
IRON_INGOT - IRON_INGOT
IRON_LEGGINGS 1
```

Gambar 20, 21, 22, dan 23. Konfigurasi *recipe* untuk (dari kiri ke kanan) iron helmet, iron boots, iron chestplates, dan iron leggings

Fungsionalitas setiap fungsi untuk tiap tool baru yang ditambahkan sama seperti item primitif.

3.1.3. Unit Testing Implementation

3.1.3.1. Item Testing

Testing dilakukan sebagai berikut:

- Tool default constructor

```
================DEFAULT TOOL CONSTRUCTOR===============
ID          : 0
Name        : -
Type        : -
Durability : 10
```

Gambar 24. Testing Item 1
Membuat tool dengan default constructor (no parameter).

- Tool user defined default durabilty

```
===============CREATE IRON SWORD DEFAULT================
ID          : 23
Name        : IRON_SWORD
Type        : TOOL
Durability : 10
```

Gambar 25. Testing Item 2
Membuat tool dengan user defined constructor dengan parameter nama,id, dan type.

- Tool user defined with durability

```
==========CREATE WOODEN PICKAXE 1 DURABILITY===========
ID          : 12
Name        : Pickaxe
Type        : TOOL
Durability : 1
```

Gambar 26. Testing Item 3

Membuat tool dengan user defined constructor dengan parameter nama, id, type, durability. Asumsi selalu benar (durability 1—10).

- Use Tool

```
======================USE PICKAXE======================
ID         : 12
Name       : Pickaxe
Type       : TOOL
Durability : 0
```

Gambar 27. Testing Item 4

Menggunakan tool (use) yaitu mengurangi durability sebesar 1 durability. Pada contoh menggunakan pickaxe yang sebelumnya memiliki durability 1 sehingga durability setelah use menjadi 0.

- Over use tool error

```
====================OVERUSE PICKAXE====================
Item Overused! Should be destructed!
ID         : 12
Name       : Pickaxe
Type       : TOOL
Durability : 0
```

Gambar 28. Testing Item 5

Menggunakan tool (use) akan error jika durability menjadi negatif. Throw ToolOverused exception. Pada contoh menggunakan pickaxe sebelumnya, karena durability sudah 0 maka timbul exception dan durability tetap 0.

- Nontool default constructor

```
============DEFAULT NONTOOL CONSTRUCTOR==============
ID          : 0
Name        : -
Type        : -
Quantity    : 1
```

Gambar 29. Testing Item 6

Membuat nontool dengan default constructor (no parameter).

- Nontool user defined default quantity

```
================CREATE OAK LOG DEFAULT================
ID          : 1
Name        : OAK_LOG
Type        : LOG
Quantity    : 1
```

Gambar 30. Testing Item 7

Membuat nontool dengan user defined constructor dengan parameter id, name, type.

- Nontool user defined with quantity

```
==================CREATE OAK PLANK 62=================
ID          : 3
Name        : OAK_PLANK
Type        : PLANK
Quantity    : 62
```

Gambar 31. Testing Item 8

Membuat nontool dengan user defined constructor dengan parameter id, name, type, quantity.

- Take from nontool

```
====================TAKE OAK PLANK 1==================
ID          : 3
Name        : OAK_PLANK
Type        : PLANK
Quantity    : 61
```

Gambar 32. Testing Item 9

Mengambil dari nontool sebesar quantity. Pada contoh menggunakan oak plank dengan quantity 62 sebelumnya sebanyak 1 sehingga berkurang menjadi 61.

- Add to nontool

```
====================ADD OAK PLANK 1===================
ID          : 3
Name        : OAK_PLANK
Type        : PLANK
Quantity    : 62
```

Gambar 33. Testing Item 10

Menambahkan ke nontool sebesar quantity. Pada contoh menggunakan oak plank yang sudah berubah menjadi 61 sebanyak 1 sehingga quantity oak plank pada akhir add menjadi 62.

- Over add to nontool

```
==================OVERADD OAK PLANK 10================
Resulting Quantity Over Than 64!
ID          : 3
Name        : OAK_PLANK
Type        : PLANK
Quantity    : 62
```

Gambar 34. Testing Item 11

Ketika add menimbulkan efek quantity melebihi dari 64 maka akan throw OverQuantityException. Pada contoh oak plank sudah memiliki quantity 62, sehingga jika di add sebanuak 10 akan menjadi 72 melebihi maximum 64 sehingga menimbulkan exception dan quantity tidak diubah.

- Over take form nontool

```
==================OVERTAKE OAK PLANK================
Cant fullfill request of 65 from 62 !
ID          : 3
Name        : OAK_PLANK
Type        : PLANK
Quantity    : 62
```

Gambar 35. Testing Item 12

Ketika take menimbulkan efek quantity kurang dari 0, maka akan throw QuantityNotMetException. Pada contoh oak plank memiliki quantity sebesar 62 sehingga jika diambil sebanyak 65 akan berdampak pada quantity negatif sehingga timbul exception dan quantity tidak diubah.

3.1.3.2. Inventory Testing

Testing pada Inventory dilakukan dengan skenario berikut:

- Construct inventory

```
inventory temp;
temp.printInfoInventory();
```

```
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 36. Testing Inventory 13

Inventory dibuat *fixed-size* 27. Saat diinisialisasi, inventory diisi item kosong.

- Pemanggilan printInfoInventory()

```
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 37. Testing Inventory 14

Karena inventory baru diinisialisasi, seluruh slot nya masih kosong.

- Penambahan item NonTool dengan jumlah lebih dari 64 ke inventory

```
[03 64] [03 01] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 38. Testing Inventory 15

Sebanyak 65 PLANK dimasukkan ke dalam inventory. Karena setiap slot maksimal berisi 64 item Non-Tool, maka sebanyak 64 PLANK dimasukkan ke slot 0 dan 1 PLANK dimasukkan ke slot 1.

- Pengambilan satu item NonTool dengan masukan item saja dari inventory

```
[03 01]
[03 63] [03 01] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 39. Testing Inventory 16

Sebuah PLANK diambil dari inventory. Pengambilan dilakukan pada slot inventory berindeks terkecil. Oleh karena itu PLANK pada slot 0 bersisa 63.

- Penambahan item Tool ke inventory

```
[03 63] [03 01] [12 05] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 40. Testing Inventory 17

Sebuah PICKAXE dengan durability 5 dimasukkan ke inventory. Item ini dimasukkan ke slot inventory berindeks terkecil.

- Pengambilan satu item Tool dengan masukan item saja dari inventory

```
[12 05]
[03 63] [03 01] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 41. Testing Inventory 18

Sebuah PICKAXE diambil dari inventory, sehingga inventory pada slot 2 menjadi kosong.

- Penambahan item ke inventory dengan masukan item dan slotID

```
[03 61] [03 01] [EMPTY] [03 05] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 42. Testing Inventory 19

Lima buah PLANK ditambahkan ke inventory pada slot ke-3. Oleh karena itu, tidak dilakukan penumpukan item Non-Tool.

- Pengambilan item dari inventory dengan masukan slotID dan kuantitas

```
[03 61] [03 01] [EMPTY] [03 04] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 43. Testing Inventory 20

Sebuah PLANK pada slot ke-2 inventory diambil. Sehingga jumlah PLANK pada slot ke-2 berkurang 1 dan tidak mempengaruhi PLANK pada slot lain.

- Pengecekan apakah inventory full

```
is Inventory full? 0
```

Gambar 44. Testing Inventory 21

Inventory temp belum semuanya terisi. Oleh karena itu mengembalikan nilai 0.

- Pengecekan apakah inventory pada slot tertentu empty

```
is Inventory at id 4 empty? 1
```

Gambar 45. Testing Inventory 22

Inventory pada slot id 4, seperti ditunjukkan pada poin sebelumnya, bertuliskan EMPTY sehingga mengembalikan nilai 1.

- Pengecekan apakah inventory pada slot tertentu adalah tool

```
is Inventory slot 2 a Tool ? 0
```

Gambar 46. Testing Inventory 23

Inventory pada slot 2 kosong, sehingga mengembalikan nilai 0.

- Penumpukan item pada slot tertentu ke slot lain

```
Success Piling inventory item! 1

[03 62] [EMPTY] [EMPTY] [03 04] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
[EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY] [EMPTY]
```

Gambar 47. Testing Inventory 24

PLANK pada slot inventory ke-1 ditumpuk ke slot inventory ke-0 sehingga slot inventory ke-1 kosong sementara slot inventory ke-0 bertambah 1.

- *Export* inventory ke file

```
3:62
0:0
0:0
3:4
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
0:0
```

Gambar 48. Testing Inventory 25

Isi dari inventory di-*export* ke dalam sebuah file example.out yang berisi seperti tangkapan layar.

- Destruct inventory

```
temp.~inventory();
```

Gambar 49. Testing Inventory 26

3.1.3.3. Crafting Testing

Testing pada crafting dilakukan pada kasus-kasus sebagai berikut:

- Pembacaan setiap resep dari konfigurasi.

```
==========================LIST OF RECIPES==============================
BIRCH_LOG BIRCH_PLANK 4
DIAMONDDIAMOND-DIAMONDSTICK--STICK DIAMOND_AXE 1
DIAMONDDIAMONDDIAMOND-STICK--STICK DIAMOND_PICKAXE 1
DIAMOND--DIAMOND--STICK DIAMOND_SWORD 1
IRON_INGOTIRON_INGOT-IRON_INGOTSTICK--STICK IRON_AXE 1
IRON_INGOT-IRON_INGOTIRON_INGOT-IRON_INGOT IRON_BOOTS 1
IRON_INGOT-IRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOT IRON_CHESTPLATES 1
IRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOT-IRON_INGOT IRON_HELMETS 1
IRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGETIRON_NUGGET IRON_INGOT 1
IRON_INGOTIRON_INGOTIRON_INGOTIRON_INGOT-IRON_INGOTIRON_INGOT-IRON_INGOT IRON_LEGGINGS 1
IRON_INGOT IRON_NUGGET 9
IRON_INGOTIRON_INGOTIRON_INGOT-STICK--STICK IRON_PICKAXE 1
IRON_INGOT--IRON_INGOT--STICK IRON_SWORD 1
OAK_LOG OAK_PLANK 4
SPRUCE_LOG SPRUCE_PLANK 4
PLANK--PLANK STICK 4
STONESTONE-STONESTICK--STICK STONE_AXE 1
STONESTONESTONE-STICK--STICK STONE_PICKAXE 1
STONE--STONE--STICK STONE_SWORD 1
PLANKPLANK-PLANKSTICK--STICK WOODEN_AXE 1
PLANKPLANKPLANK-STICK--STICK WOODEN_PICKAXE 1
PLANK--PLANK--STICK WOODEN_SWORD 1
```

Gambar 50. Testing Crafting 1

Recipe diambil dari config, berikut tampilan recipe yang terbaca dan item serta hasil jika dibuat sebuah item yang memenuhi recipe tersebut.

- Menambahkan item kepada multiple slot.

```
=========================ADD PLANKS TO C2 and C5============================
                        [EMPTY] [EMPTY] [04 04]PLANK
                        [EMPTY] [EMPTY] [04 05]PLANK
                        [EMPTY] [EMPTY] [EMPTY]
```

Gambar 51. Testing Crafting 2
Menambahkan plank sebanyak 4 ke slot C2 dan 5 ke slot C5.

- Crafting Item

```
============================CREATE STICKS=================================
Created Item STICK with quantity of 16
                        [EMPTY] [EMPTY] [EMPTY]
                        [EMPTY] [EMPTY] [04 01]PLANK
                        [EMPTY] [EMPTY] [EMPTY]
```

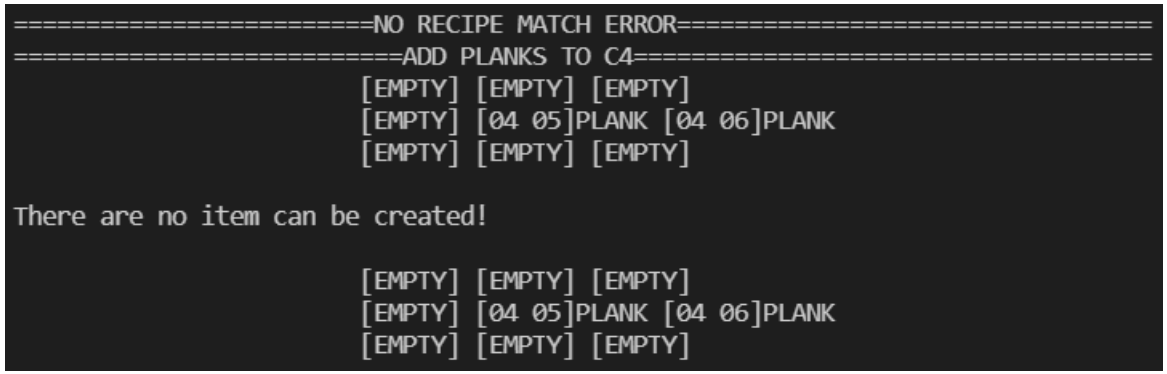
Gambar 52. Testing Crafting 3
Craft item dari crafting table, ditemukan recipe yang cocok lalu akan melakukan create item tersebut sebanyak mungkin dari kondisi crafting table saat itu dan akan mengurangi sejumlah item pada crafting table sesuai recipe. Pada contoh tersisa 1 buah plank dan telah dihasilkan STICK sebanyak 4 x 4 quantity.

- Stacking Item

```
=========================TRY STACKING ITEM==============================
                        [EMPTY] [EMPTY] [EMPTY]
                        [EMPTY] [EMPTY] [04 06]PLANK
                        [EMPTY] [EMPTY] [EMPTY]
```

Gambar 53. Testing Crafting 4
Melakukan move ke suatu slot dan item dapat distack. Stack hanya dapat dilakukan jika item bertipe sama dan bukan berupa tool (non stackable).

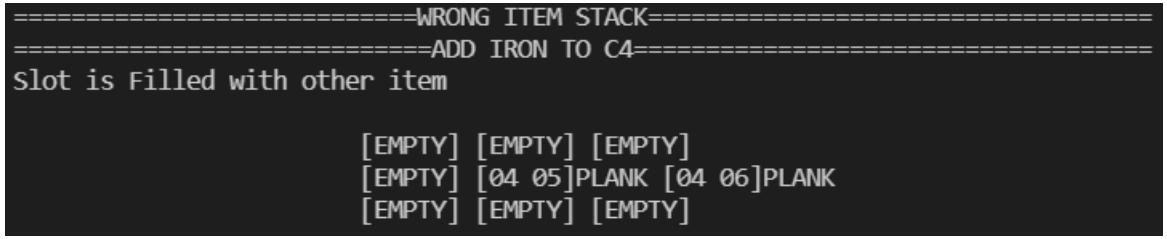
- Fail Crafting



Gambar 54. Testing Crafting 5

Crafting dapat gagal jika tidak terdapat resep yang terpenuhi maupun tool yang dapat difix. Akan throw NoneCreated atau ToolNotMatchExc.

- Wrong Stacking



Gambar 55. Testing Crafting 6

Kasus ketika item distack tidak sesuai typenya.

3.2. Bonus Kreasi Mandiri

Dalam program ini, command MOVE tidak terpaku pada jumlah N masukan. Player dapat mengisikan jumlah inventory secara bebas dan sistem akan membagikan secara otomatis. Sistem pembagian bulat, sehingga inventory yang dimasukkan dalam urutan akhir akan mendapatkan jumlah item yang lebih sedikit.

Contoh: MOVE I0 10 I1 I2 I3.
Menjalankan command ini akan memindahkan 10 item dari inventory slot ke-0 ke inventory slot ke-1, 2, dan 3. Distribusi jumlah item yang dibagikan, urut dari masukan ID slot inventory, adalah 4, 3, dan 3.

Dalam program dibuat command INFO yang akan memberikan info nama setiap item yang ada pada inventory. Info berupa tuple id dan nama dari item.

Dalam program dibuat command CHECK [SLOTID] yang akan memberikan info lengkap untuk item yang berada pada SLOTID tersebut (jika ada).

4. Pembagian Tugas

Modul (dalam poin spek)	Designer	Implementer
Resep	13520034 13520088	13520034 13520088
Item	13520034	13520019 13520034 13520115
Item Tool	13520034	13520019 13520034 13520115 13520040
Item Non-Tool	13520115	13520019 13520034 13520115 13520040
Inventory	13520019	13520019 13520034 13520040 13520088 13520115
Crafting	13520040	13520034 13520040 13520115

Operasi dan Command	13520019 13520034 13520040 13520088 13520115	13520019 13520034 13520040 13520088 13520115
Exceptions	13520040	13520019 13520034 13520040 13520088 13520115

5. Lampiran

Tugas Besar 1 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022

Kelas : 01
Nomor Kelompok : 09
Nama Kelompok : ngOOP
1. 13520019 / Maharani Ayu Putri Irawan
2. 13520034 / Bryan Bernigen
3. 13520040 / Ng Kyle
4. 13520088 / Rio Alexander Audino
5. 13520115 / Maria Khelli

Asisten Pembimbing : Jonathan Yudi Gunawan

1. Konten Diskusi

1) Bagaimana tampilan SHOW kalau terlalu panjang?
   - Tidak harus mengikuti template. Silahkan di-*custom* sesuai kebutuhan.
2) Dimana memakai template?
   - Tidak harus ada semua. Di laporan jelaskan dipakai dimana saja. Kalau ada yang tidak cocok, jelaskan kenapa.
3) Move itu bisa 1 inventory ke bbrp crafting. Bisa kah kalau ada command yg nyelip, atau sangat tidak diperbolehkan? Misal MOVE I1 3 C1 I1 I2. Itu bisa kah?
   - Nyebar masuk ke crafting dan inventory ya? Di spek tidak ada, di minecraft nggak bisa ya. Terserah mau dibikin bisa/nggak bisa. Kalau nggak bisa harus cek sampe belakang. Dibebaskan saja.
4) Kalau inventory di take sampai habis, indeks tidak geser?
5) Pasti ada tempat buat hasil craft?
   - Tidak dijamin. Item di slot crafting tidak berubah, craft tidak berubah
6) Di spek kalau di craft semuanya hilang. Apa benar?
   - Iya, asumsi masukkan craft satu-satu. Baru pikir bbrp hari lalu. Kalau craft dimasukkan 1 slot lebih dari 1, apalagi kalau kerja bonus. Nggak hilang, berkurang sesuai jenis craftingnya.
7) Laporan memang landscape?
   - Iya, biar masukin class diagram gampang
8) Diagram classnya asosiasi semua ya, sama inheritance, nggak ada yang lain? Kalau di RPL kan ada hubungan aneh-aneh?
   - Kalau salah dikit-dikit ga ngaruh, yg penting kelihtaan inheritance, atribut, method.

Tugas Besar 1 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022

9) Untuk exception, bagaimana class diagramnya? Kan banyak
   - Exceptionnya tidak. Class diagram hanya garis-garis inheritance nya. Nggak usah tulis semuanya, 1,2,…n. Tetap ditaruh, kalau kebanyakan titik-titik saja.
10) Kita kan buat kelas abstrak, simpan game state. Tapi dibuat supaya ada atribut publik, apa masalah?
   - Tidak masalah. Design pattern ada pola-polanya, dibagi 3: creational, structural, sama functional. Pokoknya biar ngelola class lebih gampang. Gamestate cocok jadi singleton. Nggak masalah. Singeton design pattern: https://refactoring.guru/design-patterns/singleton. Di Tubes 1 nggak usah pakai design pattern, dipakai di tubes 2 di Java.
11) Semisal niat, mau bikin GUI, harus linux?
   - Tidak, bebas, boleh di web juga. Idenya manggil executable nya. Kalau mau di CPP ada Qt5 sering dipakai
12) Dalam singleton, kalau create > 1, nggak bisa ya?
   - Ya, kalau dibikin 2 akan return pointer ke yang pertama. Atribut tidak pengaruh
13) Yang type kalau Tool pasti TOOL ya?
   - Kalau mau bagi-bagi lagi silakan. Kalau merasa tidak bisa boleh banyak.

Jelaskan 6 konsep OOP dipakai dimana saja, kalau tidak jelaskan kenapa. Jelaskan bonus yang dikerjakan dan pembagian tugas. Template nya spt nya ada perubahan. Download ulang.

Crafting auto masuk inventory. Kalau bisa ditumpuk, ditumpuk. Kalau nggak ada di index paling kecil

2. Screenshot Bukti

Tugas Besar 1 IF2210/Pemrograman Berorientasi Objek sem. 2 2021/2022