CptS355 - Assignment 6 - Java
Spring 2017

Assigned: Tuesday April 18, 2017

Due: Wednesday April 26, 2017

Weight: Assignment 6 will count for 8% of your course grade.

Your solutions to the assignment problems are to be your own work. Refer to the course academic integrity statement in the syllabus.

This assignment provides experience with language features of Java that we haven't encountered in previous languages. The goal is to write a simple ball game where the player tries to click on the moving balls before they get out of the scene. The player will score points every time she/he hits a ball. The faster the ball is when player hits the ball, the more points she/he will score.

Turning in your assignment

All code should be developed in the `BallGame` directory. When you are done, the directory will contain your source (java) files, object (class) files, and config.JSON file. (Before you submit your assignment, please delete the *.class* files.) To submit your assignment, turn in your file by uploading on the Assignment6 (Java) DROPBOX on Blackboard (under AssignmentSubmisions menu). You may turn in your assignment up to 5 times. Only the last one submitted will be graded.

The work you turn in is to be your own personal work. You may not copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself.

Getting Started

A skeleton of the code for this assignment is provided. You can download the files in HW6_skeleton.zip. The skeleton code includes the following files:

- `Main.java` - Implements the applet interface. Creates a single ball and player in the beginning of the game. The ball is initially located in the middle of the applet screen.
- `Ball.java` - Implements the Ball class. The Ball class provides the methods to draw the ball, move the ball, check whether it was hit by a player, and whether it is out.
- `Player.java` – Implements the Player class. Each player object keeps track of the player's total score.
- `GameWindow.java` - Implements the `GameWindow` class. The borders of the applet screen are defined in this class.

Compiling and Running Your Project

To run your code on the command line:

Put all java files, .jar files for the modules you use and the `config.JSON` file in a single directory. You can then compile the program with

```
javac *.java
```

(to compile with the `json-simple` jar file download `json-simple-1.1.1.jar` file to your project directory and run the following command:)

```
javac -cp json-simple-1.1.1.jar *.java
```

and run the applet with

```
appletviewer Main.java
```

You may alternatively use an IDE (e.g., Eclipse). Instructor will show how to run the skeleton code on Eclipse in class. (You can download Eclipse at http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neon3)

Grading

The assignment will be marked for good programming style (indentation and appropriate comments), as well as clean compilation and correct function.

You will demonstrate your project to the course TA in person on Thursday Apr 27 and Friday Apr 28, in Sloan 312. The demo schedule will be available **HERE** Please select an available slot in the list. Write down your last name only. Please do not overwrite a slot which is already taken.

Help with Java

There is extensive documentation on Java on the web. Below link includes a list of Java tutorials.

The Java Language Tutorials (from Oracle - previously Sun)

Assignment Requirements

1. (15%) Parse the `config.JSON` file: In the provided skeleton code, most of the arguments to configure the game are hard coded. You need to read these configuration values from a JSON file and initialize the classes using these values. A sample `config.JSON` file is available in the skeleton code archive file.
   Here is a Java JSON Parsing example using JSON-simple:
   https://examples.javacodegeeks.com/core-java/json/java-json-parser-example/ To make this example work you need to download json-simple-1.1.1.jar and save it in your CLASSPATH.
2. (5%) The skeleton code creates a single ball only. Your game should support an arbitrary number of balls (the number of balls will be read from the config file). Your game should maintain the instances of the ball objects in an array (or Arraylist).
3. (10%) The Ball class implements a basic ball that moves with a pre-set speed. You will change the Ball class so that every time the ball is hit or the ball is out, it's speed (in x and y directions) should be randomized. Please note that the speed of the ball can be either positive or negative. When the speed is positive, the ball will move from left to right; and when the speed is negative it will move from right to left. You may assume that the ball will have a maximum absolute speed. The initial speed of the ball and its max speed will be read form the config file.
4. (10%) In the given skeleton code, the game terminates when the ball is out. You need to update the Player class and have the player start the game with a given number of lives. The player will lose a life when a ball is out and will earn additional lives for every X points he/she earns. (The

number of lives and the X value will be read from the config file.) The current number of lives for the player should be displayed on the applet screen.

5. (10%) You will collect statistics about the player's performance and store them in the Player class. Those should be displayed on the screen at the end of the game. These include:
   - # of mouse clicks (in the current game)
   - % of successful hits
   - the type of ball with most hits (see below for the ball types).
6. You will introduce new ball types in your game by creating subclasses of the Ball class.
   - (20%) *ShrinkBall*: This is a larger ball which gets smaller by 30% each time the player hits it. The scores for each hit will be doubled as the ball shrinks. After each hit, the ball will be moved to its initial location and the ball will be assigned a random speed. When the ball size is less than or equal to 30% of the initial size the ball, the ball will be reset to it's original size and it will start from the middle of the screen with a random initial speed. As long as the ball is not out, the user won't lose any lives.
   - (25%)*BounceBall* will bounce on the borders of the applet scene but it will be out after it bounces for a certain number of times. (The bounce count will be read from the config file.) The ball should maintain the magnitude of it's speed in each bounce. Please note that the direction (sign) of the speed will change due to the bounce. The other ball types won't bounce on borders. If the bounce ball goes out of the applet screen (after bouncing "bounce count" times), the user will lose a life. The BounceBall will reappear at its initial starting location and will be assigned a random speed.
7. (5%) Visual features and overall game design.

Sample `config.JSON` file

```
{
    "GameWindow": {
      "x_leftout": "10",
      "x_rightout": "1000",
      "y_upout": "45",
      "y_downout": "800"
    },
    "Player": {
      "numLives": "10",
      "score2EarnLife": "100"
    },
    "numBalls": "3",
    "Ball": [
        {
          "id": "1",
          "type": "basicball",
          "radius": "15",
          "initXpos": "500",
```

```
          "initYpos": "400",
          "speedX": "1",
          "speedY": "-1",
          "maxBallSpeed": "2",
          "color": ["255","0","0"]
        },
        {
          "id": "2",
          "type": "bounceball",
          "radius": "20",
          "initXpos": "300",
          "initYpos": "350",
          "speedX": "1",
          "speedY": "1",
          "maxBallSpeed": "3",
          "color": ["0","0","255"],
          "bounceCount": "10"
        },
        {
          "id": "3",
          "type": "shrinkball",
          "radius": "40",
          "initXpos": "600",
          "initYpos": "450",
          "speedX": "1",
          "speedY": "-1",
          "maxBallSpeed": "2",
          "color": ["0","255","0"],
                   "shrinkRate" : "30"
        }
      ]
}
```