

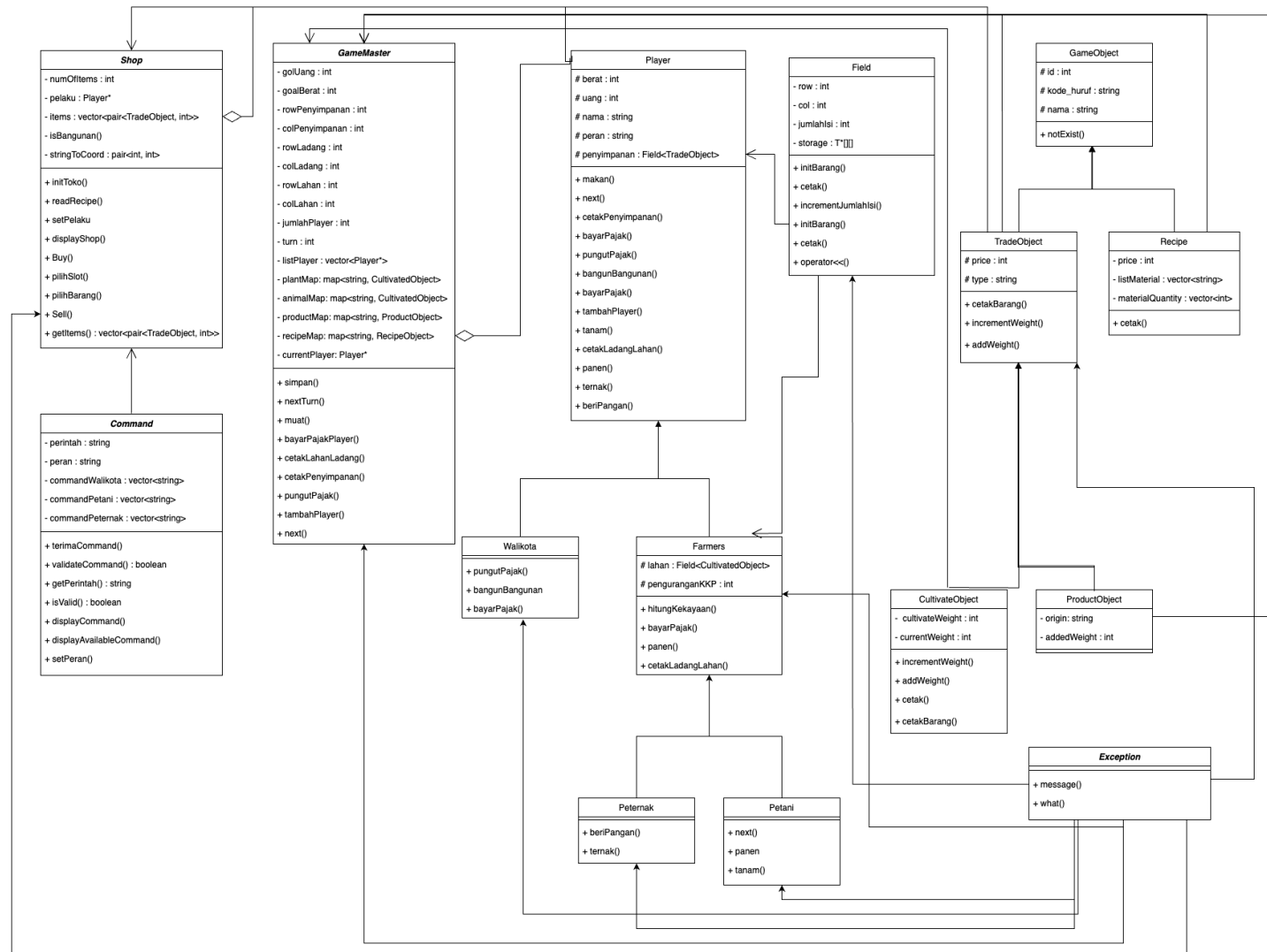
Kode Kelompok : ENC

Nama Kelompok : Encapsulators

1. 13522024 / Kristo Anugrah
2. 13522038 / Francesco Michael Kusuma
3. 13522061 / Maximilian Sulistiyo
4. 13522075 / Marvel Pangondian
5. 13522101 / Abdullah Mubarak

Asisten Pembimbing : Hilya Fadhilah Imania / 13520024

1. Diagram Kelas



Link UML : [DiagramKelasOOP1.drawio - draw.io \(diagrams.net\)](https://draw.io)
<https://drive.google.com/file/d/1o2I92TjoipdpepwU615pJg02PYCQkHH2/view?usp=sharing>

Alasan: implementasi S dan L dari SOLID, memudahkan modifikasi program.

Kelebihan: memudahkan exception handling, memudahkan modifikasi kode, memudahkan penerapan S dan L.

Kendala: penggunaan pointer, kejelasan waktu hidup objek, dikarenakan tingginya keterhubungan di beberapa inheritance maka jika kita mengubah kelas anaknya maka cascading dalam perubahan suatu kode.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

```
class Player
{
public:
    int berat;
    int uang;
    string nama;
    string peran;
    Field<TradeObject> penyimpanan;

public:
    Player();
    Player(string, string, int, int, int, int);
    void virtual next() {}
    void virtual makan(TradeObject *);
    void virtual cetakPenyimpanan();
    string getNama();
    string getPeran();
    int getUang();
    void setUang(int uangBaru);
    float persentasePajak(int KKP);
    int virtual bayarPajak();
    virtual void setBarangFirstPenyimpanan(TradeObject *);
    virtual void setBarangFirstLahan(CultivatedObject *);
    virtual void setBarangPenyimpanan(int, int, TradeObject *);
    virtual void setBarangLahan(int, int, CultivatedObject *);
    virtual void pungutPajak(vector<Player *>, int);
    virtual void bangunBangunan(Recipe);
    virtual void tambahPlayer(string);
```

```

class Walikota : public Player
{
public:
    Walikota(string nama, int berat, int uang, int rowPenyimpanan, int colPenyimpanan);
    void pungutPajak(vector<Player *> listPlayer, int num_of_players);
    void bangunBangunan(Recipe recipe);
    void tambahPlayer(string peran) override;
    int bayarPajak() override;
    Field<CultivatedObject> getLahan();
};

```

```

class Farmers : public Player
{
public:
    Field<CultivatedObject> lahan;
    int penguranganKKP;

public:
    Farmers(string nama, string tipe, int penguranganKKP, int berat, int uang, int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);
    int hitungKekayaan();
    int bayarPajak();
    void panen(int, int, int, int, ProductObject *);
    void setBarangFirstLahan(CultivatedObject *) override;
    void setBarangLahan(int row, int col, CultivatedObject *) override;
    vector<pair<pair<int, int>, pair<string, int>>> getAllPosisiNamaBerat() override;
    Field<CultivatedObject> getLahan();
    void cetakLadangLahan();
};

```

```

class Petani : public Farmers
{
public:
    Petani(string nama, int berat, int uang, int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);
    void next();
    void panen(int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan, int prodId, string prodKode, string nama, int prodPrice, string prodDesc);
    void tanam(CultivatedObject *tanaman, int row, int col);
};

```

```

class Peternak : public Farmers
{
public:
    Peternak(string nama, int berat, int uang, int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);
    void beriPangan(int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);

    void ternak(CultivatedObject *hewan, int row, int col);

    void cetakLahan();
};

```

Alasan : Disini kami menggunakan inheritance karena kami merasa bahwa beberapa objek memiliki atribut yang sama, contohnya adalah kelas Walikota dan Farmers, keduanya memiliki atribut berat, uang, peran, nama, dan penyimpanan, namun keduanya pun memiliki atribut dan method yang berbeda sehingga perlu dibedakan.

Kemudian jika diperhatikan semua Player memiliki penyimpanan yang menyimpan objek TradeObject, namun jika dilihat lagi dari alur program kita maka kita memiliki GameObject lain seperti CultivatedObject dan ProductObject, kedua objek ini merupakan turunan dari TradeObject, karena itu kami dapat menerapkan polymorphism untuk menyimpan objek-objek tersebut ke dalam penyimpanan Player sehingga tidak usah membuat beberapa penyimpanan untuk tipe objek berbeda. Juga karena kita menyimpannya sebagai pointer maka kita dapat memanggil method yang terdapat di dalam objek turunannya karena kita membuat virtual methodnya pada objek parentnya (TradeObject)

2.2. Method/Operator Overloading

```
Field<T> operator<<(T *object)
{
    for (int i = 0; i < this->row; i++)
    {
        for (int j = 0; j < this->col; j++)
        {
            if (this->storage[i][j]->getKodeHuruf() == "  ")
            {
                this->setBarang(i, j, object);
                return;
            }
        }
    }
    throw penyimpananPenuhException();
}
```

Alasan: Disini kami mengimplementasikan operator overloading << untuk memasukkan objek ke slot kosong pertama yang berada dalam field. Kami memutuskan hal tersebut untuk meningkatkan readability dari program

2.3. Template & Generic Classes

```
template <class T>
class Field
{
public:
    int row;
    int col;
    int jumlahIsi;
    string tipe;
    vector<vector<T *>> storage;
    vector<T> memory;
```

Alasan: Dalam membuat kelas kami memutuskan untuk menggunakan template class pada saat pengimplementasian Field. Disini storage dapat menyimpan vector of vector of T. Hal ini karena kami membutuhkan dua tipe Field dimana pertama Field penyimpanan yang menyimpan TradeObject dan juga Field lahan yang menyimpan CultivateObject. Dengan menggunakan template class ini maka kita tidak usah membuat dua kelas terpisah,

2.4. Exception

```
#ifndef EXCEPTION_HPP
#define EXCEPTION_HPP
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <iostream>
#include <string>

using namespace std;

class MakananSalahException : public exception
{
public:
    const char *message()
    {
        return "Makanan tidak dapat dimakan oleh hewan ini\n";
    }
    const char *what() const noexcept override
    {
        return "Makanan tidak dapat dimakan oleh hewan ini\n";
    }
};

class uangTidakCukupException : public exception
{
public:
    const char *message()
    {
        return "Uang tidak cukup!\n";
    }
    const char *what() const noexcept override
    {
        return "Uang tidak cukup!\n";
    }
};
```

```
void setBarang(int row, int col, T *object)
{
    if (this->storage[row][col]->getKodeHuruf() != " ")
    {
        throw petakTidakKosongException();
    }
    delete this->storage[row][col];
    this->storage[row][col] = object;

    if (object->getKodeHuruf() != " ")
    {
        this->jumlahIsi++;
    }
}
```

Alasan: exception digunakan untuk mempermudah mengatasi error, menciptakan program yang kokoh, mempermudah dalam *maintenance*. Salah satu contohnya adalah dalam fungsi setBarang pada field, dengan ini kami dapat memastikan bahwa kita tidak akan menimpa barang yang sudah ada kecuali barang tersebut merupakan barang kosong, dengan ini kami dapat mengurangi kelalaian kami dalam menggunakan fungsi fungsi. Selain itu kami juga dapat dengan lebih mudah mendebug kesalahan yang kami lakukan dengan mengthrow message sehingga kita dapat mengetahui secara akurat masalah apa yang terjadi.

2.5. C++ Standard Template Library

```
vector<Player *> listPlayer;
```

```
template <class T>
class Field
{
public:
    int row;
    int col;
    int jumlahIsi;
    string tipe;
    vector<vector<T *>> storage;
```

Alasan: Dalam pembuatan atribut dari beberapa kelas kami memutuskan untuk menggunakan STL vector dalam c++, hal ini karena pada atribut atribut tersebut kami membutuhkan alokasi array dinamis yang mudah untuk diakses dan diubah, contohnya adalah pada listPlayer dalam GameMaster dan juga pada attribut storage pada Field.

```
map<string, CultivatedObject> plantMap;
map<string, CultivatedObject> animalMap;
map<string, ProductObject> productMap;
map<string, Recipe> recipeMap;
```

Alasan: Kemudian selain itu kami memutuskan menggunakan map dalam pembuatan plantMap, map merupakan struktur yang menyimpan key dan juga value. Hal ini sangat cocok dimana kami memiliki kode ID dari tiap objek yang kami jadikan value, dengan itu kami dapat dengan mudah mengakses objek yang diperlukan.

2.6. Encapsulation

```
class Player
{
protected:
    int berat;
    int uang;
    string nama;
    string peran;
    Field<TradeObject> penyimpanan;
```

```
string getNama();
string getPeran();
int getUang();
void setUang(int uangBaru);
```

```
class Peternak : public Farmers
{
public:
    Peternak(string nama, int berat, int uang, int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);
    void beriPangan(int rowPenyimpanan, int colPenyimpanan, int rowLahan, int colLahan);

    void ternak(CultivatedObject *hewan, int row, int col);

    void cetakLahan();
};
```

Alasan: Enkapsulasi disini berarti suatu objek tidak “mengekspos” isi darinya, maka dari itu dari maka kelas kelas yang kami buat memiliki atribut private yang hanya bisa diakses oleh kelas itu sendiri. Oleh karena itu diberi beberapa metode untuk meminta atribut tersebut dan juga beberapa metode untuk mengubah isi nya seperti pada contoh beriPangan dan ternak pada Peternak

2.7. Association

```
class Farmers : public Player
{
public:
    Field<CultivatedObject> lahan;
    int penguranganKKP;
```

Alasan: Dalam pembuatan program ini terdapat beberapa asosiasi. Salah satu contohnya adalah pada Farmers dimana dia has-a Field of CultivatedObject, disini kelas Farmer mengatur masa hidup lahan karena ketika Farmers dihilangkan maka lahan pada Farmers ini akan ikut dihilangkan

3. Bonus Yang dikerjakan

3.1. Bonus Kreasi Mandiri

3.1.1. Diagram UML

Disini kami berusaha membuat diagram kelas dengan sesuai dengan kriteria UML. Mengikuti

4. Pembagian Tugas







Modul (dalam poin spek)	Implementer	Tester
Next	13522061	13522061
Cetak Penyimpanan	13522024, 13522061, 13522101	13522024, 13522061, 13522101
Pungut Pajak	13522061, 13522075	13522061, 13522075
Cetak Ladang dan Cetak Peternakan	13522024, 13522061, 13522101	13522024, 13522061, 13522101
Tanam	13522061, 13522075	13522061, 13522075
Ternak	13522061, 13522075	13522061, 13522075
Bangun Bangunan	13522075	13522075
Makan	13522024, 13522075	13522024, 13522075
Memberi Pangan	13522024, 13522061, 13522075	13522024, 13522061 13522075
Membeli	13522038, 13522075	13522038, 13522075
Menjual	13522038, 13522075	13522038, 13522075
Memanen	13522024, 13522061, 13522101	13522024, 13522101
Simpan	13522024	13522024

Muat	13522024	13522024
Tambah Pemain	13522024	13522024

Lampiran

1. Form Asistensi : [Link Form Asistensi](#)

Form Asistensi Tugas Besar IF2210/Pemrograman Berorientasi Objek Sem. 1 2023/2024	
No. Kelompok/Kode Kelompok : 7 / ENC	
Nama Kelompok : Encapsulators	
Anggota Kelompok (Nama/NIM) :	1. Kristo Amugrah/13522024 2. Francesco Michael Kusuma / 13522038 3. Maximilian Sulistyo / 13522061 4. Marvel Pangondian / 13522075 5. Abdullah Mubarrak / 13522101
Asisten Pembimbing : Hilya Fadhliah Imania / 13520024	
Tanggal : 3 April 2024 Tempat : Laboratorium Pemrograman	Catatan Asistensi: Price yang ada di bangunan.txt itu harga bikin dan juga harga jual Walikota gabisa menang Walikota bisa menang pajak Bangunan dapat disimpan di tempat penyimpanan Coba sebisa mungkin buat SOLID (S sama L nya yang penting)

Kehadiran Anggota Kelompok:	
No	NIM
Tanda tangan	
1	13522024
	
2	13522038
	
3	13522061
	
4	13522075
	
5	13522101
	
Tanda Tangan Asisten: 	
Hilya F. Imania	

2. Link Pranala : <https://github.com/b33rk/Tubes-1-OOP-Encapsulators>