Facultatea Calculatoare, Informatica si Microelectronica

Universitatea Tehnica a Moldovei

APPOO

Lucrarea de laborator#1

Principiile OOP

Autor:
Alexandru Stamatin

lector asistent:
Mihail Pecari

Lucrarea de laborator #1

1 Scopul lucrarii de laborator

Realizarea unui software care sa contina implementarea celor patru principii OOP

2 Obiective

- Implementarea incapsularii
- Implementarea mostenirii
- Implementarea abstractizarii
- Implementarea polimorfismului

3 Efectuarea lucrarii de laborator

3.1 Sarcinile propuse

- Alegerea limbajului si a tematicii pentru elaborarea aplicatiei
- Realizarea produsului soft care sa implementeze principiile OOP
- Incarcarea codului sursa pe git

3.2 Analiza lucrarii de laborator

In cadrul acestei lucrari de laborator am dezvoltat un joc in consola in limbajul C++ in care utilizatorul are posibilitatea de a-si construi o armata pe baza unitatilor disponibile si a se lupta cu o armata generata automat. Aplicatia implementeaza cele 4 principii OOP

- Incapsularea

  Incapsularea consta in separarea aspectelor externe ale unui obiect care sunt accesibile altor obiecte de implementarea interna care este ascunsa de utilizatorii clasei. In limbajul C++ incapsularea se realizeaza prin plasarea implementarii in portiunea private sau protected a clasei.

```
class Knight: public Unit {

private:

    static int const maxHP = 500;
    static int const blockChance = 3;

    virtual void takeDamage(int dmg) override {
        if (getRand(blockChance)) {this->HP -= 0.5*dmg;}
        else {this->HP -= dmg;}

    }
```

- Abstractizarea

  Abstractizarea este o tehnica de programare care se focuseaza pe interfata unui tip si nu pe detaliile de implementare. Aceasta permite ignorarea detaliilor despre cum un anumit tip este reprezentat si accentuarea pe operatiile care pot fi executate de tipul de date

- Mostenirea

  Mostenirea este un mod de a reutiliza codul existent permitand definirea claselor noi pe baza claselor deja existente. Clasele derivate obtinute mostenesc atributele si comportamentul clasei parinte. In limbajul C++ o clasa derivata trebuie sa specifice clasele de la care aceasta doreste sa mosteneasca prin intermediul listei de derivare. Aceasta reprezinta o lista separata prin virgula a claselor de baza fiecare avand optional un specificator de acces.

```
class Blademaster: public Unit{

private:

    static int const maxHP = 500;
    static int const critChance = 3;
    static int const critMulti = 2;
```

- Polimorfismul

  Polimorfismul se manifesta prin existenta functiilor cu acelasi nume insa functionalitate diferita. Deosebim 2 tipuri de polimorfism: compile-time si run-time. Polimorfismul compile-time se caracterizeaza prin supraincarcarea functiilor si operatorilor, iar functia care va fi apelata la executarea instructiunii se cunoaste in momentul compilarii.

```
    Unit(int HP, int DPS, int Cost):HP(HP),DPS(DPS),Cost(Cost){}
    Unit() = default;
```

  In limbajul C++ polimorfismul run-time se manifesta atunci cand o functie virtuala definita in clasa de baza se apeleaza prin intermediul unei referinte sau pointer la clasa de baza si in timpul compilarii nu se cunoaste tipul obiectului asupra caruia va fi executata operatia. Acesta poate fi obiect al clasei de baza sau clasei derivate. Polimorfismul run-time se implementeaza utilizand cuvintele cheie virtual si override.

```
class Unit{

protected:

    int HP;
    int DPS;
    int Cost;
```

```
public:

    virtual void takeDamage(int dmg) = 0;
    virtual float atack() = 0;
    virtual string getClass() = 0;
    virtual int getMaxHP () = 0 ;

class Knight: public Unit {

private:

    static int const maxHP = 500;
    static int const blockChance = 3;

    virtual void takeDamage(int dmg) override {
        if (getRand(blockChance)) {this->HP -= 0.5*dmg;}
        else {this->HP -= dmg;}

    }

    virtual float atack() override {
        return DPS;
    }

    virtual string getClass() override {
    return "Knight";
    }

    virtual int getMaxHP()override {
    return maxHP;
    }
```

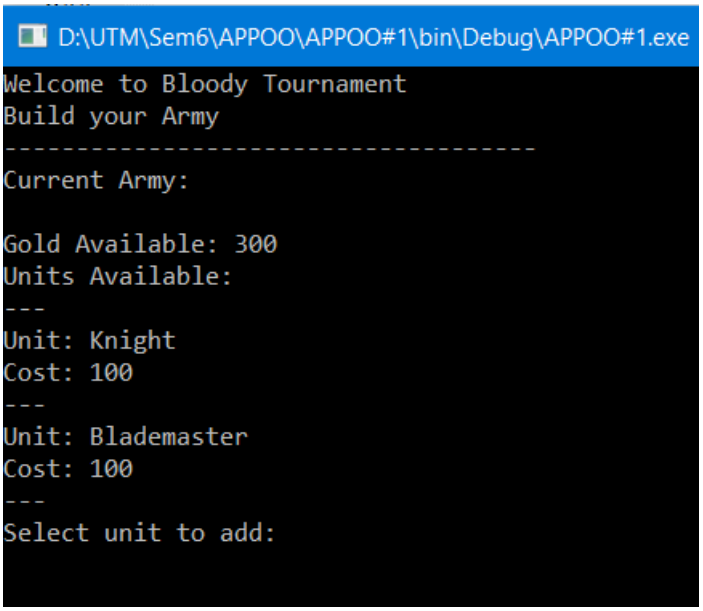

Figure 1: Selectarea unitatilor pentru armata

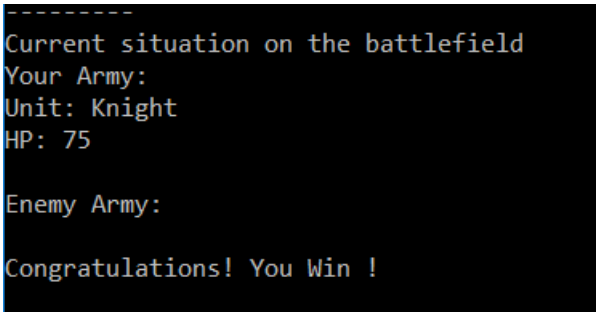

Figure 2: Rezultatul luptei

Concluzie

In cadrul realizarii lucrarii de laborator ne-am familiarizat cu principiile programarii orientate pe obiecte si am reusit dezvoltarea unui program pentru implementarea acestora. Abstractizarea si incapsularea a fost realizata prin intermediul claselor ce definesc atributele si comportamentul de baza a obiectelor si plasarea implementarii in sectiunea protected a clasei. Mostenirea a permis definirea a 2 tipuri de unitati pe baza clasei asbtracte Unit. Clasele derivate au implementat versiunile proprii ale functiilor virtuale din clasa de baza astfel realizandu-se polimorfismul.
Am ajuns la concluzia ca programarea orientata pe obecte este o paradigma de programare ce aduce multe beneficii fata de programarea clasica cum ar fi reutilizarea simpla a codului si facilitarea scrierii programelor complexe de volum mare.

References

[1] Stanley B. Lippman, Jose Lajoie, Barbara E. Moo. *C++ Primer (5th Edition)*.