FRC Team 4930 2015-2016 Programming Sub-Team Tryouts

Welcome to the 2015-2016 Programming Sub-Team Tryouts. There are three parts. You must attempt all three to qualify. Part One and Part Two are to be done solo. You may choose to team up in groups of two to complete part Three. I will only accept answers that are publicly available on GitHub. All three parts are to be added to a single GitHub repo.

Part One

This is a solo assignment. Commit your prepared 300 word essay to a GitHub repo which answers the following question:

"Why do you want to be a part of Team 4930's programming sub-team?"

Two Part

This is a solo assignment. Using Eclipse, write a program in Java that inputs from the user four integers representing the numerators and denominators of two fractions, calculate the results of the two fractions and display the values of the fractions sum, subtraction, multiplication and division each on a separate line. Your solution must be exported as a Jar file and executable via the terminal:

```
> java -jar SOLUTION.jar # # # #
```

Upload your solution to GitHub. Include both your code and the exported Jar file. Hint: This is very similar to the YouTube video I posted available at https://www.youtube.com/watch?v=QvwWnKhsi0s

Part Three

You may team up in groups of two to complete part three. Submit your answer as pseudo code with comments on GitHub. Only one member of your team needs to upload the solution to GitHub. The first line of your solution should be a comment which lists the members of your team.

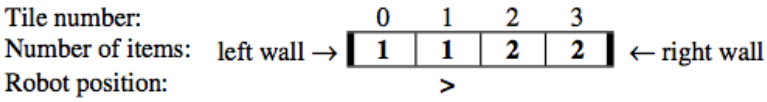
The PR2004 is a robot that automatically gathers toys and other items scattered in a tiled hallway. A tiled hallway has a wall at each end and consists of a single row of tiles, each with some number of items to be gathered.

The PR2004 robot is initialized with a starting position and an array that contains the number of items on each tile. Initially the robot is facing right, meaning that it is facing toward higher-numbered tiles.

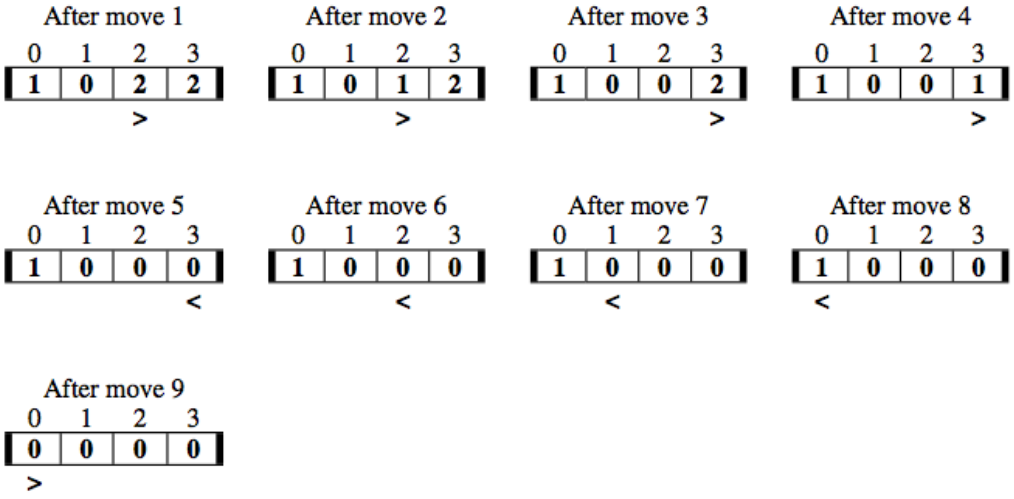
The PR2004 robot makes a sequence of moves until there are no items remaining on any tile. A move is defined as follows.

1. If there are any items on the current tile, then one item is removed.
2. If there are more items on the current tile, then the robot remains on the current tile facing the same direction.
3. If there are no more items on the current tile
   a) if the robot can move forward, it advances to the next tile in the direction that it is facing;
   b) otherwise, if the robot cannot move forward, it reverses direction and does not change position.

In the following example, the position and direction of the robot are indicated by "<" or ">" and the entries in the diagram indicate the number of items to be gathered on each tile. There are four tiles in this hallway. The starting state of the robot is illustrated in the following diagram.



The following sequence shows the configuration of the hallway and the robot after each move.



After nine moves, the robot stops because the hall is clear.

The PR2004 is modeled by the class `Robot` as shown in the following declaration.

```
public class Robot
{
  private int[] hall;
  private int pos;              // current position(tile number) of Robot
  private boolean facingRight; // true means this Robot is facing right


  // constructor not shown


  // postcondition: returns true if this Robot has a wall immediately in
  //                front of it, so that it cannot move forward;
  //                otherwise, returns false
  private boolean forwardMoveBlocked()
  {  /* to be implemented in part (a) */  }


  // postcondition: one move has been made according to the
  //                specifications above and the state of this
  //                Robot has been updated
  private void move()
  {  /* to be implemented in part (b) */  }


  // postcondition: no more items remain in the hallway;
  //                returns the number of moves made
  public int clearHall()
  {  /* to be implemented in part (c) */  }


  // postcondition: returns true if the hallway contains no items;
  //                otherwise, returns false
  private boolean hallIsClear()
  {  /* implementation not shown */  }
}
```

In the `Robot` class, the number of items on each tile in the hall is stored in the corresponding entry in the array `hall`. The current position is stored in the instance variable `pos`. The `boolean` instance variable `facingRight` is `true` if the `Robot` is facing to the right and is `false` otherwise.

(a) Write the `Robot` method `forwardMoveBlocked`. Method `forwardMoveBlocked` returns `true` if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, `forwardMoveBlocked` returns `false`.

Complete method `forwardMoveBlocked` below.

```
// postcondition: returns true if this Robot has a wall immediately in
//                front of it, so that it cannot move forward;
//                otherwise, returns false
private boolean forwardMoveBlocked()
```

(b) Write the `Robot` method `move`. Method `move` has the robot carry out one move as specified at the beginning of the question. The specification for a move is repeated here for your convenience.

1. If there are any items on the current tile, then one item is removed.
2. If there are more items on the current tile, then the robot remains on the current tile facing the same direction.
3. If there are no more items on the current tile
   a) if the robot can move forward, it advances to the next tile in the direction that it is facing;
   b) otherwise, if the robot cannot move forward, it reverses direction and does not change position.

In writing `move`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `move` below.

```
// postcondition: one move has been made according to the
//                specifications above and the state of this
//                Robot has been updated
private void move()
```

(c) Write the `Robot` method `clearHall`. Method `clearHall` clears the hallway, repeatedly having this robot make a move until the hallway has no items, and returns the number of moves made.

In the example at the beginning of this problem, `clearHall` would take the robot through the moves shown and return 9, leaving the robot in the state shown in the final diagram.

In writing `clearHall`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `clearHall` below.

```
// postcondition: no more items remain in the hallway;
//                returns the number of moves made
public int clearHall()
```