

# CptS355 - Assignment 6 (Java)

Spring 2024

**Assigned:** Monday 15 April 2024

**Due:** Friday 26 April 2024 @ 10:00pm

**Weight:** Assignment-6 will count for **10%** 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 have not 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 exit the scene. The player will score points every time they hit a ball. The more balls that are hit before they leave play, the more points the player 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 and object (`.class`) files. (Before you submit your assignment, please delete the `.class` files. Only submit the `.java` files) To submit your assignment, turn in a [zip file](#) of your `BallGame` directory by uploading to the Assignment6 (Java) area on Canvas.

The work you turn in is to be your own personal work. You may not copy another student's code, nor work together on writing code. You may not copy code from the web or do anything else that lets you avoid solving the problems for yourself.

## Getting Started

Skeleton code for this assignment (`BallGame.zip`) is provided on Canvas in the assignment entry. The skeleton code includes the following files:

- `BallGame.java` – Implements the [main](#) function. Creates a single basic ball in the beginning of the game. The ball is initially located in the middle of the screen.
- `BasicBall.java` – Implements the `BasicBall` class. The `BasicBall` class provides the methods to [draw](#) the ball, [move](#) the ball, check whether it was [hit](#) by a player, and whether it is [out](#).
- `StdDraw.java`, `StdIn.java`, `StdOut.java`, `StdRandom.java` – Standard Java libraries. **Do not** update these files.

# Compiling and Running Your Project

Follow instructions from lecture () to install JDK of your choice if needed.

On Windows: Before you run your code on command line, make sure that Windows can find the Java compiler and interpreter by adding the Java installation directory to the Windows 'Path' environment variable.

(Go to Computer -> System Properties -> Advanced system settings -> Environment Variables -> System variables and Edit Path variable under System Variables. Add the Java installation path (e.g. C:\Program Files\Java\jdk-16\bin ) to the 'Path' variable. You can check what version of java you are running with the command "java -version".

## To run your code at the command line:

In command line, browse to the BallGame directory and compile the program with the following command:

```
javac *.java
```

and run the game :

```
java BallGame 1 basic 0.08
```

(1: **number** of balls , basic: ball **type**, 0.08 ball **radius**)

Your final game should support an **arbitrary number of balls**. For each ball, the **type** and **radius** should be specified in program arguments. See Assignment Requirements for more details.

You may alternatively use an IDE (e.g., Eclipse).

# Grading

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

## Help with Java

There is extensive documentation for Java on the web. The below link includes a list of Java tutorials.

[The Java Language Tutorials \(from Oracle - previously Sun\)](#)

# Assignment Requirements

## 1. (10%) The skeleton code creates a single basic ball only.

Your game should support an [arbitrary number](#) of balls (the number of balls will be provided in the command line). Your game should maintain the instances of the ball objects in an `ArrayList`.

## 2. (2%) The Ball class implements a basic ball that moves with random speed.

You will change the `BasicBall` class so that every time the ball is hit, its [speed](#) (in x and y directions) should be [randomized](#). Please note that when the speed of the ball is positive, the ball will move from left to right; and when the speed is negative it will move from right to left. The skeleton code assumes that the [max absolute speed](#) of the ball is 0.01.

## 3. In the given skeleton code, the game terminates when the ball is out.

You need to keep track of the number of the active balls in the game as new balls are [introduced](#) and as balls [exit](#) the screen (in the skeleton code the variable `numBallsinGame` maintains the number of active balls). When all the balls are out of play, the game should [terminate](#).

## 4. (15%) You will define a class called Player which will store all player information and player's game scores.

These include:

- current player score (note that the hits to different ball types are worth different number of points: [basic → 25](#); [shrink → 20](#); [bounce → 15](#); [split → 10](#)).
- each type of ball with number of hits ([see below for the ball types](#)).

The [game window](#) should display at least the current number of balls and total scoring during game play. An end-of-game screen should display all the above information. See [\(3%\) Visual features and overall](#) game design.

## 5. (70%) You will introduce new ball types in your game by creating subclasses of the `BasicBall` class.

- (25%) [ShrinkBall](#): This is a larger ball which [gets smaller](#) by 33% (i.e., 2/3 of the original size) each time the player hits it. Similar to `BasicBall`, after each hit, the ball will be moved to its initial location and it will be assigned a random speed. When the ball size is less than or equal to 25% of the initial size the ball, the ball will be reset to its original size and it will

start from the middle of the screen with a random initial speed. A hit to a shrink ball will increase the player's score by [20 points](#).

- (20%) *BounceBall* will bounce on the borders of the scene but it will be out after it bounces for a certain number of times. (The [bounce count](#) will be 3 for all bounce balls.) The ball should maintain the magnitude of its speed in each bounce. Please note that the [direction \(sign\)](#) of the speed will change due to the bounce. The other ball types **will not** bounce on borders. After bouncing 3 times, the bounce ball will disappear out of the game window. Similar to other balls, after each hit, the ball will be moved to its initial location and it will be assigned a random speed. A hit to a bounce ball will increase the player's score by [15 points](#).
- (25%) *SplitBall* will split into 2 unique balls every time the ball is hit. The 2 split-balls will always appear at the center of the game window and they will have the [same](#) radius as the original ball. Their initial speeds will be [randomly](#) assigned. Each ball generated in a split is itself a split-ball and can be [split further](#) when hit with a mouse click. A hit to a split ball will increase the player's score by [10 points](#).

All properties and behavior common to all ball types should be defined in the `BasicBall` class. And the properties and behavior specific to a particular ball type should be included in the corresponding subclass. You need **to** use Object Oriented Programming features as much as possible in your application.

## 6. The following arguments will be passed to the game in the command line:

# of balls, ball type, ball radius – *ball type* and *ball radius* will be repeated [for each ball](#).

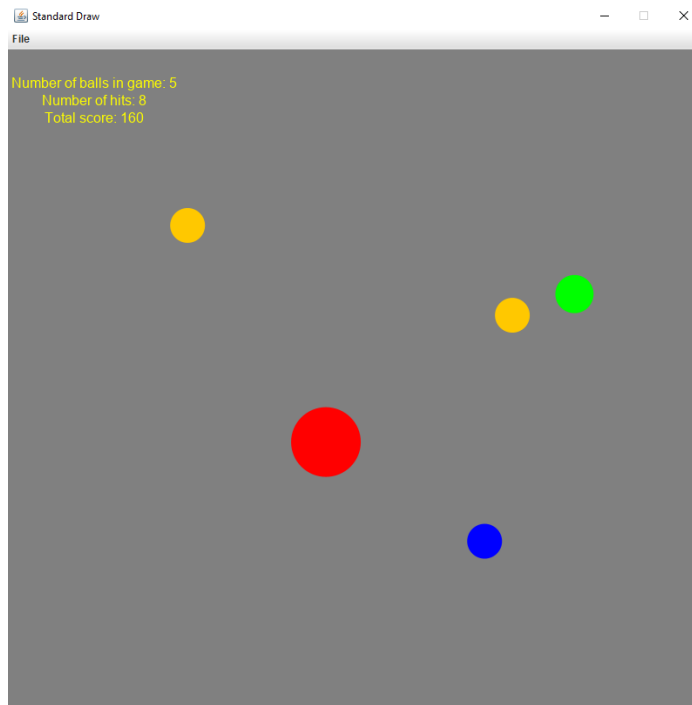
For example:

4 basic 0.10 bounce 0.05 shrink 0.13 split 0.05

The code to retrieve these arguments is already provided in the skeleton code.

## 7. (3%) Visual features and overall game design.

The following figure illustrates a snapshot of a sample game window while the game is **active**.



The following figure illustrates a snapshot of a sample game window at **end** of game.

