Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 3
Keeping Track of Objects

Introduction

This assignment is about data structures and objects, specifically the use of B-Trees to organize data in an explicit order. It extends the work done previously in Assignment 2 to include a mechanism for keeping track of all objects generated in order to i) determine when the entire set of balls has stopped moving and ii) to access each ball in order of size. You will use this new capability to generate the program described below. This assignment makes use of material from the Data Structures lectures.

Problem Description

Extend the program in Assignment 2 as follows. When the last ball stops moving (Figure 1a), determine the set of ball sizes and arrange the balls in stacks as shown in Figure 1b. Each stack is comprised of balls of approximately the same size (the precise meaning of *approximate* in this case will be formally defined a bit later). Your program should operate as follows:

1. When launched, the simulation starts and runs until the last ball stops moving. At this point it should stop and prompt the user with the message "CR to continue" . The display should appear as shown in Figure 1a.
2. When the mouse is clicked, the program proceeds with the ball sort and produces the display shown in Figure 1b, with an "All Stacked" message replacing the earlier "CR to continue."

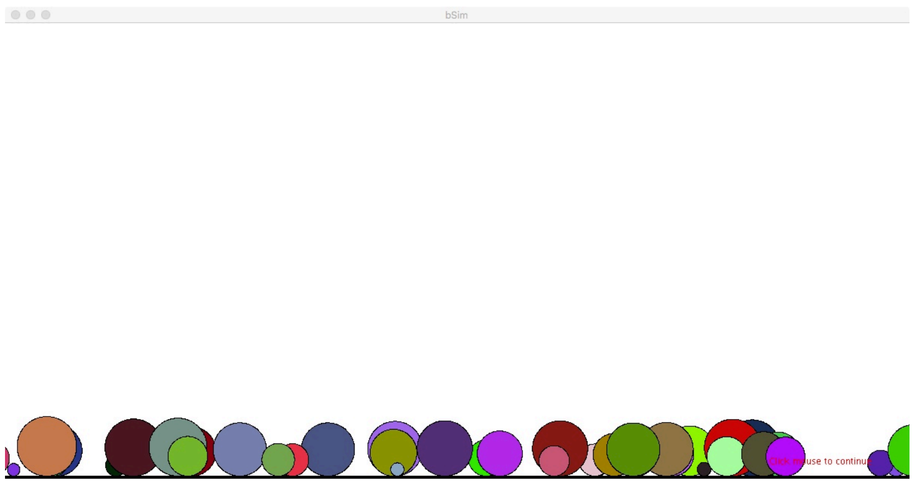

Figure 1a

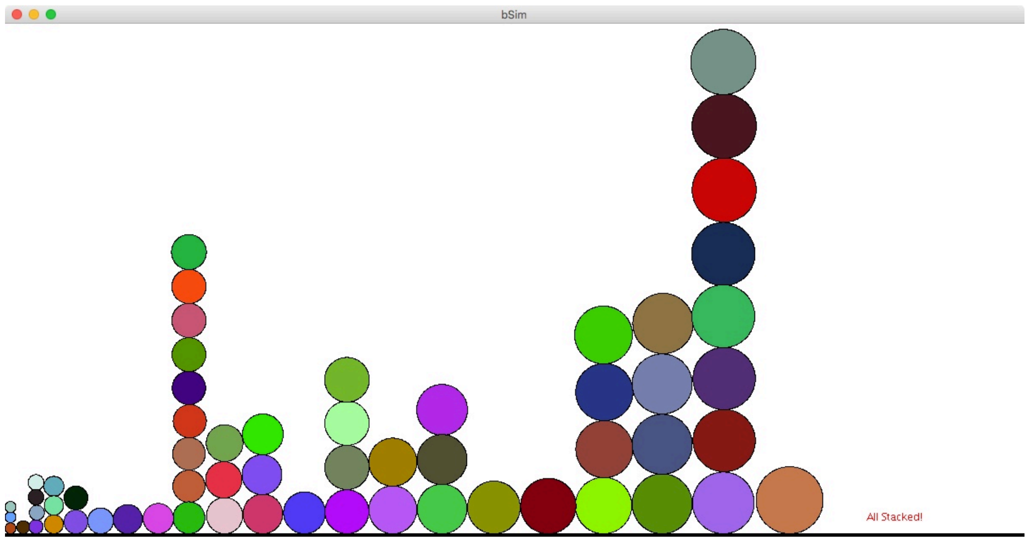

Figure 1b

As shown in Figure 1b, for the 60 balls generated in Figure 1a, there are 21 distinct size classes – i.e., a size class is defined as a set of balls of *approximately* the same size as follows:

```
1.  Given a set of balls ordered from smallest to largest.
2.  For I = 1 to Number of balls in set
3.          If current size – last size > DELTASIZE
4.                  Start a new stack
5.          Else
6.                  Put current ball on top of last ball
7.  End
```

Algorithm 1

For testing and submission of your program, please make sure to use the following parameters. Anything defined as a *double* corresponds to world (simulation) coordinates in units of meters, kilogams, seconds (MKS).

```
// Parameters used in this program

private static final int WIDTH = 1200;   // n.b. screen coordinates
private static final int HEIGHT = 600;
private static final int OFFSET = 200;
private static final double SCALE = HEIGHT/100;   // pixel/meter
private static final int NUMBALLS = 60;           // # balls
private static final double MINSIZE = 1.0;        // Min radius
private static final double MAXSIZE = 7.0;        // Max radius
private static final double EMIN = 0.2;           // Min loss
private static final double EMAX = 0.6;           // Max loss
private static final double VoMIN = 40.0;         // Min velocity
private static final double VoMAX = 50.0;         // Max velocity
private static final double ThetaMIN = 80.0;      // Min angle
private static final double ThetaMAX = 100.0;     // Max angle
```

Within the aBall class, the value of k = 0.0001. Your program should generate parameters in the same order as Assignment 2. The random number generator should have an initial seed value set by: rgen.setSeed((**long**) 424242); This should produce the same display as shown in Figure 1b.

Design Approach

The first requirement is a data structure to hold the aBall objects generated and methods for adding new data to the B-Tree, and for tree traversal. Using the bTree class code posted on myCourses is a good starting point, but you will need to modify it to store aBall objects. Consequently in the run method of your bSim class, one of the things that you need to do is create an instance of the bTree class:

```
    bTree myTree = new bTree();
```

You will have to modify the addNode method to accommodate the aBall object,

```
    void addNode(aBall iBall);      // the argument is of type aBall
```

create a new method based on the in-order traversal routine that scans the B-Tree and checks the status of each aBall,

```
    boolean isRunning();            // returns true if simulation still running
```

and finally a second new method based on the in-order traversal routine to move a ball to its sort order position instead of printing,

```
    void stackBalls(bSim link);     // this will move selected aBalls to their sorted
                                    // position
```

The stackBalls method operates by traversing the B-Tree and moving each ball to either the top of the last ball placed, or at the start of a new stack to the right (Algorithm 1). The balls should be in contact with other as shown in Figure 1b without overlap.

Here is how this code might appear in your simulation class:

```
// Set up random number generator & B-Tree

    RandomGenerator rgen = RandomGenerator.getInstance();
    bTree myTree = new bTree();
```

```
        rgen.setSeed((long) 424242);

// In the aBall generation loop

        aBall iBall = new aBall(Xi,Yi,iVel,iTheta,iSize,iColor,iLoss);
        add(iBall.myBall);
        myTree.addNode(iBall);

// Following the aBall generation loop

        while (myTree.isRunning());              // Block until termination
        Code to add a GLabel to the display      // Prompt user
        Code to wait for a mouse click           // Wait
        myTree.stackBalls(this);                 // Lay out balls in order
```

Modifications will also be needed to the aBall class. You will need to create an instance variable that indicates whether the while loop is active (that can be interrogated by the isRunning method), and a method to move a ball to a specified location, void moveTo(double x, double y), where (x,y) are in simulation coordinates.

Working Inside of a Recursion

Recall the form of a recursive tree traversal:

```
        private void traverse_inorder(bNode root) {
              if (root.left != null) traverse_inorder(root.left);
              <code to process data at the current node>
              if (root.right != null) traverse_inorder(root.right);
        }
```

Any variables declared locally inside of traverse_inorder() disappear when the current instance returns. Suppose you wanted to place each ball in succession from left to right. If X and Y represent the position of the ball, then you could **not** do this:

```
        private void traverse_inorder(bNode root) {
              double X,Y,lastSize=0;
              if (root.left != null) traverse_inorder(root.left);
              // processing for current node
              Get size of ball at current node.
              Update values of X and Y to determine where to place it.
              moveTo(X,Y) to place the ball there
              //
              if (root.right != null) traverse_inorder(root.right);
        }
```

In order to work, variables such as X, Y, lastSize, etc., need to be *persistent* across recursive calls to traverse_inorder. An easy way to do this is to make them instance variables of the bTree() class where traverse_inorder is defined. It's not hard to see how stackBalls() can be derived from traverse_inorder() with a little bit of thought.

Instructions

1. Modify the bTree class, `bTree.java`, to accommodate aBall objects and write the additional methods described earlier. For this assignment, the value of parameter DELTASIZE = 0.1.
2. Modify the aBall class, `aBall.java`, to provide a method for returning run state and moving the and placing the balls as specified.
3. Modify the bSim class, `bSim.java`, to complete the program design.
4. Before submitting your assignment, run your own tests to verify correct operation.

To Hand In:

1. The source java files. Note – use the default package.
2. A screenshot file (pdf) showing the state of the simulation at the user prompt as in Figure 1a.
3. A screenshot file showing the end of the program with the balls in sort order as in Figure 1b.

All assignments are to be submitted using myCourses (see myCourses page for ECSE 202 for details).

Note:

1. Instance variables must be defined as **private** and accessed using appropriate getter and setter methods.
2. You must use the parameters defined above.
3. Code must be appropriately commented using Javadoc convention as a minimum standard.

File Naming Conventions:

Fortunately myCourses segregates files according to student, so submit your .java files under the names that they are stored under in Eclipse, e.g., bSim.java, aBall.java, gUtil.java, bTree.java. We will build and test your code from here.

Your screenshot files should be named A3-1.pdf and A3-2.pdf respectively (again, this makes it possible for us to use scripts to manage your files automatically).

About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS, especially when the assignment is a variation from the previous semester.

https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf October 10, 2019