**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 4](#_Toc58064574)

[A) Vytvorenie turnaja 5](#_Toc58064575)

[B) Načítanie záznamu 5](#_Toc58064576)

[C) Menu 6](#_Toc58064577)

[(1) výpis pavúka 6](#_Toc58064578)

[(2) výpis zápasov na kurte 6](#_Toc58064579)

[(3) výpis rúnd 7](#_Toc58064580)

[(4) výpis zisku 7](#_Toc58064581)

[(5) uloženie záznamu 7](#_Toc58064582)

[(6) ukončenie programu 7](#_Toc58064583)

[Dedenie 8](#_Toc58064584)

[class Runda 8](#_Toc58064585)

[class Turnaj 8](#_Toc58064586)

[class TurnajB 8](#_Toc58064587)

[class TurnajC 8](#_Toc58064588)

[class TurnajD 8](#_Toc58064589)

[Viacnásobné dedenie 8](#_Toc58064590)

[class TurnajA 8](#_Toc58064591)

[class MajstrovstvaSR 8](#_Toc58064592)

[Modifikátory prístupu 9](#_Toc58064593)

[abstract class Turnaj 9](#_Toc58064594)

[class Kolo 9](#_Toc58064595)

[Balíky 10](#_Toc58064596)

[Preťažovanie metód 11](#_Toc58064597)

[abstract class Turnaj 11](#_Toc58064598)

[Prekonávanie metód 12](#_Toc58064599)

[class MajstrovstvaSR 12](#_Toc58064600)

[abstract class Turnaj 12](#_Toc58064601)

[Agregácia 13](#_Toc58064602)

[Kompozícia 13](#_Toc58064603)

[Asociácia 13](#_Toc58064604)

[Final atribút 14](#_Toc58064605)

[class Kategoria 14](#_Toc58064606)

[Final metóda 14](#_Toc58064607)

[Main 14](#_Toc58064608)

[Abstraktná trieda 15](#_Toc58064609)

[abstract class Turnaje 15](#_Toc58064610)

[abstract class Turnaj 15](#_Toc58064611)

[Abstraktná metóda 15](#_Toc58064612)

[abstract class Turnaje 15](#_Toc58064613)

[Statická metóda 16](#_Toc58064614)

[abstract class Turnaj 16](#_Toc58064615)

[class Hrac 16](#_Toc58064616)

[class Pavuk 16](#_Toc58064617)

[class Kolo 17](#_Toc58064618)

[Main 17](#_Toc58064619)

[Statický atribút 18](#_Toc58064620)

[class Turnaj 18](#_Toc58064621)

[Rozhranie 19](#_Toc58064622)

[package triedy 19](#_Toc58064623)

[Upcasting 20](#_Toc58064624)

[class Turnaj 20](#_Toc58064625)

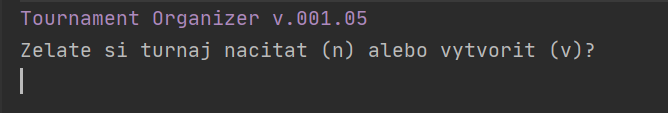
[Privátny konštruktor 21](#_Toc58064626)

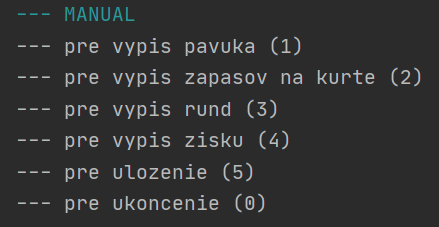
[class TurnajA 21](#_Toc58064627)

[UML 22](#_Toc58064628)

Funkcionalita

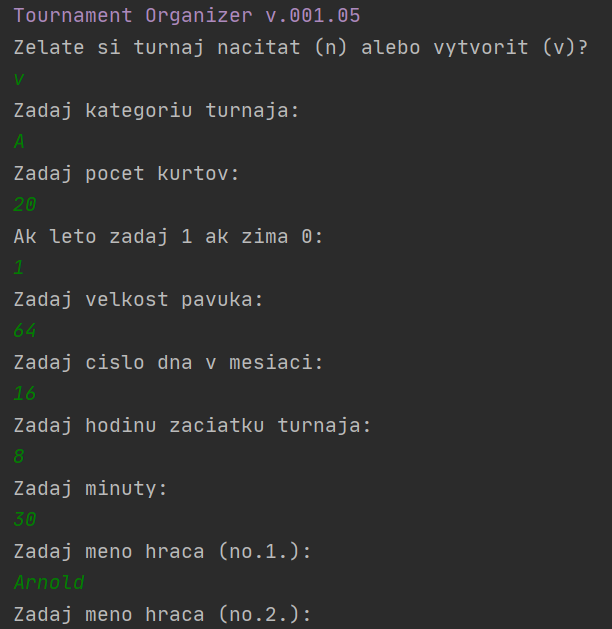
Funkcionalita pozostáva z:

* načítania existujúceho turnaju, alebo vytvorenia nového záznamu
* menu

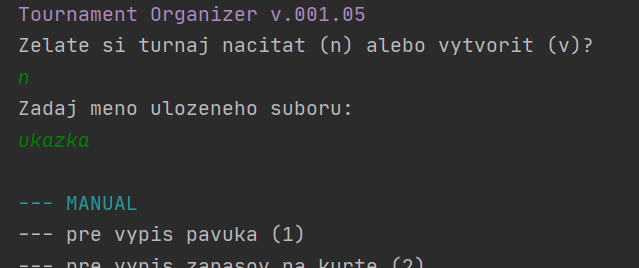


A) Vytvorenie turnaja

* + výber kategórie (A/B/C/D)
  + zadanie počtu kurtov ktoré má usporiadateľ k dispozícií
  + 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)



B) Načítanie záznamu



* + pre prácu s už vytvoreným záznamom turnaja

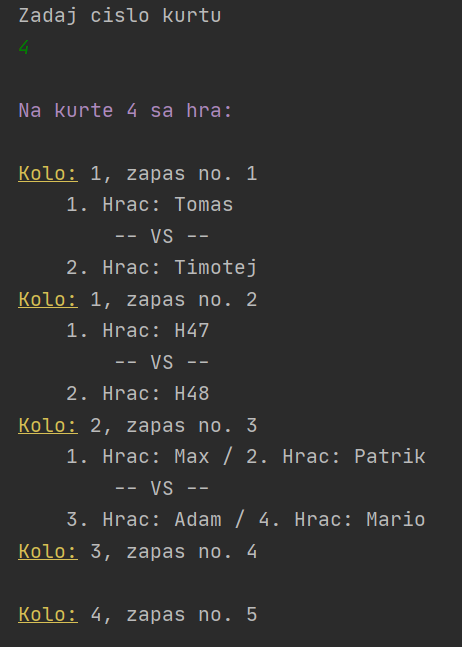
C) Menu

1. **výpis pavúka**

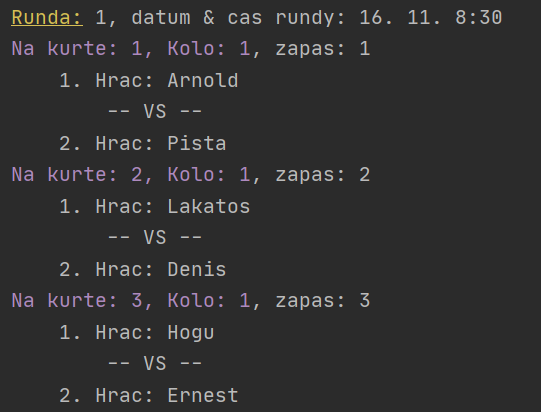
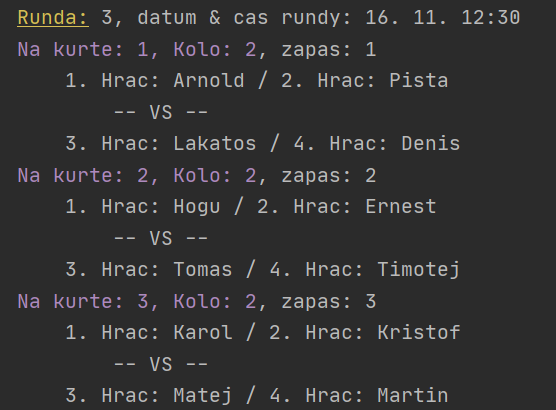


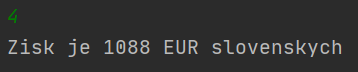
* + vypíše zápasy turnaja (prvé dve kolá s menami hráčov, ostatné podľa čísla zápasu)

1. **výpis zápasov na kurte**



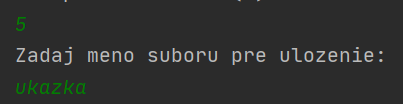
* + vypíše zoznam zápasov, ktoré sa budú v rámci turnaja hrať na danom kurte

1. **výpis rúnd**
   * 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
2. **výpis zisku**



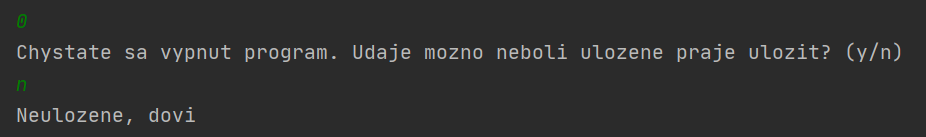
* + zisk závisí od obdobia, v ktorom sa turnaj uskutočňuje, kategórie, a počtu hráčov
  + nie sú odpočítané prevádzkové náklady

1. **uloženie záznamu**



* + pre opätovné použitie

1. **ukončenie programu**



* + 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;

}

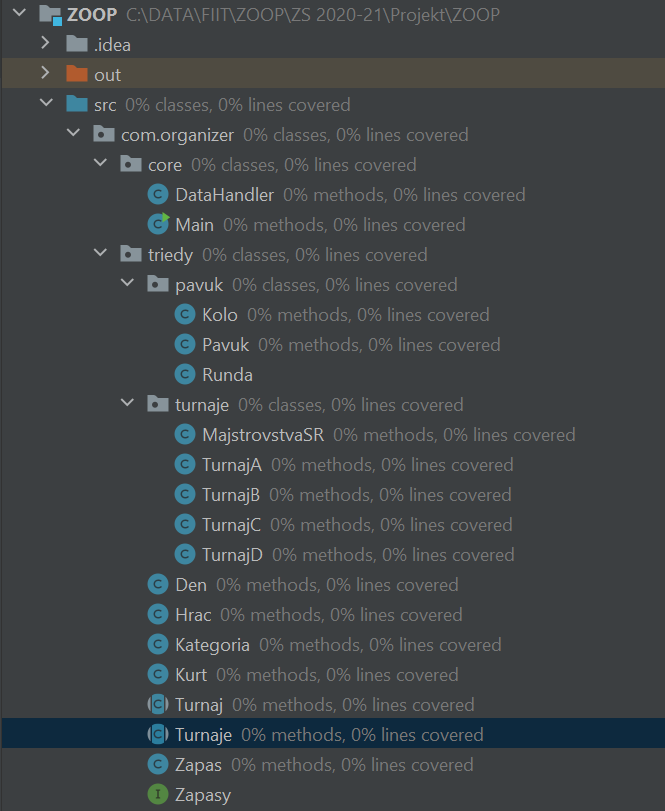
public void setZaciatokTurnaja(Date zaciatokTurnaja) {  
 this.zaciatokTurnaja = zaciatokTurnaja;  
}  
  
public int getPocetDni() {  
 return pocetDni;  
}  
  
public void setPocetDni(int pocetDni) {  
 this.pocetDni = pocetDni;  
}  
  
public int getPocetKurtov() {  
 return pocetKurtov;  
}  
  
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



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