Noah Streveler

Assignment 5

```
This program lets you play a Racing Game
Play? (t/f)?t
How many people will be playing (2-6)?4

        The Track Length is 50
                Player  1       2       3       4

Round 1: Positions      0       0       0       0
Round 2: Positions      12      9       18      1
Round 3: Positions      17      19      23      13
Round 4: Positions      20      15      26      5
Round 5: Positions      8       25      32      22
Round 6: Positions      4       31      38      13
Round 7: Positions      0       47      15      35
Round 8: Positions      27      27      16      41
Round 9: Positions      16      19      13      44
Round 10: Positions     12      12      16      56
Congratulations to Player 4 for Winning!
Play again (t/f)?f
```

```
import java.util.ArrayList;
import java.util.Collections;

public class Racer {
    private ArrayList<Integer> race;   // The cards in the hand.

    public Racer() {//Temporal Cohesion- Initializing variables
        race = new ArrayList<Integer>();
    }
    //Functional-clear and defined method
    public void clear() {
        race.clear();
    }
    //Functional-clear and defined method
    public void add(int number) {
```

```
        race.add(number);
    }
    public void remove(int index) { //Functional-clear and defined method
        if (index < 0 || index >= race.size())
            throw new IllegalArgumentException("Position does not exist: "
                    + index);
        race.remove(index);
    }

    public int getCount() { //Logical-Action performed by calling input
        return race.size();
    }

    public int getTop() {//Logical-Action performed by calling input
       return race.get(getCount() - 1);
    }

    public int getIndex(int index) {//Logical-Action performed by calling input
        if (index < 0 || index >= race.size())
            throw new IllegalArgumentException("Position does not exist: " + index);
        return race.get(index);
    }

    public String displayRacer(Racer group) {//Logical-Action performed by calling
input
             String str = "";
       for(int i = 0; i < group.getCount(); i++){
                    str += group.getIndex(i) + "\t";
             }
       return str;
    }
```

```
	public boolean isEmpty() {//Functional-clear and defined method
		return race.size() == 0;
	}
}//end of class
```

```
import java.util.ArrayList;
import java.util.List;

public abstract class GameControl {
	protected IGameView view = new IOHandler();
	protected Racer[] positions;
	protected int[] moveType;
	protected int[] timesLeft;
	protected int numPlayers;
	protected int winAmount;

	public GameControl(){//Temporal Cohesion- Initializing variables
		init();
		numPlayers = 0;
		winAmount = 50;
	}
	public void runGame(){//Functional Cohesion- All parts inside are essential to
the output
		Character input = view.getInput("Play? (t/f)" + "?");
		if( input != 't') return;
	    do {
```

```
        numPlayers = numOfPlayers();
        positions = new Racer[numPlayers];
        moveType = new int[numPlayers];
        timesLeft = new int[numPlayers];
            for(int i = 0; i < positions.length; i++) {
               positions[i] = new Racer(); //Instantiate
               }
               startGame();
        do{
               view.display(playersPosition());
               playRound();
        }while(isWinner() == -1);
        view.display(playersPosition());
        endGame();
    } while ( ((char) view.getInput("\nPlay again (t/f)" + "?")) == 't');
}
abstract void init();
abstract int numOfPlayers();
abstract void startGame();
abstract void playRound();
abstract void endGame();
abstract int isWinner();
abstract int findLeader();
abstract int findLast();
abstract int moves(int number);
//Logical-Action performed by calling input
public String playersPosition(){
        String str = "\nRound " + positions[0].getCount() + ": Positions ";
        for(int i = 0; i < numPlayers; i++){
               str +=  "\t" + positions[i].getTop();
        }
```

```
		return str;
	}
	//Logical-Action performed by calling input
	public int diceRoll() {//Rolling the dice
		int dice = ((int)(Math.random() * 6 + 1));
		return dice;
	}
	//Logical-Action performed by calling input
	public int moveType() {//Random number between 1-3
		int random = ((int)(Math.random() * 3 + 1));
		return random;
	}
	//Logical-Action performed by calling input
	public int duration() {//Random number between 1-5
		int random = ((int)(Math.random() * 5 + 1));
		return random;
	}
}

interface IGameView{//Functional-clear and defined method
	void display(String message);
	<T> T getInput(String msg);
}




import java.util.*;
import java.time.*;
import javax.swing.JOptionPane;
```

```
public class RaceGame {

   public static void main(String[] args) {//Functional Cohesion- All parts inside
are essential to the output

         GameControl controller= new GameController();

         controller.runGame();

   }

}


class IOHandler implements IGameView{

       Scanner  sc = new Scanner(System.in);

       char input;

       private static char[] matches = new char[]{'f', 't'};


       @Override

       public void display(String message) {//Functional Cohesion- All parts inside
are essential to the output

              System.out.print(message);

       }


       @Override

       public Character getInput(String msg) {

              boolean isCorrectInput = false;

              do {

                     System.out.print(msg);

                     input = sc.next().charAt(0);

             input = Character.toLowerCase(input);

             for(int i = 0; i < matches.length; i++){

              if (input == matches[i]) {

                     return new Character(input);

              }

             }

             System.out.print("Please respond with an expected character:  ");
```

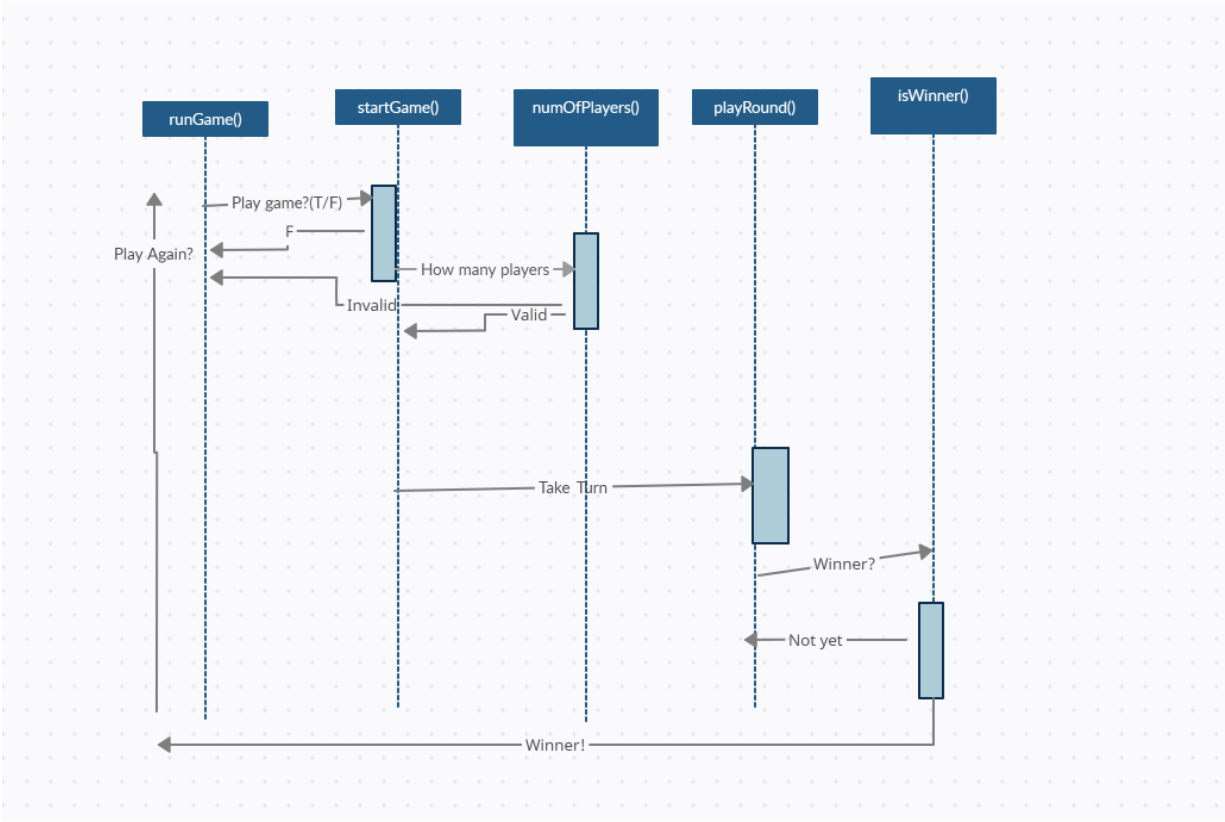
```
        } while (!isCorrectInput);
            return null;
        }
}

class GameController extends GameControl{  //game model + game control
        Racer current = null;
        int currPlay = 0;
        private Scanner sc;
        public GameController(){//Temporal Cohesion- Initializing variables
            super();
            sc = new Scanner(System.in);
        }

        @Override
        public void init(){//Logical-Action performed by calling input
            view.display("This program lets you play a Racing Game\n");
        }

        @Override
        public void startGame(){//Procedural Cohesion - The elements are grouped
together to complete a task
            view.display("\n\tThe Track Length is " + winAmount);
            view.display("\n\t\tPlayer\t");
            for(int i = 0; i < positions.length; i++) {
                view.display((i+1) + "\t");
                positions[i].add(0);
                moveType[i] = moveType();
                timesLeft[i] = duration();
            }
            view.display("\n");
        }
```

```
	@Override
	public void playRound() {//Procedural Cohesion - The elements are grouped together to complete a task
		for(int i = 0; i < positions.length; i++){
			currPlay = i;
			int newPos = 0;
			if(timesLeft[i] == 0) {
				moveType[i] = moveType();
				timesLeft[i] = duration();
			}
			else {
				 timesLeft[i]--;
			}

			newPos = (moves(moveType[i]) + positions[i].getTop());

			if(newPos < 0) {
				 positions[i].add(0);
			}
			else {
				 positions[i].add(newPos);
			}
		}
	}

	@Override
	public void endGame() {//Procedural Cohesion - The elements are grouped together to complete a task
		if(isWinner() != -1) {
			view.display("\nCongratulations to Player " + isWinner() + " for Winning!");
```

```
           }
        }

        @Override
        public int numOfPlayers() {//Functional Cohesion- All parts inside are
essential to the output
               view.display("How many people will be playing (2-6)?");
               int player = sc.nextInt();
               if(player < 2 && player > 6) {
                     System.out.println("I'm sorry, but the option that you chose is
not available, please try again");
                     System.exit(0);
               }
               return player;
        }

        @Override
        public int isWinner(){//Functional Cohesion- All parts inside are essential
to the output
              for(int i = 0; i < positions.length; i++){
                    if (positions[i].getTop() >= winAmount){
                          return (i + 1);
                   }
              }
              return -1;
        }

        @Override
        public int findLeader() {//Logical-Action performed by calling input
               int max = Integer.MIN_VALUE;
               for(int i = 0; i < positions.length; i++){
                     if(positions[i].getTop() > max)
```

```
                        max = positions[i].getTop();
                }
                return max;
        }

        @Override
        public int findLast() {//Logical-Action performed by calling input
                int min = Integer.MAX_VALUE;
                for(int i = 0; i < positions.length; i++){
                        if(positions[i].getTop() < min)
                                min = positions[i].getTop();
                }
                return min;
        }

        @Override
        public int moves(int number) {//Procedural Cohesion - The elements are
grouped together to complete a task
                int newPos = 0;
                int dice = diceRoll();
                switch (number){//Easily able to add new shapes
                case 1:
                        newPos = dice + (findLeader() - positions[currPlay].getTop())/2;
                        if(dice == 1 || dice == 2)
                                newPos = newPos - (2 * newPos);
                      break;
                case 2:
                        newPos = dice;
                        if((dice % 2) == 0)
                                newPos = 3 * newPos;
                      break;
                case 3:
```

```
                    newPos = dice + (positions[currPlay].getTop() - findLast())/2;
                     if(dice >= 3 && dice <= 6)
                           newPos = newPos - (2 * newPos);
                    break;
              }
              return newPos;
        }
}
```


runGame()
startGame()
numOfPlayers()
playRound()
isWinner()
Play game?(T/F)
F
Play Again?
How many players
Invalid
Valid
Take Turn
Winner?
Not yet
Winner!

Nath Streveler