Algoritmos e Estruturas de Dados

2022/23

Projecto 1 – Abstract Data Types

(v2)

Introduction

In the 1st Project, it is intended that students become familiar with the java language, with the development environment, and that they learn to create and use Abstract Data Types in Java. This project is relatively easy and was inspired by the “Oliveira” and “Portimão” Problems from the 1st “Festa” of the Laboratório de Programação course in the academic year 2021/2022. Thus, students will already be familiar with the data type and the desired behaviour.

Problema A: Implementation of TAI Race (3 valores)

Implement the ***Race*** data type, based on the following specification:

TAI *Race* – Represents the performance of a motorcyclist in a given race (circuit and year) of a championship. Implement the following methods:
`Race(String circuit, String racer, int year, int position, boolean finished)`
Constructor that creates a race with the information received: circuit refers to the name or abbreviation of the circuit, racer to the driver's name, year the year the race took place, position is the position reached if the driver has finished the race, finished is a logical value that indicates whether the driver has raced and finished the race.
`String getCircuit()`
Returns the circuit’s name
`String getRacer()`
Returns the pilot’s name
`int getYear()`
Returns the race’s year

`int getPosition()`	
Returns the position where the driver ended the race. If the driver has not finished the race, 0 must be returned.	
`boolean getFinished()`	
Returns true if the driver finished the race, and false otherwise.	
`String toString()`	
Returns a string with all information from the race, using the following format: <circuit>-<year>-<racer>: <position> If a pilot did not finish the race, the position element should not be included in the string. Example: POR-2015-Miguel Oliveira: 8	

Implement type Race in a file named Race.java. The file must not declare any package[1]. Submit **only the file Race.java** in Problem A.

Problema B: Implementação do TAI Championship (5 valores)

Implemente o Tipo Abstracto de Informação ***Championship***, tendo em conta a seguinte especificação:

TAI *Championship* – represents the performance of a pilot in a motorcycling championship in a given year. L
`Championship(int year, String racer, String category, String bike, int position, int points)`
Constructor that creates a championship with the information received: Year refers to the year of the championship, racer to the pilot's name, category is the type of motorcycle used in the championship, and bike is the name of the team the pilot is racing for, position is the final position obtained in the championship and points the total number of points obtained by the pilot in the championship.
`int getYear()`
Returns the championship's year
`String getRacer()`
Returns the championship's pilot name
`String getCategory()`
Returns the bike's category or type
`String getBike()`
Returns the name of the bike/team
`int getPosition()`
Returns the pilot's final position in the championship
`int getPoints()`
Returns the pilot's total points in the championship
`void addRace(int raceNumber, Race r)`
Adds a race r (or changes the existing race) to the championship, in the order corresponding to the race number. Races are numbered between 1 and 21. If the race received does not match the championship information (i.e. it corresponds to a different driver or year) an exception of type *IllegalArgumentException* must be thrown indicating the reason. If the race number received is invalid, an exception of type *IndexOutOfBoundsException* must be thrown.
`Race getRace(int raceNumber)`

[1] A class defined in a file without package declaration is considered to belong to the default package (also known as unnamed)

Returns the race corresponding to the race number received. If there is no race defined for this number, null must be returned. If the race number received is invalid, an exception of type *IndexOutOfBoundsException* must be thrown.	
`String`	`toString()`
Returns a String with the complete championship information with the following format: <racer>/<year>/<category>/<bike>\n<race1>/…/<race21> Where <raceN> represents information about race N and has the format: <circuit> <position> If the driver has not finished a race, the position must not be included in the string. If there is no information about a run N, a single whitespace “ “ must be used. Example (the '\n' causes a line break when printed on the screen): Miguel Oliveira/2015/Moto3/KTM QAT 16/AME /ARG 4/SPA 8/…/ / /	
`int`	`compareTo(Championship c2)`
Compare the championship with another c2 championship in order to sort them. Returns 0 if they are equal, returns an integer >= 1 if the championship is to be sorted after c2 and an integer <= -1 if it is to be sorted before c2. The sorting criteria are as follows: 1st: Pilot's name (ascending alphabetical order) 2nd: Year (descending order)	

Implement the data type Championship in a file Championship.java. The file must not declare any package. Submit **only the file Championship.java** in Problem B.

Problema C: Implementation of Library Class Utils (12 Valores)

In this Problem we will use the data types created previously to implement functions that operate on championships and races. For this we will implement a library class, called Utils.java that implements the following static methods:

`static List<Championship>`	`readChampionshipsFromCSV (String fileName)`
Given a String with the name of a csv file, this method should open the file and read a set of championships, knowing that the 1st line of the file is a line with the name of the columns, and that each two of the following lines correspond to information about a championship. Each row contains 27 columns, and all columns are separated by commas. Example: Racer,Year,Class,Bike,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,1 9,20,21 ,Pos,Pts Miguel,2011,125cc,Aprilia,QAT,SPA,POR,FRA,CAT,GBR,NED,ITA,GER,CZE,IND,RSM,ARA,JPN,AUS,MAL,VAL,,,,,14th,44 ,,,,10,Ret,7,9,Ret,,,8,Ret,23,8,10,Ret,,,,,,,,, Miguel,2015,Moto3,KTM,QAT,AME,ARG,SPA,FRA,ITA,CAT,NED,GER,IND,CZE,GBR,RSM,ARA,JPN,AUS,MAL,VAL,,,,2nd,254 ,,,16,Ret,4,2,8,1,5,1,DNS,15,8,13,2,1,2,1,1,1,,,,, In the example, the 1st line corresponds to the name of the columns, the 2nd and 3rd line correspond to a championship of the pilot Miguel (Oliveira) and the lines n.º 4 and 5 correspond to another championship of Miguel. The 1st row of a pair has general information about the championship, and the name of each circuit. The 2nd row of a pair has information on the position reached by the rider on each of the circuits. When for a circuit the column is empty or does not have a number, it means that the driver did not participate or did not finish the race. This method should return a list of all championships read from the file.	
`static void`	`sortChampionships(List<Championship> champs)`
Given a list of championships, it modifies the received list, ordering it by the Championship class ordering criterion.	
`static void`	`printChampionships (List<Championship>champs)`

Given a list of championships, write the championships to the standard output, writing one championship at a time in the order received. The format on each line will be the format used by the String representation of the Championship class.	
static List<Championship>	*filterByRacer(List<Championship> champs, String racer)*
Given a list of championships, and a driver's name, it returns a new list with all championships for that driver. If there is no championship for this driver, an empty list should be returned.	
static List<Championship>	*filterByTeam(List<Championship> champs, String teamName, String category)*
Given a list of championships, the name of a team, and the category, it returns a new list with the championships of all the drivers of that team for that category. If there is no championship for that team, an empty list should be returned.	
static void	*printRacerStats(List<Championship> champs, String racer)*
Given a list of championships (which may belong to more than one driver), and a driver's name, it calculates and prints the driver's statistics on the screen, in the following format: Racer: <name> 1st: <firstplaces> 2nd: <secondplaces> 3rd: <thirdplaces>. Name refers to the driver's name, firstplaces to the number of times the driver was placed 1st, secondplaces to the number of times the driver was placed 2nd, and thirdplaces the number of times the driver was placed 3rd. Example: Racer: Miguel Oliveira 1st: 14 2nd: 11 3rd: 9	
static void	*printRacerCircuitRanking(List<Championship> champs, String racer)*
This method should calculate and print on the screen a list with the performance/ranking of a rider in his best circuits. To build the ranking, 1024 points are assigned to a circuit where the driver won, 512 to a circuit where he was in 2nd place, 256 to a circuit where he was in 3rd place, and so on, dividing by 2, up to 1 point for a circuit where you finished in 10th place. If you did not finish, or finished from 11th place, 0 points are awarded. Calculate the ranking for each circuit in which the driver has participated, order from best to worst circuit and write the rankings on the screen (one circuit per line), using the following format: <circuit_name> <points> Example: VAL 2888 MAL 2820 … TUES 32 Circuits with 0 points must not be printed. If there is a tie between 2 or more circuits, use ascending alphabetic order (of the circuit names) as the secondary ordering criterium.	
static List<Race>	*getBestRaces(List<Championship> champs, String racer)*
Given a list of championships (which can belong to more than one driver), and a driver's name, returns a list of all races in which the driver finished and placed at least 5th. The list should be sorted using the following criteria: 1st position obtained (descending order) 2nd Year of the race (descending order) 3rd Circuit name (ascending alphabetical order)	
static List<String>	*getTeams(List<Championship> champs, String category))*
Given a list of championships, and the name of a category, it returns a list of all teams for which there are records for the received category. Repeated names should not be returned.	

If you find it necessary to implement an auxiliary class, define an internal class (defined in the Utils file itself). Submit the **Utils.java file** in Problem C. The file cannot declare any package. It is not necessary to submit any more files.

Tips and suggestions

Although its submission is not required, we recommend the implementation of a Main class and a main method so that you can test the various types and static methods implemented.

The List type in Java is an Interface that is implemented by the ArrayList and LinkedList classes. We suggest the use of the ArrayList class to implement the indicated methods. Since sorting methods have not yet been lectured, you can use the sort method of the List interface which takes the comparison method to be used as an argument.

In this 1st project, the temporal and spatial complexities of the methods developed will not be evaluated, so it will not be necessary to worry too much about their efficiency. In future projects we will look in more detail at these aspects.

Realization conditions

The 1st project is worth 10% of the final grade in the frequency component of the course. The project must be carried out individually. Submission of copies or very similar projects (to other students) will lead to failure in the discipline. The lecturers of the course will be the only judges of what is considered or not to be copy in a project.

The project code must be delivered electronically, through the Mooshak system, **by 11:59 pm on October 7th**. Validations will take place the following week, **10-14 October**. Students will have to validate the code together with the teacher **during** laboratory hours corresponding to the Laboratory in which they are enrolled. **The evaluation and corresponding grade of the project will only take effect after the validation of the code by the teacher**.

The evaluation of code execution is done automatically through the Mooshak system, using various tests configured in the system. The execution time of each test is limited, as well as the memory used. More information about registration and access to the Mooshak system will be available soon.