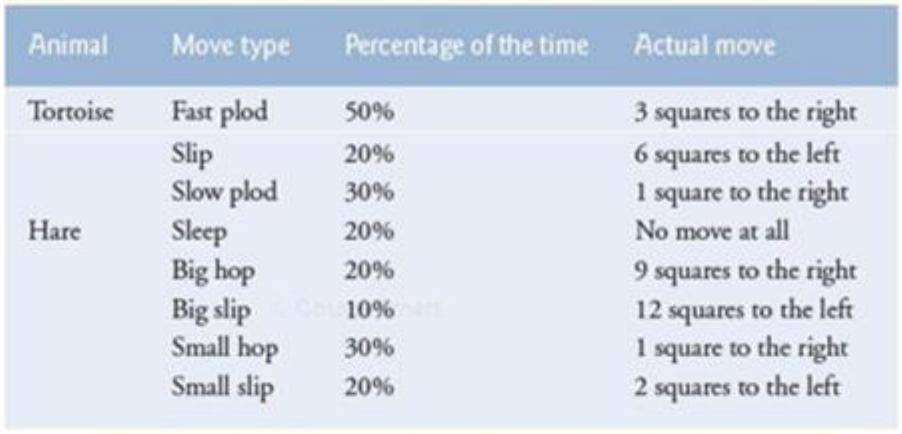
CSc221

Software Design Laboratory

Update History:

Programming Assignment #2

(Simulation: The Tortoise and the Hare) In this problem, you'll re-create the classic race of the tortoise and the hare. You'll use random-number generation to develop a simulation of this memorable event. Our contenders begin the race at square 1 of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. 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 Fig. 7.32. Use variables to keep track of the positions of the animals (i. e., position numbers are 1—70). Start each animal at position 1 (the "starting gate"). If an animal slips left before square 1, move it back to square 1.



Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Rules for adjusting the positions of the tortoise and the hare.

Generate the percentages in Fig by producing a random integer i in the range $1 \leq i \leq 10$. For the tortoise, perform a "fast plod" when $1 \leq i \leq 5$, a "slip" when $6 \leq i \leq 7$ or a "slow plod" when $8 \leq i \leq 10$. Use a similar technique to move the hare. Begin the race by displaying

```
BANG !!!!!
AND THEY'RE OFF !!!!!
```

Then, for each tick of the clock (i. e., each repetition of a loop), display a 70-position line showing the letter `T` in the position of the tortoise and the letter `H` in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your application should display `OUCH!!!` beginning at that position. All output positions other than the `T`, the `H` or the `OUCH!!!` (in case of a tie) should be blank. After each line is displayed, test for whether either animal has reached or passed square 70. If so, display the winner and terminate the simulation. If the tortoise wins, display `TORTOISE WINS!!! YAY!!!` If the hare wins, display `Hare wins. Yuch`. If both animals win on the same tick of the clock, you may want to favor the tortoise (the "underdog"), or you may want to display `It's a tie`. If neither animal wins, perform the loop again to simulate the next tick of the clock.

When you're ready to run your application, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets! Later in the book, we introduce a number of Java capabilities, such as graphics, images, animation, sound and multithreading. As you study those features, you might enjoy enhancing your tortoise-and-hare contest simulation.

Requirements

- Properly exhibits right logic, i.e., readable and compliable coding
- Properly modularized with helper functions
- Properly implements moves of tortoise and hare
- Properly writes out racing activities
- Properly restarts a new race inputted by an elected number of squares (default 70)