Fakulta informatiky a informačných technológií
Slovenskej technickej univerzity v Bratislave

Základy objektovo-orientovaného programovania

Organizačný softvér tenisových turnajov

Použité princípy objektovo-orientovaného programovania

Emma Macháčová

Zimný semester 2020/2021

Obsah

Funkcionalita

Funkcionalita pozostáva z:

- načítania existujúceho turnaju, alebo vytvorenia nového záznamu

```
Tournament Organizer v.001.05
Zelate si turnaj nacitat (n) alebo vytvorit (v)?
```

- menu

```
--- MANUAL
--- pre vypis pavuka (1)
--- pre vypis zapasov na kurte (2)
--- pre vypis rund (3)
--- pre vypis zisku (4)
--- pre ulozenie (5)
--- pre ukoncenie (0)
```

A) Vytvorenie turnaja

- výber kategórie (A/B/C/D)
- zadanie počtu kurtov ktoré má usporiadateľ k dispozícii
- zadanie obdobia (leto 1 / zima 0)
- zadanie veľkosti pavúka (128, 64, 32, 16, 8)
- udanie času a dátumu zahájenia
- mená hráčov
- výber dvojhry / štvorhry (primárne je zatiaľ implementovaná dvojhra)

```
Tournament Organizer v.001.05
Zelate si turnaj nacitat (n) alebo vytvorit (v)?
v
Zadaj kategoriu turnaja:
A
Zadaj pocet kurtov:
20
Ak leto zadaj 1 ak zima 0:
1
Zadaj velkost pavuka:
64
Zadaj cislo dna v mesiaci:
16
Zadaj hodinu zaciatku turnaja:
8
Zadaj minuty:
30
Zadaj meno hraca (no.1.):
Arnold
Zadaj meno hraca (no.2.):
```

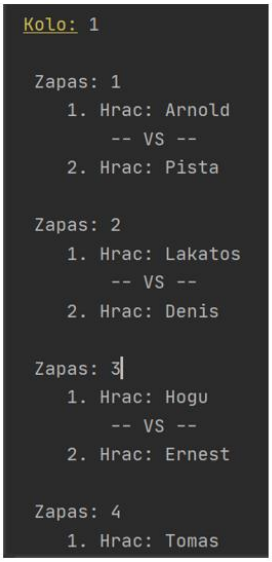
B) Načítanie záznamu

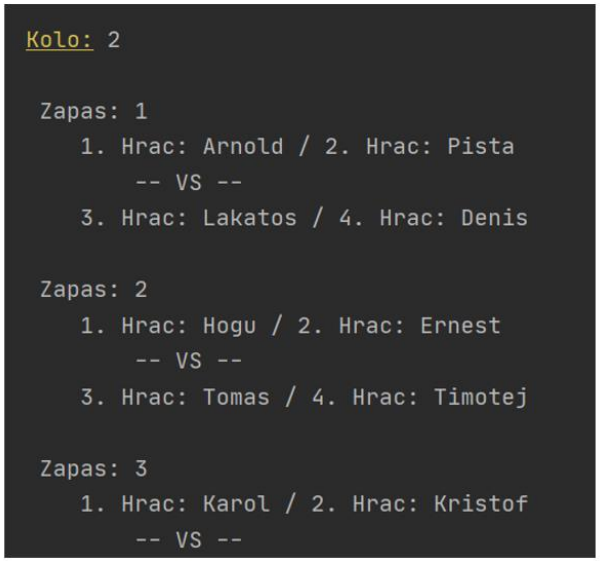
```
Tournament Organizer v.001.05
Zelate si turnaj nacitat (n) alebo vytvorit (v)?
n
Zadaj meno ulozeneho suboru:
ukazka

--- MANUAL
--- pre vypis pavuka (1)
--- pre vypis zapasov na kurte (2)
```

- pre prácu s už vytvoreným záznamom turnaja

C) Menu

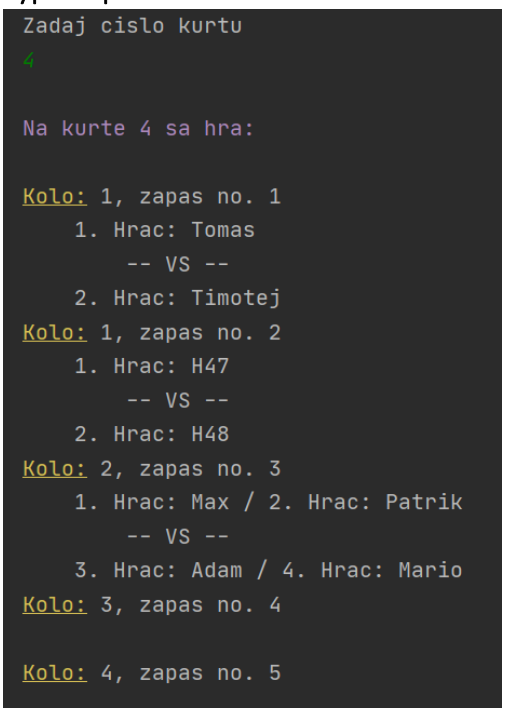
(1) výpis pavúka





- vypíše zápasy turnaja (prvé dve kolá s menami hráčov, ostatné podľa čísla zápasu)

(2) výpis zápasov na kurte



- vypíše zoznam zápasov, ktoré sa budú v rámci turnaja hrať na danom kurte

(3) výpis rúnd

```
Runda: 1, datum & cas rundy: 16. 11. 8:30
Na kurte: 1, Kolo: 1, zapas: 1
    1. Hrac: Arnold
        -- VS --
    2. Hrac: Pista
Na kurte: 2, Kolo: 1, zapas: 2
    1. Hrac: Lakatos
        -- VS --
    2. Hrac: Denis
Na kurte: 3, Kolo: 1, zapas: 3
    1. Hrac: Hogu
        -- VS --
    2. Hrac: Ernest
```

```
Runda: 3, datum & cas rundy: 16. 11. 12:30
Na kurte: 1, Kolo: 2, zapas: 1
    1. Hrac: Arnold / 2. Hrac: Pista
        -- VS --
    3. Hrac: Lakatos / 4. Hrac: Denis
Na kurte: 2, Kolo: 2, zapas: 2
    1. Hrac: Hogu / 2. Hrac: Ernest
        -- VS --
    3. Hrac: Tomas / 4. Hrac: Timotej
Na kurte: 3, Kolo: 2, zapas: 3
    1. Hrac: Karol / 2. Hrac: Kristof
        -- VS --
    3. Hrac: Matej / 4. Hrac: Martin
```

- runda predstavuje zápasy, ktoré sa hrajú o tom istom čase na kurtoch, ktoré sú k dispozícií
- rôzne kolá (semifinále / štvrťfinále..) sa nehrajú v tej istej runde
- prvý zápas nového kola je pridelený na centrálny kurt (kurt 1)
- na jeden deň sa hrá toľko zápasov (z pohľadu hráča) koľko je špecifikované v kategórií turnaja

(4) výpis zisku

```
4
Zisk je 1088 EUR slovenskych
```

- zisk závisí od obdobia, v ktorom sa turnaj uskutočňuje, kategórie, a počtu hráčov
- nie sú odpočítané prevádzkové náklady

(5) uloženie záznamu

```
5
Zadaj meno suboru pre ulozenie:
ukazka
```

- pre opätovné použitie

(6) ukončenie programu

```
0
Chystate sa vypnut program. Udaje mozno neboli ulozene praje ulozit? (y/n)
n
Neulozene, dovi
```

- program upozorní na možnosť straty údajov

Dedenie

Dedenie je forma vytvárania tried pričom trieda kt. dedí už v sebe obsahuje všetky metódy a atribúty triedy z ktorej dedí.

class Runda

```
public class Runda extends Kolo {
}
```

class Turnaj

```
public abstract class Turnaj extends Turnaje implements Serializable {
}
```

class TurnajB

```
public class TurnajB extends Turnaj {
}
```

class TurnajC

```
public class TurnajC extends Turnaj {
}
```

class TurnajD

```
public class TurnajD extends Turnaj {
}
```

Viacnásobné dedenie

class TurnajA

```
public class TurnajA extends Turnaj {
}
```

class MajstrovstvaSR

```
public class MajstrovstvaSR extends TurnajA {
}
```

Modifikátory prístupu

Modifikátory prístupu slúžia na obmedzenie prístupu k atribútom triedy.

abstract class Turnaj

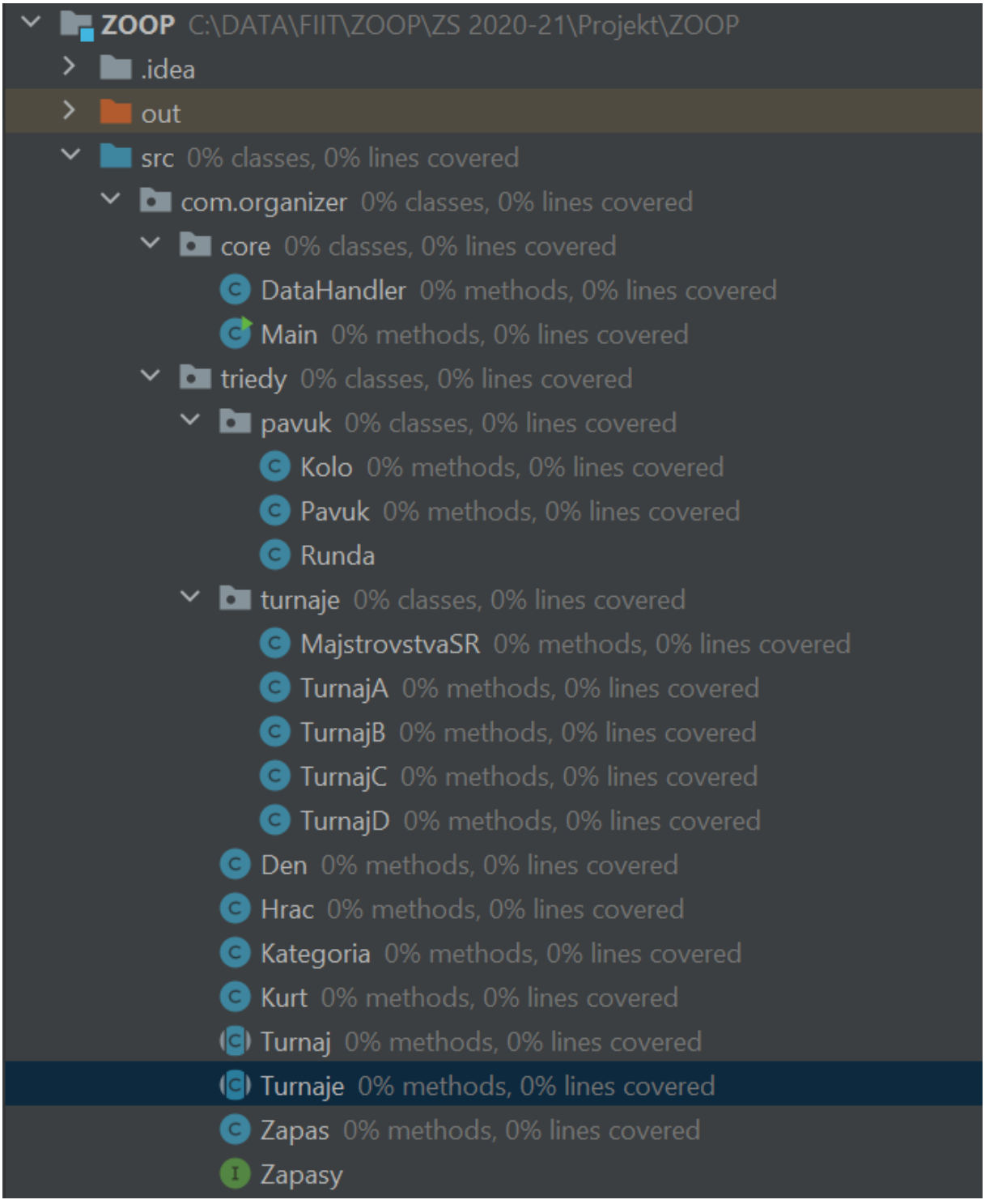
```
public abstract class Turnaj extends Turnaje implements Serializable {
    protected static int velkostPavuka;
    protected Kategoria kategoria;
    protected int pocetKurtov;
    protected int pocetDni;
    protected Date zaciatokTurnaja;
    protected ArrayList<Hrac> hraci;
    protected ArrayList<Zapas> zapasy;
    protected ArrayList<Den> dni;
    protected ArrayList<Kurt> zoznamKurtov;
    protected Pavuk pavuk;
}
```

```
public void setZaciatokTurnaja(Date zaciatokTurnaja) {
    this.zaciatokTurnaja = zaciatokTurnaja;
}

public int getPocetDni() {
    return pocetDni;
}

public void setPocetDni(int pocetDni) {
    this.pocetDni = pocetDni;
}

public int getPocetKurtov() {
    return pocetKurtov;
}

public void setPocetKurtov(int pocetKurtov) {
    this.pocetKurtov = pocetKurtov;
}

public int getVelkostPavuka() {
    return velkostPavuka;
}

public void setVelkostPavuka(int velkostPavuka) {
    Turnaj.velkostPavuka = velkostPavuka;
}
```

class Kolo

```
public class Kolo implements Serializable {
    protected ArrayList<Zapas> zapasy;
    protected int poradie;
}
```

Balíky


ZOOP C:\DATA\FIIT\ZOOP\ZS 2020-21\Projekt\ZOOP
.idea
out
src 0% classes, 0% lines covered
com.organizer 0% classes, 0% lines covered
core 0% classes, 0% lines covered
DataHandler 0% methods, 0% lines covered
Main 0% methods, 0% lines covered
triedy 0% classes, 0% lines covered
pavuk 0% classes, 0% lines covered
Kolo 0% methods, 0% lines covered
Pavuk 0% methods, 0% lines covered
Runda
turnaje 0% classes, 0% lines covered
MajstrovstvaSR 0% methods, 0% lines covered
TurnajA 0% methods, 0% lines covered
TurnajB 0% methods, 0% lines covered
TurnajC 0% methods, 0% lines covered
TurnajD 0% methods, 0% lines covered
Den 0% methods, 0% lines covered
Hrac 0% methods, 0% lines covered
Kategoria 0% methods, 0% lines covered
Kurt 0% methods, 0% lines covered
Turnaj 0% methods, 0% lines covered
Turnaje 0% methods, 0% lines covered
Zapas 0% methods, 0% lines covered
Zapasy

Preťažovanie metód

Preťažovanie metód je spôsob akým vieme vytvoriť metódu s rovnakým názvom ale inou implementáciou.

abstract class Turnaj

```
public static Turnaj vytvorTurnaj() {

        .....

    return turnaj;
}
```

```
public static Turnaj vytvorTurnaj(Turnaj truenaj) throws
CloneNotSupportedException {

    Turnaj turnaj = truenaj.clone();

    return turnaj;

}
```

Prekonávanie metód

Prekonávanie je spôsob akým vieme implementovať metódu ktorá už bola implementovaná v rodičovskej triede.

class MajstrovstvaSR

```
public static Turnaj vytvorTurnaj(char typ) {
    Date datum = new Date();
    datum.setMonth(3);
    datum.setDate(12);
    datum.setHours(8);
    datum.setMinutes(0);
    datum.setSeconds(0);
    ArrayList<Hrac> hrac = Hrac.nacitajHracov(velkostPavuka);
    Pavuk pavuk;
    MajstrovstvaSR MS = new MajstrovstvaSR(3, true, velkostPavuka, datum,
hrac);
    pavuk = new Pavuk(velkostPavuka, MS);
    MS.setPavuk(pavuk);
    return MS;
}
```

abstract class Turnaj

```
public static Turnaj vytvorTurnaj() {

        .....

    return turnaj;
}
```

Agregácia

Napr. class Turnaj má hráčov (class Hrac) a s hráčmi môže byt manipulované len cez objekt Turnaj.

Kompozícia

Napr. class Turnaj má pavúk (class Pavuk), a ked zanikne Turnaj zanikne aj Pavuk.

Asociácia

Napr. class Turnaj má kategóriu (class Kategoria) ale viacero turnajov môže mať tú istú Kategóriu pričom zmena v turnaji ovplyvní kategóriu ale zmena v kategórii neovplyvní turnaj.

Alebo medzi hráčmi a kolami – hráč môže existovať vo viacerých kolách.

Final atribút

Final atribút je určovanie nejakej konštanty ktorá sa počas behu programu nemení, len sa raz zadá.

class Kategoria

```
public class Kategoria implements Serializable {
    private char kategoria; // A B C D
    private boolean leto; // leto / zima
    private int maxDvojhry;
    private int maxStvorhry;
    private int maxZapasov; // limit zapasov
    private int vklad; // poplatok hraca

    private final int odhadTrvania = 2;
}
```

Final metóda

Final metóda je metóda kt. počas svojho behu nemení hodnotu žiadnej premennej.

Main

```
public static void main(String[] args) {
    uvodnyText();

    .....
}
```

```
public static final void uvodnyText(){
    System.out.println("\u001B[35m" + "Tournament Organizer v.001.05" +
"\u001B[0m");
}
```

Abstraktná trieda

Abstraktná trieda je trieda ktorá môže obsahovať abstraktné metódy a nemôže byť vytvorená jej inštancia.

abstract class Turnaje

```
abstract public class Turnaje {
}
```

abstract class Turnaj

```
public abstract class Turnaj extends Turnaje implements Serializable {
}
```

Abstraktná metóda

Je to predpis metódy bez implementácie funkcionality danej metódy.

abstract class Turnaje

```
abstract public Pavuk getPavuk();
```

Statická metóda

Je to metóda ktorá sa dá zavolať aj bez inštancie triedy v ktorej je implementovaná.

abstract class Turnaj

```
public static Turnaj vytvorTurnaj() {

        .....

    return turnaj;
}
```

```
public static Turnaj vytvorTurnaj(Turnaj truenaj) throws
CloneNotSupportedException {

    Turnaj turnaj = truenaj.clone();

    return turnaj;

}
```

class Hrac

```
public static ArrayList<Hrac> nacitajHracov(int pocetHracov) {

    Scanner keyboard = new Scanner(System.in);
    ArrayList<Hrac> hraci = new ArrayList<Hrac>(0);
    for (int i = 0; i < pocetHracov; ++i) {
        System.out.println("Zadaj meno hraca (no." + (i + 1) + ".):");
        String meno = keyboard.nextLine();
        hraci.add(new Hrac(meno));
    }

    return hraci;
}
```

class Pavuk

```
public static int zistiPocetKol() { // zisti kolko kol sa bude hrat
    int velkost = velkostPavuka;
    pocetKol = 0;
    while (velkost % 2 == 0) {
        velkost /= 2;
        pocetKol++;
    }

    return pocetKol;
}
```

```
public static void vypisPavuk(Pavuk pavuk) {
    .....
}
```

```
public static void vypisRundy(Pavuk pavuk) {
    .....
}
```

```
public static int getVelkostPavuka() { return velkostPavuka;}

public static void setVelkostPavuka(int velkostPavuka) {
    Pavuk.velkostPavuka = velkostPavuka;
}

public static int getPocetKol() { return pocetKol;}

public static void setPocetKol(int pocetKol) { Pavuk.pocetKol = pocetKol; }
```

class Kolo

```
public static ArrayList<Zapas> vytvorKolo(ArrayList<Hrac> hraci, boolean
dvojhra, ArrayList<Kurt> kurty, Kolo kolo) {
    .....
    return odhady;
}
```

```
public static ArrayList<Zapas> doplnKoloPavuka(int pocetZvysnychKol, Kolo
kolo) {
    .....
    return doplnujuce;
}
```

Main

```
public static void main(String[] args) {
    .....
}
```

Statický atribút

class Turnaj

```
public abstract class Turnaj extends Turnaje implements Serializable {

    protected static int velkostPavuka;

    protected Kategoria kategoria;
    protected int pocetKurtov;
    protected int pocetDni;
    protected Date zaciatokTurnaja;
    protected ArrayList<Hrac> hraci;
    protected ArrayList<Zapas> zapasy;
    protected ArrayList<Den> dni;
    protected ArrayList<Kurt> zoznamKurtov;
    protected Pavuk pavuk;
}
```

Rozhranie

Rozhranie môže obsahovať iba abstraktné metódy.

package triedy

```
public interface Zapasy {

    ArrayList<Hrac> getHraci();


    .....

}
```

Upcasting

class Turnaj

Do premennej typu Turnaj (parrent) sa ukladajú inštancie objektov typu TurnajA, B, C, D (child).

```
public static Turnaj vytvorTurnaj() {

    .....

    Turnaj turnaj;
    switch (Character.toUpperCase(typ)) {
        case 'A':
            turnaj = new TurnajA(3, leto, velkostPavuka, datum, hrac);
            break;
        case 'B':
            turnaj =  TurnajB.vytvorObjekt(3,leto,velkostPavuka,datum,
hrac);
            break;
        case 'C':
            turnaj = new TurnajC(3, leto, velkostPavuka, datum, hrac);
            break;
        case 'D':
            turnaj = new TurnajD(3, leto, velkostPavuka, datum, hrac);
            break;
        default:
            System.out.println("Neplatny vstup");
            return null;
    }
}
```

Privátny konštruktor

Privátny konštruktor sa používa keď chceme aby trieda vedela vytvoriť iba jednu inštanciu. Takáto trieda sa vola singleton.

class TurnajA

```
public class TurnajB extends Turnaj {
    private static TurnajB turnaj;

    private TurnajB(int pocetDni, boolean leto, int velkostPavuka, Date
zaciatokTurnaja, ArrayList<Hrac> hraci){
        super();
        this.kategoria = new Kategoria('B', leto); // instanciuje kategoriu
        this.dni = new ArrayList<Den>(pocetDni);
        Turnaj.velkostPavuka = velkostPavuka;
        this.zaciatokTurnaja = zaciatokTurnaja;
        this.hraci = hraci;
    };

    public static Turnaj vytvorObjekt(int pocetDni, boolean leto, int
velkostPavuka, Date zaciatokTurnaja, ArrayList<Hrac> hraci){
        if(turnaj == null){
            turnaj = new
TurnajA(pocetDni,leto,velkostPavuka,zaciatokTurnaja,hraci);
        }
        return turnaj;
    }

}
```

UML