

Assignment 2 – Fall 2020

Due Date: by Wednesday September 30, 2020 11:59PM

How to submit: upload JAVA files to Blackboard

The Tortoise and the Hare, a race simulation.

In this problem, you'll recreate a version of the classic race of the tortoise and the hare. You'll use a random number generator to simulate the race. The contenders begin race along a track of 100 squares. Each square represents a possible position along the race course. The finish line is at square 100. The first contender to reach or pass square 100 wins. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground. A clock ticks once per second. With each tick of the clock, your application should adjust the position of the animals according to the rules in Figure 8. Use variables to keep track of the positions of the animals (i.e., position numbers are 1–100). Start each animal at position 1 (*the "starting gate"*). If an animal slips before square 1, move it back to square 1.

Note:

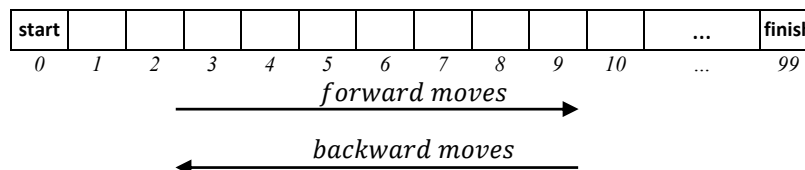
- ✓ *This is an individual assignment; please do your own work, sharing and/or copying code and/or solution ideas with/from others will result in a grade of 0 and disciplinary actions for all involved parties. If you run into problems and have done your best to solve them, please contact me before/after class or by e-mail.*
- ✓ *A 20% grade deduction for every day the assignment is late.*

Racer	Move Type	% of time	Direction	Squares to move and direction
Tortoise	Sleep	10%	–	No movement
	Jump	40%	Forward	Random # of Squares between 1 and 3
	Slip	30%	Backwards	Random # of Squares between 1 and 6
	Walk	20%	Forward	Random # of Squares between 0 and 1
Hare	Sleep	10%	–	No movement
	Jump	30%	Forward	Random # of Squares between 1 and 5
	Small slip	20%	Backwards	Random # of Squares between 1 and 2
	Big Slip	10%	Backwards	Random # of Squares between 1 and 7
	Walk	30%	Forward	Random # of Squares between 0 and 1

Figure 1: Rules for adjusting the positions of the Tortoise and the Hare.

Hint:

- ✓ Generate the percentages in Figure 1 by producing a random integer i in the range $0 \leq i \leq 9$. For the Tortoise, perform a "Jump" when $0 \leq i \leq 3$, a "slip" when $4 \leq i \leq 6$, a "walk" when $7 \leq i \leq 8$, or a sleep when $i = 9$.
- ✓ Since you're using random numbers, outputs will be different.
- ✓ To avoid long-running simulations, stop the simulation if it goes over 200 iterations.
- ✓ A unit of time is one full iteration in the program. i.e. a counter shows how many iterations were made until one contestant wins. If the simulation takes 20 iterations, then this is treated as 20 seconds.
- ✓ Think of the 100-square track as shown below. A forward move means, move up the track, a backward move means move down the track.





What to submit:

- ✓ One Java class source file for class *TortoiseAndHare* with the following minimum members (UML diagram Figure 8):
 1. At least one use of:
 - The WHILE loop
 - The FOR loop
 - The SWITCH decision structure
 2. *MAX_MOVES*, a final, and static field with value 100
 3. *main*, starts the race by creating an instance of class *TortoiseAndHare*.
 4. Default constructor:
 - Prints the start message in Figure 2
 - Print the racer's position using the method *printPositions*
 - Move both players using the methods *simulateTortoiseMove()* and *simulateHareMove()*
 - Repeat step 3 until either one wins, there is a tie, or the race times out (i.e. maximum of 200 simulations).
 - Print the results of the simulation. The result should state if:
 1. The Tortoise wins
 2. The Hare wins
 3. There is a tie
 4. There is a timeout. Also indicate who won.
 5. *randomBetween*, returns an integer random number between two limits (inclusive)
 6. *printPositions*, prints the race track and shows the position of the Tortoise and the Hare. Use the Letter 'T' to represent the Tortoise, 'H' for the Hare, and 'B' if both are on the same square. See Figure 3
 7. *simulateTortoiseMove*, a function which simulates the movements of the Tortoise as shown in Figure 1.
 8. *simulateHareMove*, a function which simulates the movements of the Hare as shown in Figure 1.

Grading:

Item	Points
Comments (Javadoc and major steps)	10
Static members	10
WHILE loop	5
FOR loop	5
Switch	5
Race simulation until one wins, tie, or timeout	10
Boundary checks	5
Constructor	5
<i>randomBetween</i>	10
<i>printPositions</i>	10
<i>simulateTortoiseMove</i>	10
<i>simulateHareMove</i>	10
Correct output	5
	100

Figures:

ON YOUR MARK, GET SET
BANG !!!!!
AND THEY'RE OFF !!!!!

Figure 2: initial message printed when the race starts

```
Iteration: 0
B
-----
Iteration: 1
H T
-----
Iteration: 2
H T
-----
Iteration: 3
B
-----
Iteration: 4
H T
-----
Iteration: 5
H T
-----
Iteration: 6
H T
-----
```

Figure 3: With every iteration, the racer's position is printed. Positions differ with every iteration and each simulation.

TORTOISE WINS!!! YAY!!!
Time Elapsed = 766 seconds

Figure 4: Message if the Tortoise wins. Time differs for each simulation.

Hare wins. Yuch!
Time Elapsed = 305 seconds

Figure 5: Message if the Hare wins. Time differs for each simulation.

It's a tie
Time Elapsed = 527 seconds

Figure 6: Message if a tie occurs. Time differs for each simulation.

Time Out!
Hare wins. Yuch!
Time Elapsed = 2000 seconds

Figure 7: Message if a timeout occurs after 2000 iterations (seconds). The winner message is also printed.

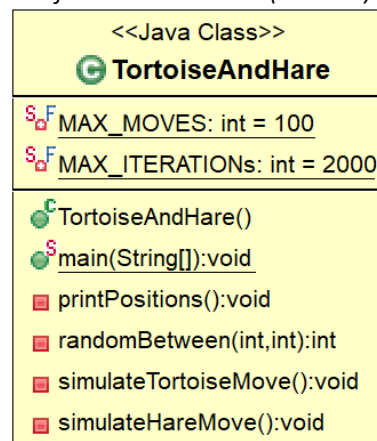














Figure 8: Class Diagram – UML legend shown below

UML Diagram Legend

Symbol	Description
<u> </u>	Indicates a static member
	A private member (i.e. variable or method)
	A private final member (i.e. variable)
	A public field (i.e. variable or method)
	A public abstract member (i.e. variable or method)
	A public constructor
	A static public member
	An interface
	A public class
	A public abstract class
	A hollowed arrow indicates inheritance
	An open-ended arrow indicates composition
	A dotted line and hollowed arrow indicate class implementation