# Final Project: Marathon race runners

**User Story:** A sports company has hired you to write an application that would simulate a marathon race between interesting groups of participants. They could be slow as a tortoise, as fast as a hare, and anything in between. The runners differ in their speed and how often they need to rest. Some may be slow and never rest; others may run fast but lose steam quickly and rest a lot of the time. Still others could be anything in between.

## Example operation between two runners: A Tortoise and a Hare

- Say in this case, there are only two runners. The runners differ in their speed and how often they need to rest. One of the runners, named "Tortoise," is slow but never rests. The other runner, named "Hare," is ten times as fast but rests 90% of the time.
- There is a random element to the runners' performance, so the outcome of the race is different each time the application is run.
- The race is run over a course of 1000 meters. Each time one of the runners move, the application display the runner's new position on the course. The first runner to reach 1000 wins the race.
- When one of the runners finishes the race, the application declares that runner to be the winner and the other runner concedes.

## Example specifications for N runners (*this gives you lot of hints; however, you are encouraged to come up with your own innovative design*)

- Each runner should be implemented as a separate thread using a class named ThreadRunner. The ThreadRunner class should include a constructor that accepts three parameters: a string representing the name of the runner, an int value from 1 to 100 indicating the likelihood that on any given move the runner will rest instead of run, and an int value that indicates the runners speed—that is, how many meters the runner travels in each move.
- Each runner has following information:

| RunnersName | RunnersSpeed | RestPercentage |
|---|---|---|
| Tortoise | 10 | 0 |
| Hare | 100 | 90 |
| Dog | 50 | 40 |
| Cat | 30 | 75 |

  The program accommodates N number of runners.

- You will read the above runners information from an input source.
- The run method of the ThreadRunner class should consist of a loop that repeats until the runner has reached 1000 meters. Each time through the loop, the thread should decide whether it should run or rest based on a random number and the percentage passed to the constructor (**RestPercentage**). If this random number indicates that the runner should run, the class should add the speed value passed to the constructor (**RunnersSpeed**). The run method should sleep for 100 milliseconds on each repetition of the loop.
- To determine whether a thread should run or rest, calculate a random number between 1 and 100. Then, have the thread rest if the number is less than or equal to the percentage of time that the thread rests (**RestPercentage**). Otherwise, the thread should run.

- If the run method is interrupted, it should display a message that concedes the race and quits.
- The main method of the application's main class should create as many runner threads as are read from the input source and start all of them. You will pass all the information read from input source into the threads through constructor. Use RunnersName to be name of the thread.
- This class should also include a method named finished that one of the threads can call when it finishes the race. That method should declare the thread that calls it to be the winner and should interrupt the other thread so it can concede the race.
- The finished method should provide for the possibility that the two or more threads will finish the race at almost the same time. If that happens, it should ensure that only one of the threads is declared the winner. (There are no ties!)
- The finished method in the main application class will need to know which thread called it.

## Part of the console example for default two runners. Start and end of the race between a Tortoise and a Hare

```
Welcome to the Marathon Race Runner Program

Select your data source:

1.  Derby database
2.  XML file
3.  Text file
4.  Default two runners
5.  Exit

Enter your choice: 2
Enter XML file name: FinalXMLData.xml
....
<program will run for however many runners are in FinalXMLData.xml file>
<The menu repeats after the conclusion and gives the choice again>
<User can select menu 1 to 4 to run the marathon, or 5 to quit>
<Second run might look like this:>
...
Dog : 1000
Dog : I finished!

The race is over! The Dog is the winner.

Hare: You beat me fair and square.
Rabbit: You beat me fair and square.
Cat: You beat me fair and square.

Press any key to continue . . .

Welcome to the Marathon Race Runner Program

Select your data source:

1. Derby database
2. XML file
3. Text file
4. Default two runners
5. Exit

Enter your choice: 4
```

```
Get set...Go!
Tortoise : 10
Tortoise : 20
Tortoise : 30
Tortoise : 40
Hare : 100
Tortoise : 50
Tortoise : 60
Tortoise : 70
Tortoise : 80
Hare : 200
Tortoise : 90
Tortoise : 100
.
.   (output lines omitted)
.
Hare : 500
Tortoise : 900
Tortoise : 910
Tortoise : 920
Tortoise : 930
Tortoise : 940
Tortoise : 950
Tortoise : 960
Tortoise : 970
Tortoise : 980
Tortoise : 990
Tortoise : 1000
Tortoise: I finished!

The race is over! The Tortoise is the winner.

Hare: You beat me fair and square.

Press any key to continue . . .

Welcome to the Marathon Race Runner Program

Select your data source:

1. Derby database
2. XML file
3. Text file
4. Default two runners
5. Exit

Enter your choice: 5

Thank you for using my Marathon Race Program
```

## Requirement

1. Write an object oriented Java program to fulfill above requirement. You need to identify the classes and their hierarchies. You should also have proper interfaces identified, which mandates all the required behaviors as described in the problem domain. Provide an application (user) class as well, which uses these classes to complete the user interaction. Your program needs to be idiot proof as well. Meaning, you need to validate the user data, command line arguments etc.

2. Runners Information: The runner's information will come from one of the input sources. Your program is capable of reading runners data from:
    a. A text File
    b. A XML file
    c. A derby database (rdbms), or
    d. Hard coded default data for two runners
3. You need to provide all four interfaces in the program. The user of your program is free to use either data source. When program starts, show a menu for the data source to choose from. User will select one of the choices and provide the name of the data source. You will read all the runners data from data source and run the race for that many runners. You can assume that number of runners are unknown, however, it will not exceed 5 runners.
4. In case of default runners, there are only two runners, Tortoise and Hare. You can hard code these values for them and run the race:

| **RunnersName** | **RunnersSpeed** | **RestPercentage** |
|---|---|---|
| Tortoise | 10 | 0 |
| Hare | 100 | 90 |

5. An example FinalXMLData.xml could look like this, use same element, and attribute names

```
<?xml version="1.0" encoding="UTF-8"?>
<Runners>
  <Runner Name="Tortoise">
    <RunnersMoveIncrement>10</RunnersMoveIncrement>
    <RestPercentage>0</RestPercentage>
  </Runner>
  <Runner Name="Hare">
    <RunnersMoveIncrement>100</RunnersMoveIncrement>
    <RestPercentage>90</RestPercentage>
  </Runner>
  <Runner Name="Dog">
    <RunnersMoveIncrement>50</RunnersMoveIncrement>
    <RestPercentage>70</RestPercentage>
  </Runner>
  <Runner Name="Cat">
    <RunnersMoveIncrement>30</RunnersMoveIncrement>
    <RestPercentage>75</RestPercentage>
  </Runner>
</Runners>
```

6. An example FinalTextData.txt could look like this:

| | | |
|---|---|---|
| Tortoise | 10 | 0 |
| Hare | 100 | 90 |
| Dog | 50 | 40 |
| Cat | 30 | 75 |

7. Create a suitable RunnersDB Derby database as well. The database has only one table RunnersStats with three fields: Name (VARCHAR(20)), RunnersSpeed (DOUBLE), RestPercentage(DOUBLE). Write a JDBC interface to read the data from this database.
8. Write JavaDoc comments for all of your interfaces, classes, and methods. Generate a JavaDoc as well.

9. You should name your Java project <YourName>_JavaCompFinal. For me the name of the project will be *BineetSharma_JavaCompFinal*.
10. Create an Eclipse zip file following the document **How To Package Your Final.pdf**
11. Name the Eclipse zip file as <YourName>_JavaCompFinal_EclipseProject.zip. This zip should include all of your source code, and java doc so that I can easily create an Eclipse project while grading. For me the name of this file will be *BineetSharma_JavaCompFinal_EclipseProject.zip.*
12. Before submitting please make sure that your submission works, by importing yourself and running the application. You could rename your old project and then import it for the testing purposes as Eclipse will not import two projects with same name.
13. Submit the project in the assignment section in your web portal and attach the zip file
14. Only submit <YourName>_JavaCompFinal_EclipseProject.zip as an attachment in the assignment section when you submit your final

## Score

Maximum score you can receive for this final project is 100, and it is divided into the following categories:

1) Program compiles, runs, and provides the output in the format as shown without errors. The program runs at least for default two runners (70%)
2) The application design follows OOP concepts covered in the class with data validation, exception handling etc (10%)
3) You have written well-commented and formatted program. JavaDoc is meaningful. (10%)
4) The racers information can also be read from XML file. (5%)
5) The racers information can also be read from a Derby database (5%)