Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

MYS – Manage your season

Objektovo-orientované programovanie

Adam Jurčišin

Dátum : 14. 5. 2023

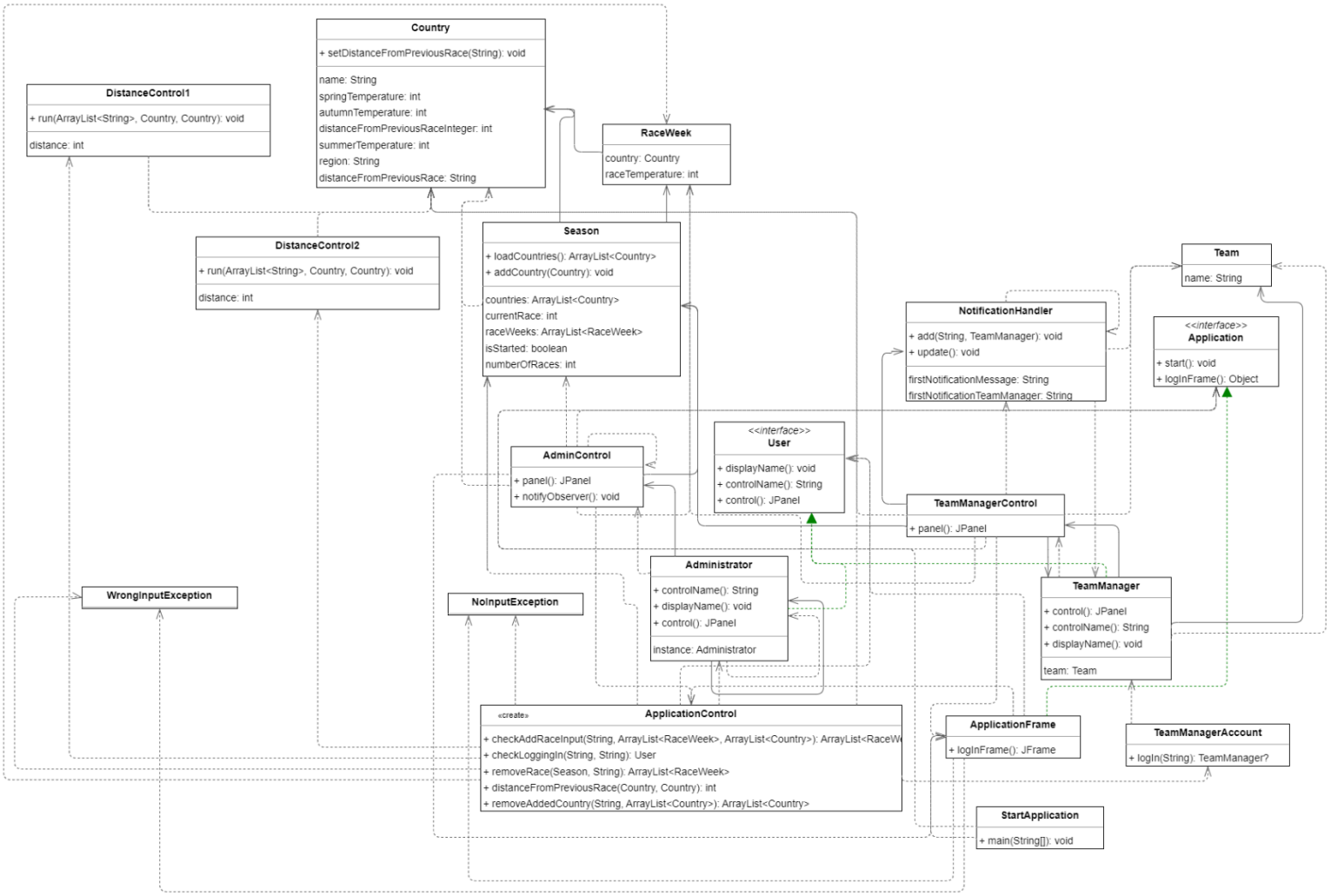
Zámer

Sezóna v motoristickom športe dokáže byť dlhá a finančne náročná. Celé tímy so svojimi jazdcami absolvujú mnoho kilometrov v lietadle ročne aby sa zúčastnili na jednotlivých pretekoch. V tomto projekte sa pozrieme na plánovanie sezóny pre Formulu 1.

Sezóna Formuly 1 začína v marci a pozostáva z 24 pretekových víkendov, počas ktorých sa v piatok odohrávajú 2 tréningy, v sobotu 1 tréning a následne kvalifikácia, ktorá rozhodne o štartovnom poradí v nedeľu na hlavné preteky. Z dôvodu teplotných rozdielov v rôznych krajinách sa preteky v teplejších krajinách konajú v chladnejšom období. V prípade, že je vzdialenosť medzi krajinami príliš veľká, koná sa týždňová alebo dvojtýždňová prestávka. V projekte vytvoríme plánovanie takejto sezóny, kde administrátor pred začiatkom sezóny vyberie v ktorých krajinách sa preteky budú konať, následne sa vytvorí zoznam pretekov s dátumom konania. Následne začne sezóna, v prípade, že sa niektorí z tímov alebo jazdcov nebude môcť zúčastniť pretekov, bude musieť do systému zadať neúčasť na pretekoch. V prípade problémov s konaním pretekov bude vedieť administrátor zrušiť ich konanie a presunúť na iný dátum.

Vytvorením projektu vznikne funkčná aplikácia na riadenie jednej celej sezóny, ktorá bude použiteľná aj v nasledujúcich ročníkoch.

UML Diagram


Country
+ setDistanceFromPreviousRace(String): void
name: String
springTemperature: int
autumnTemperature: int
distanceFromPreviousRaceInteger: int
summerTemperature: int
region: String
distanceFromPreviousRace: String
DistanceControl1
+ run(ArrayList<String>, Country, Country): void
distance: int
RaceWeek
country: Country
raceTemperature: int
DistanceControl2
+ run(ArrayList<String>, Country, Country): void
distance: int
Season
+ loadCountries(): ArrayList<Country>
+ addCountry(Country): void
countries: ArrayList<Country>
currentRace: int
raceWeeks: ArrayList<RaceWeek>
isStarted: boolean
numberOfRaces: int
Team
name: String
NotificationHandler
+ add(String, TeamManager): void
+ update(): void
firstNotificationMessage: String
firstNotificationTeamManager: String
<<interface>>
Application
+ start(): void
+ logInFrame(): Object
<<interface>>
User
+ displayName(): void
+ controlName(): String
+ control(): JPanel
AdminControl
+ panel(): JPanel
+ notifyObserver(): void
TeamManagerControl
+ panel(): JPanel
Administrator
+ controlName(): String
+ displayName(): void
+ control(): JPanel
instance: Administrator
TeamManager
+ control(): JPanel
+ controlName(): String
+ displayName(): void
team: Team
WrongInputException
NoInputException
«create»
ApplicationControl
+ checkAddRaceInput(String, ArrayList<RaceWeek>, ArrayList<Country>): ArrayList<RaceW
+ checkLoggingIn(String, String): User
+ removeRace(Season, String): ArrayList<RaceWeek>
+ distanceFromPreviousRace(Country, Country): int
+ removeAddedCountry(String, ArrayList<Country>): ArrayList<Country>
ApplicationFrame
+ logInFrame(): JFrame
TeamManagerAccount
+ logIn(String): TeamManager?
StartApplication
+ main(String[]): void

Triedy

StartApplication

- StartApplication je hlavná trieda obsahujúca main, ktorá spúšťa aplikáciu.

ConstoleLogging

- ConsoleLogging je AspectJ trieda, ktorá vypisuje záznamy do konzoly.

AdminControl

- AdminControl je hlavná trieda pre Administrátora, ktorá rieši všetky jeho požiadavky.

ApplicationControl

- ApplicationControl je trieda, ktorá rieši pomocné no veľmi dôležité problémy prihlasovania, pridávania a mazania krajín, výpočty vzdialenosti použitím Viacnitovosti.

DistanceControl1 & DistanceControl2

- Tieto triedy fungujú paralelne na základe využitia nití a vypočítavajú obe odlišným spôsobom vzdialenosť krajín.

NotificationHandler

- Táto trieda poskytuje komunikáciu pomocou notifikácií od tímových manažérov smerom ku Administrátorovi.

TeamManagerAccount

- Táto trieda nám vytvára rozdielnych tímových manažérov na základe prihlasovacieho mena.

TeamManagerControl

- Hlavná trieda pre Tímových Manažérov, ktorá rieši všetky ich možnosti a požiadavky.

Administrator

- Trieda Administrátora.

ControlPanel

- Abstraktná trieda ControlPanel, ktorú dedia triedy AdminControl a TeamManagerControl.

Country

- Trieda Krajiny, kde sa budú konať preteky.

NoInputException

- Trieda vlastnej výnimky pokiaľ je prihlasovacie textové pole prázdne.

RaceWeek

- Trieda týždňa v ktorom sa odohrávajú preteky.

Season

- Najdôležitejšia trieda obsahujúca všetky údaje o stave sezóny, nasledujúcich pretekoch, krajinách.

Team

- Trieda s názvom tímu.

TeamManager

- Trieda tímový manažér, ktorý riadi jeden tím.

User

- Rozhranie používateľa.

WrongInputException

- Trieda vlastnej výnimky pokiaľ je prihlasovacie textové pole nesprávne.

Application

- Rozhranie aplikácie.

ApplicationFrame

- Hlavná trieda obsahujúca prihlasovacie menu.

Prehľad odovzdaných verzií

Pri vytváraní finálnej verzie projektu sme vytvorili celkovo pod 30 commitov, ktoré boli odovzdané na GitHub. Väčšina z nich pozostávala z postupného pridávania nových funkcionalít. Každý commit obsahuje komentár v ktorom bolo popísané na čom konkrétnom sa pracovalo. Niektoré commity nemali za úlohu obsah pridávať ale naopak odoberať, prípadne sme commitom spojili lokálne a vzdialené vetvy, ktoré neboli totožné.

Posledná a finálna verzia projektu je commit, ktorý nesie názov „Final Version“.

Splnené hlavné kritéria

Miera splnenia zadania dosahuje cieľ určený podľa zámeru, administrátor môže vytvoriť sezónu z doporučených krajín alebo pridať inú krajinu. Plánovanie sezóny pozostáva z pridávania pretekov, ktoré aplikácia odporúča na základe regiónov v ktorých sa odohrávajú predchádzajúce preteky a taktiež podľa teploty v danom období. Následne vie administrátor sezónu spustiť. Nasledujúcim prihlásením do systému vie administrátor simulovať priebeh sezóny posúvaním sa na ďalšie preteky. Počas behu sezóny dokáže preteky zrušiť alebo celú sezónu zrušiť. Administrátor dokáže obdržať notifikácie od tímových manažérov so správami, tie vedia po začiatku sezóny manažéri poslať a tým kontaktovať administrátora o možných problémoch.

V projekte využívame dedenie vo viacerých triedach (AdminControl, TeamManagerControl, ApplicationFrame, DistanceControl1, DistanceControl2), využívame rozhrania (rozhranie User, rozhranie Application a rozrania Serializable, Thread a Exception).

Polymorfizmus využívame v triede ApplicationFrame, kde volaním metódy user.control() obdržíme rozdielne výsledky v podobe menu pre administrátora alebo menu pre manažéra a taktiež aj v triedach NotificationHandler, ApplicationControl a AdminControl.

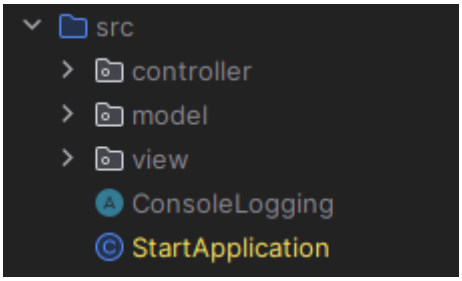
Zapúzdrenie môžeme nájsť v takmer každej triede (18 tried z 20 využíva zapúzdrenie).

Agregáciu vieme taktiež nájsť viackrát, trieda AdminControl má triedu Season, NotificationHandler a podobne. Kompozíciu vieme nájsť v triede TeamManager.

Splnené ďalšie kritéria

Návrhové vzory

- Využívame návrhový vzor MVC (Model, View, Controller) pre triedenie tried do balíkov.



- Využívame návrhový vzor Observer pre zistenie notifikácií.

```
public void notifyObserver() {
    if(handler != null) handler.update();
}
```

Trieda AdminControl

```
public void update() {
    if(notifications.size() > 0) {
        JFrame notificationFrame = new JFrame( title: "MYS | Notification");
        JPanel notificationPanel = new JPanel(new GridLayout( rows: 0,  cols: 1));

        JLabel notificationTitle = new JLabel( text: "Notification", SwingConstants.CENTER);
        notificationTitle.setFont(new Font( name: "Arial", Font.PLAIN,  size: 25));
        JLabel notificationTeam = new JLabel( text: "From: " + getFirstNotificationTeamManager(), SwingConstants.LEFT);
        notificationTeam.setFont(new Font( name: "Arial", Font.BOLD,  size: 20));
        JLabel notificationMessage = new JLabel(getFirstNotificationMessage(), SwingConstants.CENTER);
        notificationTeam.setFont(new Font( name: "Arial", Font.ITALIC,  size: 20));

        notificationPanel.add(notificationTitle);
        notificationPanel.add(notificationTeam);
        notificationPanel.add(notificationMessage);
```

Trieda NotificationHandler

- Využívame návrhový vzor Singleton pre vytvorenie administrátora.

```
public static Administrator getInstance() {
    if (administrator == null) {
        administrator = new Administrator();
    }
    return administrator;
}
```

Trieda Administrator

Vlastná výnimka

- Využívame vlastnú výnimku NoInputException v prípade, že sú textové polia mena alebo hesla pre prihlásenie prázdne.

```
public class NoInputException extends Exception{
    /**
     * Konštruktor triedy NoInputException.
     * Vypisuje chybovú správu, že vstup pre používateľské meno alebo heslo je prázdny.
     */
    Adam Jurcisin +1
    public NoInputException() { System.out.println("Empty Username or Password Input"); }
}
```

Trieda NoInputException

- Využívame vlastnú výnimku WrongInputException v prípade, že sú zadané nesprávne prihlasovacie údaje.

```
public class WrongInputException extends Exception{
    /**
     * Konštruktor triedy WrongInputException.
     * Vypisuje chybovú správu, že vstup pre používateľské meno alebo heslo je nesprávny.
     */
    Adam Jurcisin
    public WrongInputException() { System.out.println("Wrong Username or Password"); }
}
```

Trieda WrongInputException

Grafické používateľské rozhranie

- Pre grafické používateľské rozhranie využívame Java Swing knižnice, oddelenie logiky máme zabezpečené MVC návrhovým vzorom.

Viacniťovosť

- V triedach DistanceControl1 a DistanceControl2 využívame pre výpočet vzdialenosti dvoch krajín od seba na základe rozdielnych výpočtov.

```java
//THREADS TO GET DISTANCE BETWEEN COUNTRIES
DistanceControl1 thread1 = new DistanceControl1();
DistanceControl2 thread2 = new DistanceControl2();

thread1.run(regions, prevCountry, nextCountry);
thread2.run(regions, prevCountry, nextCountry);

return Math.min(thread1.getDistance(), thread2.getDistance());
```

Trieda ApplicationControl

Vhniezdené triedy

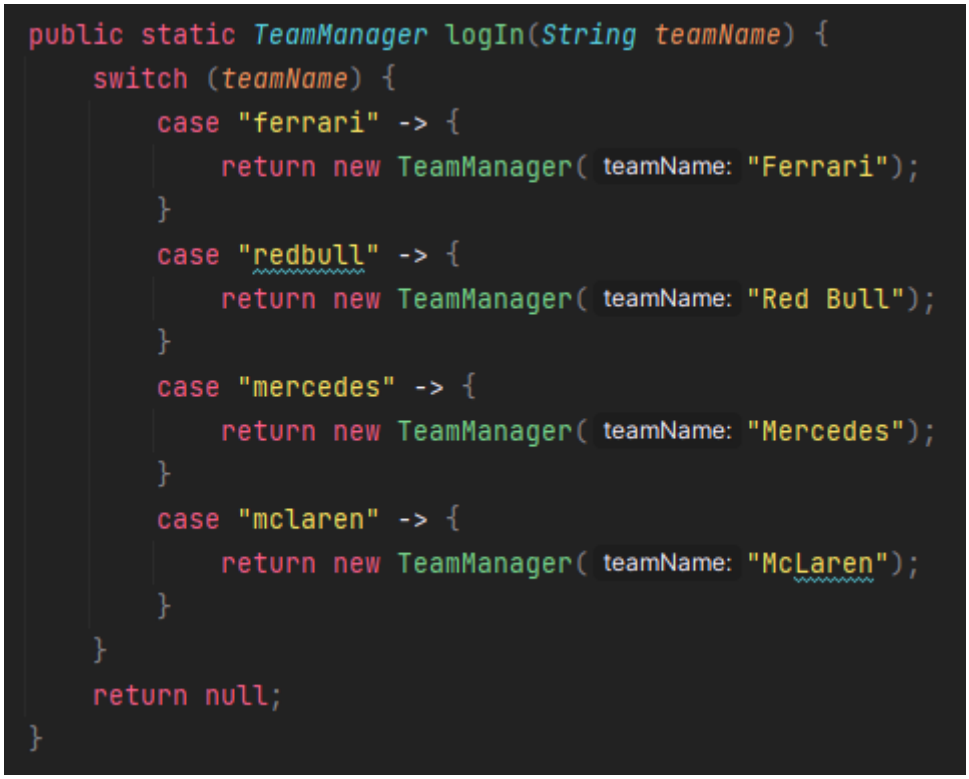
- V triede NotificationHandler máme vhniezdenú triedu Notification.

```java
public class NotificationHandler implements Serializable {
    private ArrayList<Notification> notifications;

    /**
     * Konštruktor pre vytvorenie novej inštancie NotificationHandler.
     * Vytvára prázdny zoznam notifikácií.
     */
    Adam Jurcisin
    public NotificationHandler() { notifications = new ArrayList<>(); }

    Adam Jurcisin
    private class Notification implements Serializable{
        private TeamManager teamManager;
        private String message;

        /**
         * Konštruktor pre vytvorenie novej notifikácie.
         * @param message      správa notifikácie
         * @param teamManager manažér tímu notifikácie
         */
        Adam Jurcisin
        private Notification(String message, TeamManager teamManager) {
            this.message = message;
            this.teamManager = teamManager;
        }
```

Trieda NotificationHandler

Lambda výrazy

- Lambda výrazy využívame v triede TeamManagerAccount pre prihlásenie tímového manažéra.



Trieda TeamManagerAccount

- Lamba výrazy využívame taktiež v GUI pri tlačidlách volaním metód ActionListener.

Implicitná implementácia metódy v rozhraní

- V rozhraní Application využívame implicitnú implementáciu metódy pre štart aplikácie.

```java
public interface Application {

    /**
     * Metóda start inicializuje aplikáciu.
     */
    Adam Jurcisin
    default void start() {
        Object frame = logInFrame();
    }
```

Rozhranie Applicaton

Použitie AspectJ

- Použitím AspectJ zapisujeme do konzoly vykonávané operácie.

```java
public aspect ConsoleLogging {

    before(): execution(* ApplicationControl.checkLogIn(String, String)) {
        System.out.println("Checking logging in");
    }
    after(): execution(* ApplicationControl.checkLogIn(String, String)) {
        System.out.println("Logged in");
    }
    after(): execution(* ApplicationControl.checkAddRaceInput(String, ArrayList<RaceWeek>, ArrayList<Country>)) {
        System.out.println("Checking Adding Race");
    }
}
```

Aspect ConsoleLogging

Použitie serializácie

- Serializáciu využívame mnohokrát pri uložení stavu Sezóny

```java
// DESERIALIZATION - načítanie Season zo súboru "season.ser"
try {
    FileInputStream fileIn = new FileInputStream( name: "season.ser");
    ObjectInputStream in = new ObjectInputStream(fileIn);
    season = (Season) in.readObject();
    in.close();
    fileIn.close();
} catch(IOException i) {
    i.printStackTrace();
} catch(ClassNotFoundException c) {
    System.out.println("Season class not found");
    c.printStackTrace();
}
```

Deserializácia v Triede AdminControl

- Taktiež pre uloženie stavu Notifikácií

```java
// DESERIALIZATION - načítanie NotificationHandler zo súboru "notificationHandler.ser"
try {
    FileInputStream fileIn = new FileInputStream( name: "notificationHandler.ser");
    ObjectInputStream in = new ObjectInputStream(fileIn);
    handler = (NotificationHandler) in.readObject();
    in.close();
    fileIn.close();
} catch(IOException i) {
    i.printStackTrace();
} catch(ClassNotFoundException c) {
    System.out.println("NotificationHandler class not found");
    c.printStackTrace();
}
```

Deserializácia v Triede AdminControl