ELEC 2543 Object-Oriented Programming and Data Structures

Programming Exercise 3a

Topics: OO Design, Data Visibility

Due Date: Feb 13, 2017

Overview: In this exercise, you are going to develop a simple random walk game. Different types of entities are represented using different classes. You have to apply the appropriate visibility modifiers for all variables and methods.

The Random Walk Game

*Overview*

There are two players (P1 and P2) in the random walk game. They both walk on a linear path spanning from left to right. The game works in rounds. In each round, one player, selected at random, can move on the path. The game ends when both players are at the same position on the path.

*Details*

The path spans from 1 to *size*. In the beginning, P1 is at position 1, while P2 is at position *size*. In each round:

1. A map showing the current positions of the players is printed
2. a player is selected to move
3. the direction (left or right) to move is selected at random (Only one direction is possible if the player is at the path boundary.)
4. the distance to move is selected from [1, 3] (if the path boundary is hit, the distance to move is reduced to the maximum allowed.)
5. a message showing the move details is printed out
6. update the new position of the player who moves
7. game ends if both payers are now in the same position

*Sample Output*

1........2 [Player 2 moves to left by 3 unit(s)]

1.....2... [Player 1 moves to right by 2 unit(s)]

..1...2... [Player 1 moves to left by 1 unit(s)]

.1....2... [Player 1 moves to right by 2 unit(s)]

...1..2... [Player 2 moves to left by 1 unit(s)]

...1.2.... [Player 2 moves to left by 2 unit(s)]

Player 1 and Player 2 meet each other at position 4

The above is a sample output when *size* = 10. The first map shows that P1 is at the left end of the path while P2 is at the right end of the path. The message in square brackets tells the next move. For example, the fist move of the game is Player 2 moves to left by 3 units. The map on the next line tells the positions of the two players after this move.

In the last line of the map, P1 is 2 units to the left of P2. P2 now moves to the left by 2 units. The result is both P1 and P2 are in the same position. The game ends, and a message is printed to indicate the players’ final position.

Implementation

*Class Files*

[Step-by-step guide on implementation is provided after explaining the class files. You are recommended to read the whole description before coding. You do not have to follow the guide if you do not want to. We will test your final submission only.]

In Object-Oriented programming, each type of entities is modeled through a class. In this game, the necessary entities are: the game itself and the players. It would be desirable to use a rolling die to generate random numbers. A driver program is also needed. Therefore, there are four class files. Define the appropriate instance variables and methods as described below. **You have to put in appropriate visibility modifiers (private or public) for each instance variable and method**, even the method/variable has been defined for you.

Class Player: This class represents a player in the game. You should define the following:

* Define an instance variable representing the current position of the player. You cannot define other instance variable.
* Define a constructor that takes the initial position as the parameter.
* Define getter and setter method for the instance variable. The getter method **must be named getPosition()**.
* Other methods can be defined if needed.

Class RandomWalkDie: This class represents the dies to be used in the game for selecting direction, player, and distance to move. Two instance variables have been defined. max represents the maximum face value this die can have. That is, if max = 3, the roll() method should return a number from 1 to 3. This number will be passed as a parameter when creating a new die. faceValue is the current die face value. Further enhance the class as follows:

* Use an appropriate visibility modifier for the two instance variables
* Define an appropriate constructor and roll() method that new RandomWalkDie(3) creates a die that returns a random integer from 1 to 3 when the die is rolled.
* No other instance variables can be defined.
* You can define other methods if needed.

Class RandomWalkGame: This class represents one complete game that starts with the players are put in their initial positions, then a player makes a move each round until both players are in the same position.

Three instance variables have been declared for the size of the path and the players. You cannot define other instance variables. Method void printMap() has been provided to print a map showing the current locations of the two players. Note that if both players are in the same position, it shows an ‘X’ in their common position. Work on the following:

* Define the constructor such that the size of the path is passed as parameter
* Use the appropriate visibility modifier for the instance variables and method void printMap()
* Implement method void play() that simulates the game. You have to use the appropriate RandomWalkDie objects for identifying the player, the direction, and the distance to make a move.

*Phase 1: Player and Game initialization*

1. Define the instance variables and methods as described for class Player
2. Define the constructor of class RandomWalkGame. Create the appropriate Player objects in the constructor.
3. In method play() of RandomWalkGame, in the do-while loop, after printing out the map, set the positions of both players to be the same. This makes sure your loop will terminate.
4. Test your implementation by running RandomWalkGameDriver.java. Check whether the correct output can be produced. The first map should be the same as the sample output.

*Phase 2: Die Preparation and Generate move details*

1. Develop the class definition of RandomWalkDie as described above
2. In method play(), create two RandomWalkDie objects, one for selecting players and directions, one for generate the distance to move.
3. Use the two RandomWalkDie objects to identify the player, direction, and distance to move. Print out the message about the move as shown in the sample output. [Move validation is in the next phase.]
4. Test your program [do not undo Step 3 of Phase 1]

*Phase 3: Validate a move and Make a move*

Not every move generated from the random dies is feasible. For example, if a player is at position 1, moving to the left is not feasible. To validate a move and make a move, answer the following:

How to determine the updated position if a player makes a move?

What positions a player will be for invalid moves?

After you have answered the two questions, in method play(), in each iteration of the do-while loop, select one player to make a valid move. Make sure you update the player’s position.

Remove the statement you put for Step 3 of Phase 1 and test your program.

Handin

Submit your Player.java, RandomWalkDie.java, and RandomWalkGame.java (do not zip them together) to Moodle before the deadline. We will use our own driver program to test your submission.