Design

Considerations

File storage system suggested in the assignment specifications does not take into account the total number of points per athlete. A field will be added for this. The new “participants.txt” file will look like this:
r1, sprinter, Max, 21, VIC, 7
s2, swimmer, Claudia, 27, NT, 0
c9, cyclist, Bernardino, 18, ACT, 2
o2, official, Albert, 43, VIC, 0

All participants will be assigned 0 points at the start, including officials.
Taking this into account, it is no longer necessary for the “gameResults.txt” to store the number of points as it will be redundant. Instead it will store the place in the following way:

Database

The database used for this assignment is similar to the one given as an example in the assignment’s specification, which indicates two tables: one for participants and one for games. The database structure used includes three tables, one for participants, one for games and an auxiliary table. The structure is the following:

Participants:
ppt_id VARCHAR(11) -> The id of the participant
name VARCHAR(100)-> name of the participant
age INT-> age of the participant
state VARCHAR(4)-> state of the participant
points INT -> current points
type INT -> type of participant (1 = Sprinter, 2 = Swimmer, 3 = Cyclist, 4 = Super Athlete, 5 = Official)

Games:
game_id VARCHAR(11) -> id of the game
off_id VARCHAR(11)-> id of the official set for the game

Auxiliary:
game_id VARCHAR(11) -> id of game
ppt_id VARCHAR(11) ->participant id
score DOUBLE -> the score of the participant for that game
place INT -> the place that participant obtained in the race.

Design questions

1. Explain the changes if you use a different design compared to your assignment 1

A. A different design was used for this assignment only in the user interactions and data storage. For the first assignment, object constructors included decoding strings which were read from files, which made the code more messy. In this assignment Specific reading methods were introduced in a Reader class to improve Cohesion. The fact that this assignment had to include a database and file handling made it more challenging and introduced changes in the design, making the Reader class the heaviest and most important class. By using fxml and css a lot of coding was reduced, as the designs were no longer console based, this changed the design. Packaging was also a very important change as the first assignment only included one package, this assignment included 5 packages:

- control:
  - includes all the controllers for the events and generation of screens.
- exceptions
  - includes all checked exceptions that were used for the assignment
- games
  - includes the Game class as well as its subclasses
- participants
  - includes the Participant class as well as its subclasses, including Athlete and Official.
- storage
  - Includes the Reader class which reads and writes into the files and database.

2. Explain how the new classes are organized

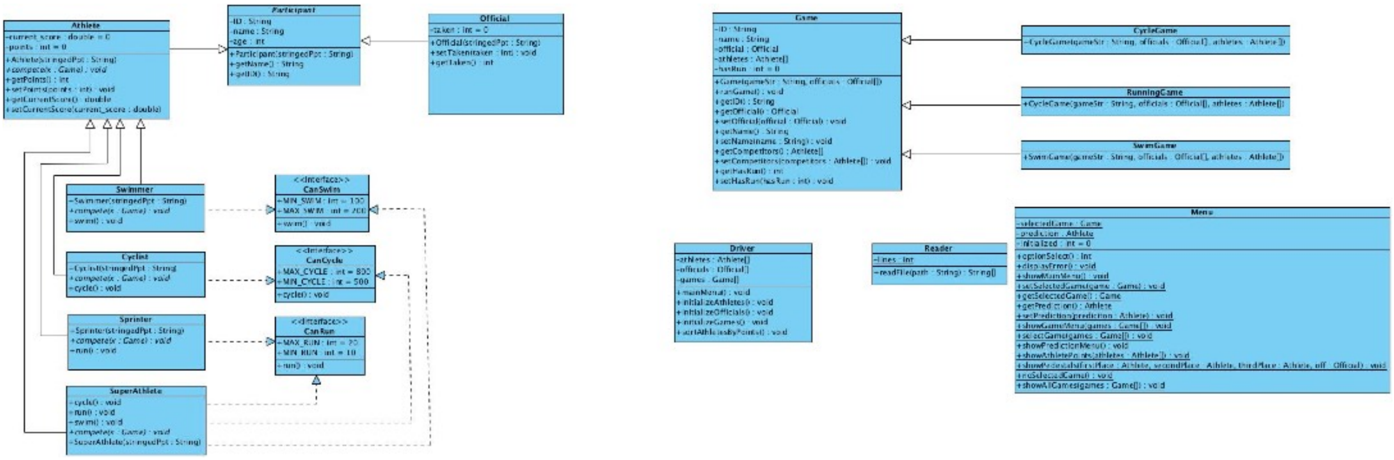
A. The Reader class is stored in the storage package and contains a variety of static methods for reading and writing files. All new exceptions are stored in a different package, and all Controller classes are stored in the control class. The Controller classes usually have an initialize method which will be called when the scene is displayed. Also, the controller classes have parameters for each element that can interact in the scene.

3. Explain the process by which your program will interact with user and external data source to run a game.

A. Using JavaFX elements, the interface designed is very simple to use and prevents errors by displaying messages to the user. The first screen is just an introduction to the game with a button. When the button is pressed, the main menu is displayed which shows three big buttons for each game with an image of the game, and two small buttons at the bottom which will display results. The main lobby for a game includes a drop-down list which allows the user to select a participant to add to a game or the athletes can be added at random. The user selects how many athletes they want to compete in the game. The official can also be selected and the game won't run unless there is an official and 4-8 athletes. There will also be a countdown to simulate an online game lobby in which people are added spontaneously until the timeout is finished. Once the timeout is finished or the user selects to run the game, a new screen is displayed which will show the tracks for the athletes and an image of a generic athlete with a label that indicates the name of the athlete. This image will move from one side of the track to the other and when it's finished the result will be recorded. Once all participants are done, a game over screen will be shown. The result showing are pretty simple as there is no interaction.

For the data source, the program initializes the database as soon as the first stage is initialized. Every time information about athletes or official is required, the database is consulted, if no result is returned, the text file will be checked as a backup. When a game is finished, the results are stored both in the database and in the text file to ensure synchronization.

Class Diagram


Athlete
-current_score : double = 0
-points : int = 0
+Athlete(stringedPpt : String)
+compete(x : Game) : void
+getPoints() : int
+setPoints(points : int) : void
+getCurrentScore() : double
+setCurrentScore(current_score : double)
Participant
-ID : String
-name : String
-age : int
+Participant(stringedPpt : String)
+getName() : String
+getID() : String
Official
-taken : int = 0
+Official(stringedPpt : String)
+setTaken(taken : int) : void
+getTaken() : int
Swimmer
-Swimmer(stringedPpt : String)
+compete(x : Game) : void
+swim() : void
<<Interface>>
CanSwim
+MIN_SWIM : int = 100
+MAX_SWIM : int = 200
+swim() : void
Cyclist
-Cyclist(stringedPpt : String)
+compete(x : Game) : void
+cycle() : void
<<Interface>>
CanCycle
+MAX_CYCLE : int = 800
+MIN_CYCLE : int = 500
+cycle() : void
Sprinter
-Sprinter(stringedPpt : String)
+compete(x : Game) : void
+run() : void
<<Interface>>
CanRun
+MAX_RUN : int = 20
+MIN_RUN : int = 10
+run() : void
SuperAthlete
+cycle() : void
+run() : void
+swim() : void
+compete(x : Game) : void
+SuperAthlete(stringedPpt : String)
Game
-ID : String
-name : String
-official : Official
-athletes : Athlete[]
-hasRun : int = 0
+Game(gameStr : String, officials : Official[])
+runGame() : void
+getID() : String
+getOfficial() : Official
+setOfficial(official : Official) : void
+getName() : String
+setName(name : String) : void
+getCompetitors() : Athlete[]
+setCompetitors(competitors : Athlete[]) : void
+getHasRun() : int
+setHasRun(hasRun : int) : void
CycleGame
+CycleGame(gameStr : String, officials : Official[], athletes : Athlete[])
RunningGame
+CycleGame(gameStr : String, officials : Official[], athletes : Athlete[])
SwimGame
+SwimGame(gameStr : String, officials : Official[], athletes : Athlete[])
Driver
-athletes : Athlete[]
-officials : Official[]
-games : Game[]
+mainMenu() : void
+initializeAthletes() : void
+initializeOfficials() : void
+initializeGames() : void
+sortAthletesByPoints() : void
Reader
-lines : int
+readFile(path : String) : String[]
Menu
-selectedGame : Game
-prediction : Athlete
-initialized : int = 0
+optionSelect() : int
+displayError() : void
+showMainMenu() : void
+setSelectedGame(game : Game) : void
+getSelectedGame() : Game
+getPrediction() : Athlete
+setPrediction(prediction : Athlete) : void
+showGameMenu(games : Game[]) : void
+selectGame(games : Game[]) : void
+showPredictionMenu() : void
+showAthletePoints(athletes : Athlete[]) : void
+showPedestals(firstPlace : Athlete, secondPlace : Athlete, thirdPlace : Athlete, off : Official) : void
+noSelectedGame() : void
+showAllGames(games : Game[]) : void