Универзитет „Св. Кирил и Методиј“ - Скопје
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

# Напредно програмирање

Аудиториски вежби 4

Верзија 1.0, 20 Септември, 2016

# Содржина

# 1. Bank (наследување, полиморфизам и интерфејси)

Дадени се следниве пет класи:

1. `Bank`
2. `Account`
3. `NonInterestCheckingAccount`
4. `InterestCheckingAccount`
5. `PlatinumCheckingAccount`

како и интефејс наречен `InterestBearingAccount` кои се однесуваат на следниот начин:

- Во `Bank` чува листа од сите видови сметки, вклучувајќи сметки за штедење и за трошење, некои од нив подложни на камата, а некои не. Во `Bank` постои метод `totalAssets` кој ја враќа сумата на состојбата на сите сметки. Исто така содржи метод `addInterest` кој го повикува методот `addInterest` на сите сметки кои се подложни на камата.
- `Account` е апстрактна класа. Во секој сметка се чуваат името на сопственикот на сметката, бројот на сметката (секвенцијален број доделен автоматски), моменталната состојба. Во класата се имплементираат конструктор за иницијализација на податочните членови, методи за пристап до моменталната состојба, како и за додавање и одземање од моменталната состојба.
- `InterestBearingAccount` интерфејсот деклларира единствен метод `addInterest` (без параметри и не враќа ништо - `void`) кој ја зголемува состојбата со соодветната камата за овој вид на сметка.
- `InterestCheckingAccount` е сметка `Account` која е исто така `InterestBearingAccount`. Повикување `addInterest` ја зголемува состојбата за 3%.
- `PlatinumCheckingAccount` е `InterestCheckingAccount`. Повикување `addInterest` ја зголемува состојбата двојно од каматата за `InterestCheckingAccount` (колку и да е таа).

- `NonInterestCheckingAccount` е сметка `Account` но не е `InterestBearingAccount`. Нема дополнителни функционалности надвор од основните од класата `Account`.

За оваа задача, потребно е да се имплментира функционалност дадена во претходниот текст:

1. Пет од шест класи од споменатите формираат хиерархија. За овие класи да се нацрта оваа хиерархија.
2. Да се имплементира `Account`.
3. Да се имплементира `NonInterestCheckingAccount`.
4. Да се напише `InterestBearingAccount` интерфејсот.
5. Да се имплементира `Bank`.
6. Да се имплементира `InterestCheckingAccount`.
7. Да се имплементира `PlatinumCheckingAccount`.

*Решение (`Account.java`)*

```
package mk.ukim.finki.np.av4.bank;

public abstract class Account {

    private String holderName;
    private int number;
    private double currentAmount;

    public Account(String holderName, int number, double currentAmount) {
        this.holderName = holderName;
        this.number = number;
        this.currentAmount = currentAmount;
    }

    public double getCurrentAmount() {
        return currentAmount;
    }

    public void addAmount(double amount) {
        currentAmount += amount;
    }

    public void withdrawAmount(double amount) {
        currentAmount -= amount;
    }

}
```

*Решение (*NonInterestCheckingAccount.java*)*

```
package mk.ukim.finki.np.av4.bank;

public class NonInterestCheckingAccount extends Account {

    public NonInterestCheckingAccount(String holderName, int number, double currentAmount)
{
        super(holderName, number, currentAmount);
    }
}
```

*Решение (*InterestBearingAccount.java*)*

```
package mk.ukim.finki.np.av4.bank;

public interface InterestBearingAccount {
    void addInterest();
}
```

*Решение (*Bank.java*)*

```
package mk.ukim.finki.np.av4.bank;

import java.util.Arrays;

public class Bank {
    private Account[] accounts;
    private int totalAccounts;
    private int max;

    public Bank(int max) {
        this.totalAccounts = 0;
        this.max = max;
        accounts = new Account[max];
    }

    public void addAccount(Account account) {
        if (totalAccounts == accounts.length) {
            accounts = Arrays.copyOf(accounts, max * 2);
        }
        accounts[totalAccounts++] = account;
    }

    public double totalAssets() {
        double sum = 0;
        for (Account account : accounts) {
            sum += account.getCurrentAmount();
        }
        return sum;
    }

    public void addInterest() {
        for (Account account : accounts) {
            if (account instanceof InterestBearingAccount) {
                InterestBearingAccount iba = (InterestBearingAccount) account;
                iba.addInterest();
            }
        }
    }
}
```

*Решение (*`InterestCheckingAccount.java`*)*

```
package mk.ukim.finki.np.av4.bank;

public class InterestCheckingAccount
        extends Account implements InterestBearingAccount {

    public static final double INTEREST_RATE = .03; // 3%

    public InterestCheckingAccount(String holderName, int number, double currentAmount) {
        super(holderName, number, currentAmount);
    }

    @Override
    public void addInterest() {
        addAmount(getCurrentAmount() * INTEREST_RATE);
    }

}
```

*Решение (*`PlatinumCheckingAccount.java`*)*

```
package mk.ukim.finki.np.av4.bank;

public class PlatinumCheckingAccount extends Account implements
        InterestBearingAccount {

    public PlatinumCheckingAccount(String holderName, int number, double currentAmount) {
        super(holderName, number, currentAmount);
    }

    @Override
    public void addInterest() {
        addAmount(getCurrentAmount() * InterestCheckingAccount.INTEREST_RATE * 2);
    }

}
```

# 2. Наследување (рефакторирање)

Следниот код е дизајниран од J. Hacker за видео игра. Постои класа `Alien` која репрезентира вонземјанин и класа `AlienPack` која репрезентира група вонземјани и колку штета може да нанесат:

`Alian.java`

```
package mk.ukim.finki.np.av4;

public class Alien {
    public static final int SNAKE_ALIEN = 0;
    public static final int OGRE_ALIEN = 1;
    public static final int MARSHMALLOW_MAN_ALIEN = 2;
    public int type; // Stores one of the three above types
    public int health; // 0=dead, 100=full strength
    public String name;

    public Alien(int type, int health, String name) {
        this.type = type;
        this.health = health;
        this.name = name;
    }
}
```

AlianPack.java

```
package mk.ukim.finki.np.av4;

public class AlienPack {
    private Alien[] aliens;

    public AlienPack(int numAliens) {
        aliens = new Alien[numAliens];
    }

    public void addAlien(Alien newAlien, int index) {
        aliens[index] = newAlien;
    }

    public Alien[] getAliens() {
        return aliens;
    }

    public int calculateDamage() {
        int damage = 0;
        for (int i = 0; i < aliens.length; i++) {
            if (aliens[i].type == Alien.SNAKE_ALIEN) {
                damage += 10;// Snake does 10 damage
            } else if (aliens[i].type == Alien.OGRE_ALIEN) {
                damage += 6;// Ogre does 6 damage
            } else if (aliens[i].type == Alien.MARSHMALLOW_MAN_ALIEN) {
                damage += 1;
                // Marshmallow Man does 1 damage
            }
        }
        return damage;
    }
}
```

Кодот не е многу објектно ориентиран и не подржува криење на информациите во класата `Alien`. Да се пренапише, така што ќе се искористи наследување за да се репрезентираат различни типови вонземјани, наместо да се користи параметарот `type`. Исто така пренапишете ја класата `Alien` така што ќе ги крие инстанцните променливи и додадете метод `getDamage` кој за секоја од изведените класа ќе ја враќа штетата која ја предизвикува. На крај пренапишете го методот `calculateDamage` да го користи `getDamage` и напишете `main` метод да ја тестирате класата.

# 3. Читање од стандарден влез (SI)

## 3.1. Калкулатор

Да се напише програма едноставен калкулатор. Калкулаторот чува еден број од тип `double` со име резултат и неговата почетна вредност е `0.0`. Во циклус му се дозволува на корисникот да додаде, одземе, помножи или подели со втор број. Резултатот од овие операции е новата вредност на резултатот. Пресметката завршува кога корисникот ќе внесе R за `result` (како мала или голема буква).

Корисникот може да направи уште една пресметка од почеток или да ја заврши програмата `(Y/N)`. Ако корисникот внесе различен знак за оператор од `+`, `-`, `*` или `/`, тогаш се фрла исклучок `UnknownOperatorException` и се чека повторно на внес.

*Пример форматот на влезните податоци:*

```
Calculator is on.
result = 0.0
+5
result + 5.0 = 5.0
new result = 5.0
* 2.2
result * 2.2 = 11.0
updated result = 11.0
% 10
% is an unknown operation.
Reenter, your last line:
* 0.1
result * 0.1 = 1.1
updated result = 1.1
r
Final result = 1.1
Again? (y/n)
yes
result = 0.0
+10
result + 10.0 = 10.0
new result = 10.0
/2
result / 2.0 = 5.0
updated result = 5.0
r
Final result = 5.0
Again? (y/n)
N
End of Program
```

*Решение*

```
package mk.ukim.finki.np.av4;

public class Calculator {
    private double result;
    private static final char PLUS = '+';
    private static final char MINUS = '-';
    private static final char MULTIPLY = '*';
    private static final char DIVIDE = '/';

    public Calculator() {
        result = 0;
    }

    public String init() {
        return String.format("result = %f", result);
    }

    public double getResult() {
        return result;
    }

    public String execute(char operator, double value)
            throws UnknownOperatorException {

        if (operator == PLUS) {
            result += value;
        } else if (operator == MINUS) {
            result -= value;
        } else if (operator == MULTIPLY) {
            result *= value;
        } else if (operator == DIVIDE) {
            result /= value;
        } else {
            throw new UnknownOperatorException(operator);
        }
        return String.format("result %c %f = %f", operator, value, result);
    }

    class UnknownOperatorException extends Exception {
        public UnknownOperatorException(char operator) {
            super(String.format("%c is unknown operation", operator));
        }
    }

    @Override
    public String toString() {
        return String.format("updated result = %f", result);
    }

}
```

*Решение (Тест програма)*

```
package mk.ukim.finki.np.av4;

import java.util.Scanner;


public class CalculatorTest {
    static final char RESULT = 'r';

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            Calculator calculator = new Calculator();
            System.out.println(calculator.init());
            while (true) {
                String line = scanner.nextLine();
                char choice = getCharLower(line);
                if (choice == RESULT) {
                    System.out.println(String.format("final result = %f", calculator
.getResult()));
                    break;
                }
                String[] parts = line.split("\\s+");
                char operator = parts[0].charAt(0);
                double value = Double.parseDouble(parts[1]);
                try {
                    String result = calculator.execute(operator, value);
                    System.out.println(result);
                    System.out.println(calculator);
                } catch (Calculator.UnknownOperatorException e) {
                    System.out.println(e.getMessage());
                }
            }
            System.out.println("(Y/N)");
            String line = scanner.nextLine();
            char choice = getCharLower(line);
            if (choice == 'n') {
                break;
            }
        }
    }

    static char getCharLower(String line) {
        if (line.trim().length() > 0) {
            return Character.toLowerCase(line.charAt(0));
        }
        return '?';
    }
}
```

# 4. Изворен код од примери и задачи

https://github.com/finki-mk/NP/

Source Code ZIP