

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 lucrării de laborator

Realizarea unui software care să conțină implementarea celor patru principii OOP

## 2 Obiective

- Implementarea încapsulării
- Implementarea mostenirii
- Implementarea abstractizării
- Implementarea polimorfismului

## 3 Efectuarea lucrării de laborator

### 3.1 Sarcinile propuse

- Alegerea limbajului și a tematicii pentru elaborarea aplicației
- Realizarea produsului soft care să implementeze principiile OOP
- Încărcarea codului sursă pe git

### 3.2 Analiza lucrării de laborator

În cadrul acestei lucrări de laborator am dezvoltat un joc în consolă în limbajul C++ în care utilizatorul are posibilitatea de a-și construi o armată pe baza unităților disponibile și a se lupta cu o armată generată automat. Aplicația implementează cele 4 principii OOP

- Încapsularea

Încapsularea constă în separarea aspectelor externe ale unui obiect care sunt accesibile altor obiecte de implementarea internă care este ascunsă de utilizatorii clasei. În limbajul C++ încapsularea se realizează prin plasarea implementării în porțiunea `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 tehnică de programare care se focusează pe interfața unui tip și nu pe detaliile de implementare. Aceasta permite ignorarea detaliilor despre cum un anumit tip este reprezentat și accentuarea pe operațiile 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 attributele 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 in sa functionalitate diferita. Deosebim 2 tipuri de polimorfism: compile-time si run-time. Polimorfismul compile-time se caracterizeaza prin supraincercarea 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;  
}
```

```
D:\UTM\Sem6\APPOO\APPOO#1\bin\Debug\APPOO#1.exe
Welcome to Bloody Tournament
Build your Army
-----
Current Army:

Gold Available: 300
Units Available:
---
Unit: Knight
Cost: 100
---
Unit: Blademaster
Cost: 100
---
Select unit to add:
```

Figure 1: Selectarea unitatilor pentru armata

```
-----
Current situation on the battlefield
Your Army:
Unit: Knight
HP: 75

Enemy Army:

Congratulations! You Win !
```

Figure 2: Rezultatul luptei

## Concluzie

În cadrul realizării lucrării de laborator ne-am familiarizat cu principiile programării orientate pe obiecte și am reușit dezvoltarea unui program pentru implementarea acestora. Abstractizarea și încapsularea a fost realizată prin intermediul claselor ce definesc atributele și comportamentul de bază a obiectelor și plasarea implementării în secțiunea `protected` a clasei. Mostenirea a permis definirea a 2 tipuri de unități pe baza clasei abstracte `Unit`. Clasele derivate au implementat versiunile proprii ale funcțiilor virtuale din clasa de bază astfel realizându-se polimorfismul.

Am ajuns la concluzia că programarea orientată pe obiecte este o paradigmă de programare ce aduce multe beneficii față de programarea clasică cum ar fi reutilizarea simplă a codului și facilitarea scrierii programelor complexe de volum mare.

## References

- [1] Marian Cotelea, Vitalie Cotelea. *SQL Server 2014 Pas cu Pas*. Chisinau, 2015