

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Použité OOP princípy

ZOOP – Základy objektovo-orientovaného programovania

Andrej Stuchlý

Utorok 12:00-13:50

1. Funkcionalita

Kód je plne funkčný tak, ako som ho zamýšľal – na začiatku vytvára všetky potrebné dáta (jazdcov, tímov a kalendár zloží z pretekov), simuluje celú sezónu pomocou simulácii pretekov: určí im počasie, podľa neho nastaví jazdcom pneumatiky, náhodne ich preusporiada a vykoná predbiehajúce podľa typu a stavu pneumatík. V závislosti od počasia sa tiež dejú ďalšie veci: ak je sucho, jazdcom sa degraduje kvalita pneumatík, čo sťažuje predbiehajúce, pri mokrom počasí majú jazdci šancu roztočiť sa a spadnúť na koniec poradia pretekov. Na konci sezóny sa vypíšu jazdci a tímy zoradení podľa počtu bodov.

```
Highlights of race in USA - Austin:
Valtteri Bottas spun
Carlos Sainz overtook Max Verstappen
Sergio Perez overtook George Russell
Lewis Hamilton overtook Pierre Gasly
Esteban Ocon overtook Lance Stroll
Alexander Albon spun
Pierre Gasly spun

Standings of race in USA - Austin, Weather: Rainy
1. Yuki Tsunoda
   Tyre choice: Inter Tyre
   Points for race: 25
2. Daniel Ricciardo
   Tyre choice: Full Wet Tyre
   Points for race: 18
3. Mick Schumacher
   Tyre choice: Full Wet Tyre
   Points for race: 15
```

```
Final standings of drivers:
Yuki Tsunoda total points: 218
Lando Norris total points: 188
Kevin Magnussen total points: 159
Valtteri Bottas total points: 155
Nicolas Latifi total points: 153
Lewis Hamilton total points: 149
Alexander Albon total points: 122
Esteban Ocon total points: 121
Lance Stroll total points: 121
George Russell total points: 103
Pierre Gasly total points: 95
Daniel Ricciardo total points: 94
Gunayu Zhou total points: 92
Sergio Perez total points: 92
Mick Schumacher total points: 78
Max Verstappen total points: 76
Charles Leclerc total points: 70
Sebastian Vettel total points: 57
Fernando Alonso total points: 47
Carlos Sainz total points: 32

The champion is Driver Yuki Tsunoda

Final standings of teams:
AlphaTauri total points: 313
McLaren total points: 282
Williams total points: 275
```

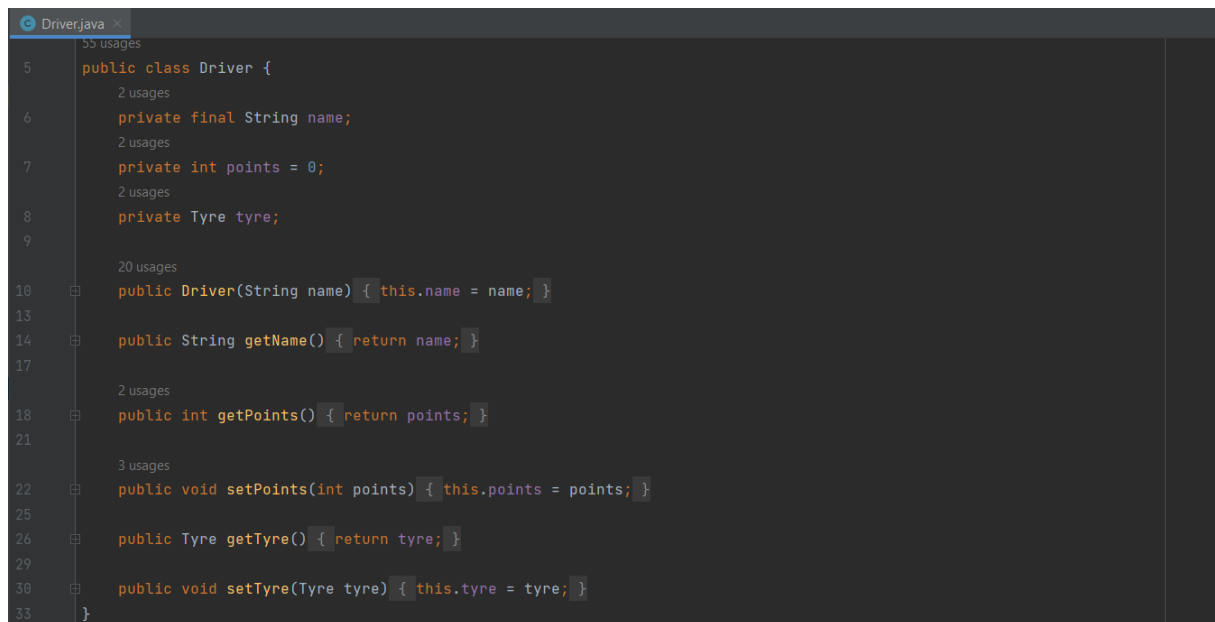
2. Dedenie

Dedenie sa vyskytuje u pneumatík – napríklad Classy FullWetTyres a InterTyres dedia od Classy WetTyres, ktorá dedí z abstraktnej Classy Tyres.

```
1 package entity.tyres;
2
3 public abstract class Tyre {
4     int speed;
5
6     public abstract int getSpeed();
7     public abstract void setSpeed(int speed);
8     public abstract String getName();
9 }
10
11 package entity.tyres;
12
13 public class WetTyre extends Tyre {
14     int grip;
15
16     public WetTyre() {
17     }
18
19     public int getSpeed() { return this.speed; }
20     public void setSpeed(int speed) { this.speed = speed; }
21     public String getName() { return "WetTyre"; }
22     public int getGrip() { return grip; }
23 }
24
25 package entity.tyres;
26
27 public class InterTyre extends WetTyre {
28     int speed = 30;
29     int grip = 50;
30
31     public InterTyre() {
32     }
33
34     public int getSpeed() { return this.speed; }
35     public void setSpeed(int speed) { this.speed = speed; }
36     public String getName() { return "Inter Tyre"; }
37 }
```

3. Zapuzdrenie

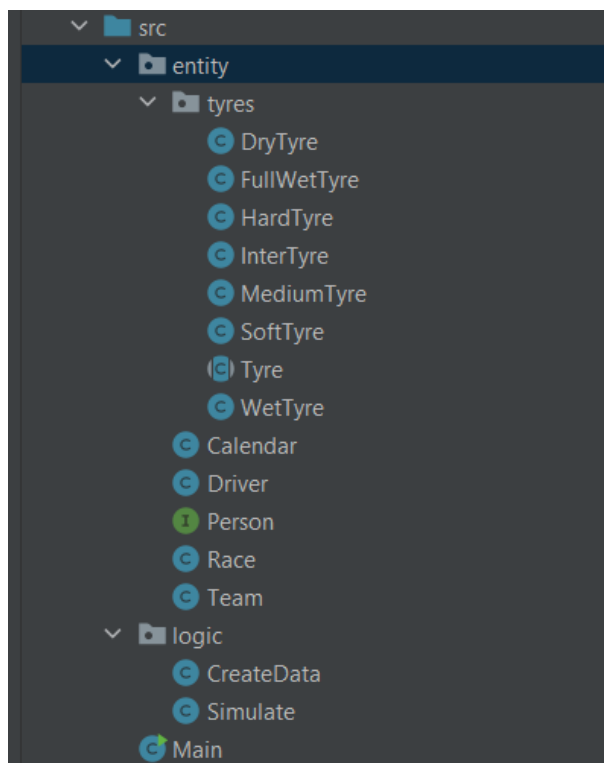
V Classe Driver pristupujem k private atribútom cez getery a setery.



```
5 public class Driver {
6     private final String name;
7     private int points = 0;
8     private Tyre tyre;
9
10    20 usages
11    public Driver(String name) { this.name = name; }
12
13
14    2 usages
15    public String getName() { return name; }
16
17
18    2 usages
19    public int getPoints() { return points; }
20
21
22    3 usages
23    public void setPoints(int points) { this.points = points; }
24
25
26    public Tyre getTyre() { return tyre; }
27
28
29    public void setTyre(Tyre tyre) { this.tyre = tyre; }
30
31
32
33 }
```

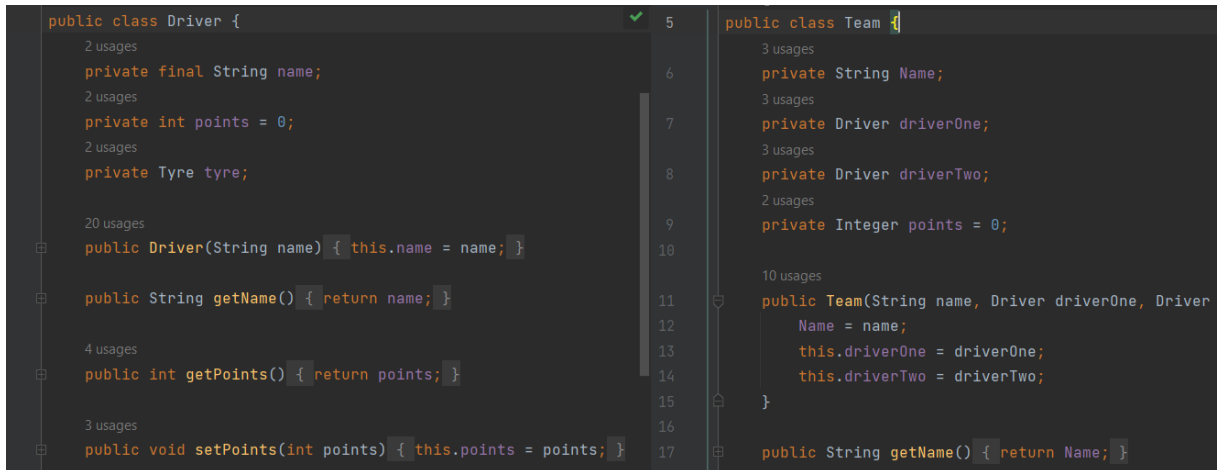
4. Organizácia do balíkov

Projekt je roztriedený do adekvátnych balíkov pre vhodnú prehľadnosť



5. Konzistentný štýl písania

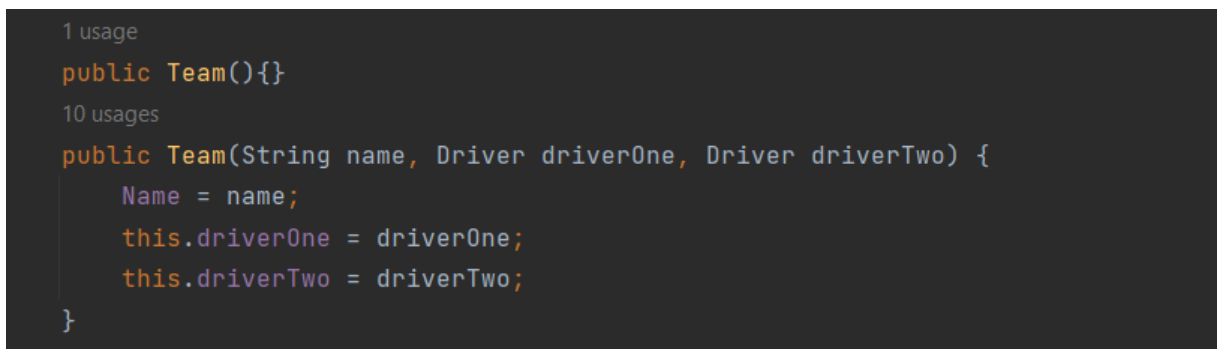
Konzistentné pomenúvanie pri Classach aj metódach – pri Classach som používal všade PascalCase, pri metódach zas camelCase



```
public class Driver {  
    2 usages  
    private final String name;  
    2 usages  
    private int points = 0;  
    2 usages  
    private Tyre tyre;  
  
    20 usages  
    public Driver(String name) { this.name = name; }  
  
    public String getName() { return name; }  
  
    4 usages  
    public int getPoints() { return points; }  
  
    3 usages  
    public void setPoints(int points) { this.points = points; }  
}  
  
public class Team {  
    3 usages  
    private String Name;  
    3 usages  
    private Driver driverOne;  
    3 usages  
    private Driver driverTwo;  
    2 usages  
    private Integer points = 0;  
  
    10 usages  
    public Team(String name, Driver driverOne, Driver  
        Name = name;  
        this.driverOne = driverOne;  
        this.driverTwo = driverTwo;  
    }  
  
    public String getName() { return Name; }  
}
```

6. Preťažovanie – Overloading

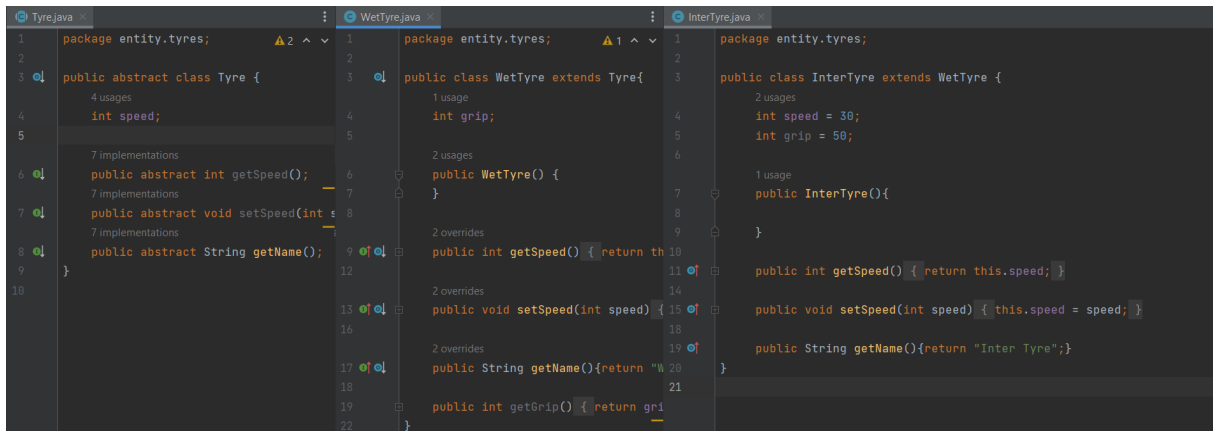
Tento princíp som využil pri konštruktoroch Driver a Team, nakoľko pri šampiónoch chcem bezparametrový konštruktor, ale vo vytváraní dát potrebujem hodnoty



```
1 usage  
public Team(){}  
  
10 usages  
public Team(String name, Driver driverOne, Driver driverTwo) {  
    Name = name;  
    this.driverOne = driverOne;  
    this.driverTwo = driverTwo;  
}
```

7. Prekonávanie – Overriding

Prekonávanie využívam napríklad pri dedení pneumatík a funkcii getName (riadky 8, 17, 19).



```
1 package entity.tyres;
2
3 public abstract class Tyre {
4     int speed;
5
6     public abstract int getSpeed();
7     public abstract void setSpeed(int speed);
8     public abstract String getName();
9 }
10
11 package entity.tyres;
12
13 public class WetTyre extends Tyre {
14     int grip;
15
16     public WetTyre() {
17     }
18
19     public int getSpeed() { return this.speed; }
20     public void setSpeed(int speed) { this.speed = speed; }
21     public String getName() { return "Wet Tyre"; }
22     public int getGrip() { return grip; }
23 }
24
25 package entity.tyres;
26
27 public class InterTyre extends WetTyre {
28     int speed = 30;
29     int grip = 50;
30
31     public InterTyre() {
32     }
33
34     public int getSpeed() { return this.speed; }
35     public void setSpeed(int speed) { this.speed = speed; }
36     public String getName() { return "Inter Tyre"; }
37 }
```

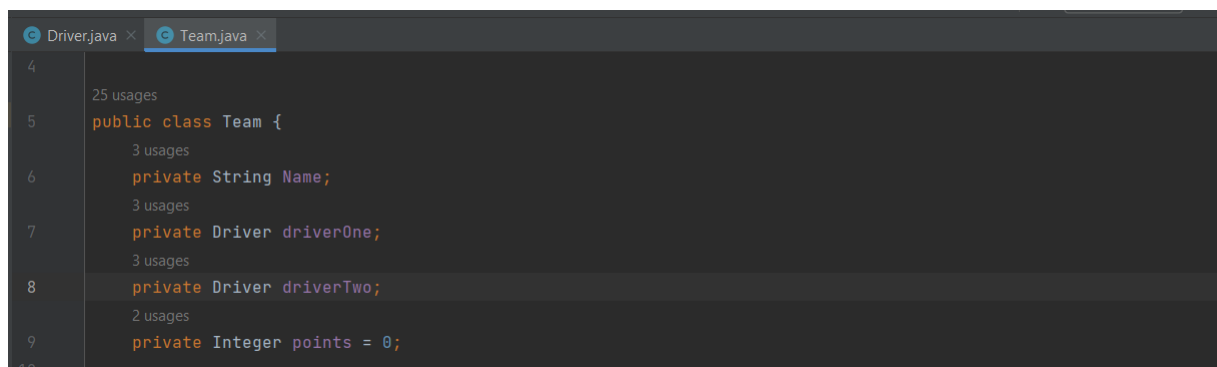
9. Kompozícia

Tento princíp som využil pri kalendári, aby sa pri ňom zobrazoval šampión medzi jazdcami aj tímami

```
public class Calendar {
    private Race[] races;
    private Driver championDriver = new Driver();
    private Team championTeam = new Team();
}
```

10. Asociácia

Classa Team obsahuje atribúty typu Driver – ide o atribúty driverOne a driverTwo.



```
4
5 public class Team {
6     private String Name;
7     private Driver driverOne;
8     private Driver driverTwo;
9     private Integer points = 0;
10 }
```

11. Finálny atribút

Rozhodol som sa, že tento atribút využijem pri Stringu lokácia pretekov, nakoľko sa nemení.

```
Race.java x
6   public class Race {
7       3 usages
8       private final String location;
9       4 usages
10      private Driver[] driverStandings;
11      3 usages
12      private String weather;
```

12. Finálna metóda

Finálnu metódu je možné nájsť opäť v Classe Race, tentokrát pri výpise výsledku pretekov.

```
Race.java x
42      public final void printRaceResults(){
43          1 usage
44          int position = 1;
45          System.out.println("Race in "+ location +", Weather: "+weather);
46          for(Driver driver: driverStandings){
47              System.out.println(position+" "+driver.getName() + " \n\tTyre choice: "+ driver.getTyre().getName()
48                  +"\n\tPoints for race: "+Simulate.pointsForPosition(position));
49              position++;
50          }
51          System.out.println("\n");
52      }
53  }
54  }
```

13. Abstraktná trieda

Na základe abstrakcie som postavil začiatok dedenia – Classa Tyre je len abstraktná.

```
Tyre.java x
1   package entity.tyres;
2
3   public abstract class Tyre {
4       4 usages
5       int speed;
```

14. Abstraktná metóda

Taktiež je možné vidieť abstrakciu vo vyššie spomenutej classe pri jej abstraktných metódach

A screenshot of an IDE showing the code for `Tyre.java`. The code defines an abstract class `Tyre` in the `entity.tyres` package. It has a private static field `speed` of type `int`. There are three abstract methods: `getSpeed()` returning `int`, `setSpeed(int speed)` returning `void`, and `getName()` returning `String`. The IDE shows 4 usages for the class and 7 implementations for each of the three methods.

```
1 package entity.tyres;
2
3 public abstract class Tyre {
4     int speed;
5
6     public abstract int getSpeed();
7     public abstract void setSpeed(int speed);
8     public abstract String getName();
9 }
```

15. Statická metóda

Využitie statickej metódy je u mňa v kóde vidno v celej Classe Simulate, nakoľko sa skladá len zo statických metód.

A screenshot of an IDE showing the code for `Simulate.java`. The code defines a class `Simulate` with several static methods. The methods are: `setDriverTyres(Driver driver, int tyreChoice)`, `setTyres(Driver[] standings, String weather)`, `shuffleDrivers(Driver[] standings)`, `overtakes(Driver[] standings)`, `pointsForPosition(int pos)`, `givePoints(Driver[] standings, Team[] teams)`, and `simRace(Race race, Team[] teams)`. The IDE shows 6 usages for the class and 1 usage for each of the seven methods.

```
10
11 public class Simulate {
12
13     public static void setDriverTyres(Driver driver, int tyreChoice){...}
14
15     public static void setTyres(Driver[] standings, String weather){...}
16
17     public static void shuffleDrivers(Driver[] standings){...}
18
19     public static void overtakes(Driver[] standings){...}
20
21     public static int pointsForPosition(int pos){...}
22
23     public static void givePoints(Driver[] standings, Team[] teams){...}
24
25     public static void simRace(Race race, Team[] teams){...}
26 }
```

16. Statický atribút

Tento typ atribútu som sa rozhodol použiť pri pneumatikách DryTyre a WetTyre, nakoľko tieto atribúty sa ešte nijako konkrétne nevyužívajú a je v poriadku volať ich staticky a nie z inštancie

```
3 usages 2 inheritors
public class WetTyre extends Tyre{
    1 usage
    static int grip;
```

17. Vlastné rozhranie

Rozhranie je možné nájsť pri Classe Driver, ktorá implementuje Person a hovorí o popise práce daného človeka v metóde jobDescription.

```
59 usages
public class Driver implements Person{
    2 usages
    private String name;
    2 usages
    private int points = 0;
    2 usages
    private Tyre tyre;

    1 usage
    public Driver(){ }

    20 usages
    public Driver(String name) { this.name = name; }

    no usages
    public String jobDescription(){
        return "This person works in Formula 1 as a driver.";
    }
}
```


19. Downcasting

Koncept je využitý pri vytváraní pneumatík – najprv sa vytvorí všeobecnejší typ – Wet/Dry Tyre a následne sa downcastne na konkrétny typ z daných 5, ktorý už vie používať všetky potrebné atribúty a metódy pre správny chod programu

```
public static void setDriverTyres(Driver driver, int tyreChoice){
    switch (tyreChoice) {
        case 0 -> {
            DryTyre dryTyre = new SoftTyre();
            SoftTyre softTyre = (SoftTyre) dryTyre;
            driver.setTyre(softTyre);
        }
        case 1 -> {
            DryTyre dryTyre = new MediumTyre();
            MediumTyre mediumTyre = (MediumTyre) dryTyre;
            driver.setTyre(mediumTyre);
        }
        case 2 -> {
            DryTyre dryTyre = new HardTyre();
            HardTyre hardTyre = (HardTyre) dryTyre;
            driver.setTyre(hardTyre);
        }
        case 3 -> {
            WetTyre wetTyre = new InterTyre();
            InterTyre interTyre = (InterTyre) wetTyre;
            driver.setTyre(interTyre);
        }
        case 4 -> {
            WetTyre wetTyre = new FullWetTyre();
            FullWetTyre fullWetTyre = (FullWetTyre) wetTyre;
            driver.setTyre(fullWetTyre);
        }
    }
}
```

20. Singleton

Takto som sa rozhodol zakomponovať Singleton do projektu: Classa Calendar je riešená tak, že pokiaľ neexistuje inštancia kalendára, vytvorí sa nový

```
public class Calendar {
    2 usages
    private Race[] races;
    2 usages
    private Driver championDriver = new Driver();
    2 usages
    private Team championTeam = new Team();
    3 usages
    private static Calendar singletonCalendar = null;
    1 usage
    public Calendar(Race[] races) {
        this.races = races;
    }
    1 usage
    public static Calendar getCalendar(Race[] races) {
        if (singletonCalendar == null)
            singletonCalendar = new Calendar(races);

        return singletonCalendar;
    }
}
```

21. Vyznačenie princípov

Požadované princípy sú vyznačené a ukázané v tomto súbore

22. Prekonávanie v entitách

V každej entite dochádza k prekonávaniu základných metód, na ukážku som pridal fotku z Classy Driver

```
public String toString(){
    return "Driver "+getName();
}

public int hashCode(){
    return getName().length()*getName().charAt(0);
}

1 usage
public boolean equals(Driver other){
    return hashCode() == other.hashCode();
}
```

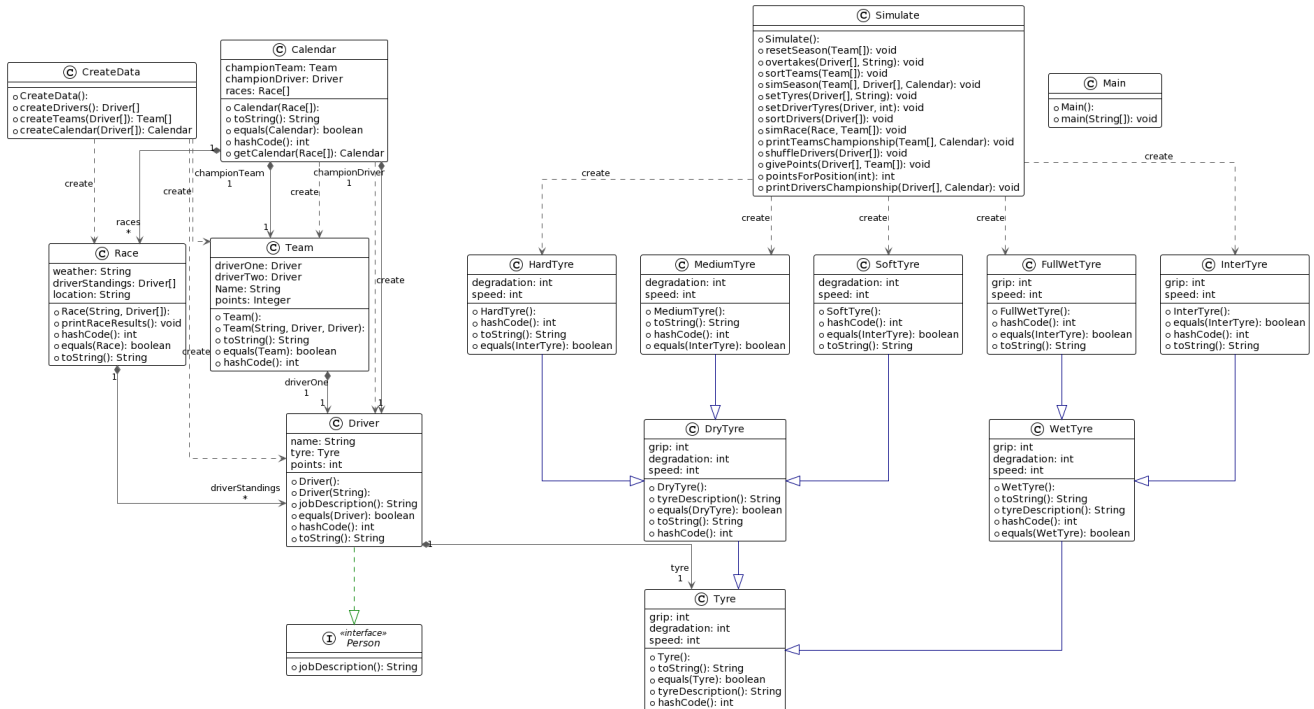
23. Správne použitie názvov

Kód aj komentáre sú písane v angličtine, Classy PascalCasom, metody camelCasom (spomínane už v bode 5)

```
7  heritors
public abstract class Tyre {
    4 usages
    int speed;

    3 usages 7 implementations
    public abstract int getSpeed();
}
```

24. UML diagram



25. Dokumentácia kódu

Kód je dostatočne rozdelený do vlastných Class a funkcií pre prehľadnosť, k daným funkciám sú aj pridané komentáre pre väčšiu prehľadnosť

```
//final print of championship of teams
1 usage
public static void printTeamsChampionship(Team[] teams) {
    sortTeams(teams);
    System.out.println("\nFinal standings of teams:");
    for (Team t:teams){
        System.out.println(t.getName() + " total points: "+t.getPoints());
    }
}

//simulation of whole season
1 usage
public static void simSeason(Team[] teams, Driver[] drivers, Calendar calendar){
    for(Race race: calendar.getRaces()){
        simRace(race, teams);
    }
    printDriversChampionship(drivers);
    printTeamsChampionship(teams);
}

//reset all points
1 usage
```