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, arrange each ball from left to right in size order. Figure 1a shows a configuration of 15 balls just as the last ball stops moving, and Figure 1b shows the result of arranging the balls in size order. 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 return key is pressed, the program proceeds with the ball sort and produces the display shown in Figure 1b. *Bonus:* Make the program fully interactive by using a GLabel to display "Click mouse to continue", the waitForClick() method for user input, and the setLabel() method to indicate "All Sorted!".

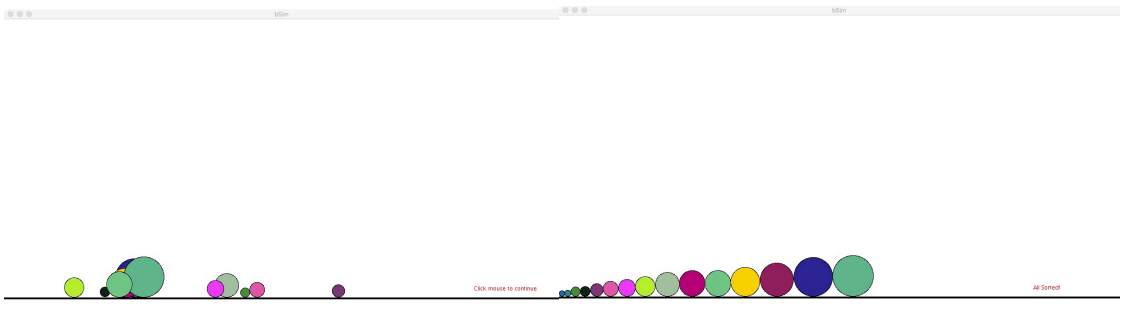

Figure 1a Figure 1b

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).

```
private static final int WIDTH = 1200;     // WIDTH, HEIGHT, and OFFSET
private static final int HEIGHT = 600;     // define the screen paramaters
private static final int OFFSET = 200;     // and represend PIXEL coordinates
private static final int NUMBALLS = 15;    // # balls to simulate
private static final double MINSIZE = 1;   // Min ball size (MKS from here down)
private static final double MAXSIZE = 8;   // Max ball size
private static final double XMIN = 10;     // Min X starting location
private static final double XMAX = 50;     // Max X starting location
private static final double YMIN = 50;     // Min Y starting location
private static final double YMAX = 100;    // Max Y starting location
private static final double EMIN = 0.4;    // Minimum loss coefficient
private static final double EMAX = 0.9;    // Maximum loss coefficient
private static final double VMIN = 0.5;    // Minimum X velocity
private static final double VMAX = 3.0;    // Maximum Y velocity
```

Design Approach

The first requirement is a data structure to hold the gBall 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 gBall 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 gBall object,

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

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

```
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 moveSort(iBall);           // this will move all gBalls to their sorted position
```

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();
```

```
// In the gBall generation loop

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

// Following the gBall generation loop

      while (myTree.isRunning());              // Block until termination
      String CR = readLine("CR to continue");  // Prompt user
      myTree.moveSort();                       // Lay out balls in order
```

Modifications will also be needed to the gBall 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.

Instructions

1. Modify the bTree class, `bTree.java`, to accommodate gBall objects and write the additional methods described earlier.
2. Modify the gBall class, `gBall.java`, to provide a method for returning run state and moving the ball.
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).

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, gBall.java, gUtil.java, bTree.java. We will build and test your code from here.

Your screenshot files should be named prompt.pdf and end.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.

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 Oct 2018

```java
import java.awt.Color;

import acm.graphics.GLabel;
import acm.graphics.GOval;
import acm.graphics.GRect;
import acm.program.GraphicsProgram;
import acm.util.RandomGenerator;

/**
 * The main class in Assignment 3.
 * Here the canvas and ball objects are created and the resulting simulation
 * runs its course.
 * As each ball is created, it is recorded in a B-Tree, so that it
 * can be accessed later on.  Once the simulation ends
 *
 * @author ferrie
 *
 */

public class bSim extends GraphicsProgram{

// 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 int NUMBALLS = 15;                         // # balls to simulate
    private static final double MINSIZE = 1;                        // Minumum ball size
    private static final double MAXSIZE = 8;                        // Maximum ball size
    private static final double XMIN = 10;                          // Min X starting location
    private static final double XMAX = 50;                          // Max X starting location
    private static final double YMIN = 50;                          // Min Y starting location
    private static final double YMAX = 100;                         // Max Y starting location
    private static final double EMIN = 0.4;                         // Minimum loss coefficient
    private static final double EMAX = 0.9;                         // Maximum loss coefficient
    private static final double VMIN = 0.5;                         // Minimum X velocity
    private static final double VMAX = 3.0;                         // Maximum Y velocity
```

```
    public static void main(String[] args) {                        // Standalone Applet
        new bSim().start(args);
        }

    public void run() {
        this.resize(WIDTH,HEIGHT+OFFSET);           // optional, initialize window size

// Create the ground plane

    GRect gPlane = new GRect(0,HEIGHT,WIDTH,3);
    gPlane.setColor(Color.BLACK);
    gPlane.setFilled(true);
    add(gPlane);

// Set up random number generator & B-Tree

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

// Generate a series of random gballs and let the simulation run till completion

        for (int i=0; i<NUMBALLS; i++) {
            double Xi = rgen.nextDouble(XMIN,XMAX);             // Current Xi
            double Yi = rgen.nextDouble(YMIN,YMAX);             // Current Yi
            double iSize = rgen.nextDouble(MINSIZE,MAXSIZE);    // Current size
            Color iColor = rgen.nextColor();                    // Current color
            double iLoss = rgen.nextDouble(EMIN,EMAX);          // Current loss coefficient
            double iVel = rgen.nextDouble(VMIN,VMAX);           // Current X velocity

            gBall iBall = new gBall(Xi,Yi,iSize,iColor,iLoss,iVel);     // Generate instance
            add(iBall.myBall);                                          // Add to display list
            myTree.addNode(iBall);                                      // Save instance
            iBall.start();                                              // Start this instance
        }

// Wait until simulation stops

        while (myTree.isRunning());         // Block until simulation terminates
```

```
//      String CR = readLine("CR to continue");

// For standalone application with no console, use graphics display

        GLabel myLabel = new GLabel("Click mouse to continue");
        myLabel.setLocation(WIDTH-myLabel.getWidth()-50,HEIGHT-myLabel.getHeight());
        myLabel.setColor(Color.RED);
        add(myLabel);
        waitForClick();
        myLabel.setLabel("All Sorted!");

//      myTree.clearBalls(this);
//      myTree.inorder();                   // Echo sizes on console
        myTree.moveSort();                  // Lay out balls from left to right in size order


    }
}
```

```java
import java.awt.Color;

import acm.graphics.GOval;

/**
 * This class provides a single instance of a ball falling under the influence
 * of gravity.  Because it is an extension of the Thread class, each instance
 * will run concurrently, with animations on the screen as a side effect.  We take
 * advantage here of the fact that the run method associated with the Graphics
 * Program class runs in a separate thread.
 *
 * @author ferrie
 *
 * Updates:  added new methods to support Assigment 3
 *
 *      void setBState(boolean state)           // Enables (true) or disables (false) simulation
 *      void moveTo(double x, double y)         // Forces ball to new (x,y) location where
 *                                              // x and y are in simulation coordinates
 */
public class gBall extends Thread {

    /**
     * The constructor specifies the parameters for simulation.  They are
     *
     * @param Xi        double The initial X position of the center of the ball
     * @param Yi        double The initial Y position of the center of the ball
     * @param bSize     double The radius of the ball in simulation units
     * @param bColor    Color  The initial color of the ball
     * @param bLoss     double Fraction [0,1] of the energy lost on each bounce
     * @param bVel      double X velocity of ball
     */

    public gBall(double Xi, double Yi, double bSize, Color bColor, double bLoss, double bVel) {

        this.Xi = Xi;                                       // Get simulation parameters
        this.Yi = Yi;
        this.bSize = bSize;
        this.bColor = bColor;
```

```
    this.bLoss = bLoss;
    this.bVel = bVel;

    // Create instance of ball using specified parameters
    // Remember to offset X and Y by the radius to locate the
    // bounding box

    myBall = new GOval(gUtil.XtoScreen(Xi-bSize),gUtil.YtoScreen(Yi+bSize),
                       gUtil.LtoScreen(2*bSize),gUtil.LtoScreen(2*bSize));
    myBall.setFilled(true);
    myBall.setFillColor(bColor);
    bState=true;                                        // Simulation on by default

}

/**
 * The run method implements the simulation from Assignment 1.  Once the start
 * method is called on the gBall instance, the code in the run method is
 * executed concurrently with the main program.
 * @param void
 * @return void
 */

public void run() {

    // Run the same simulation code as for Assignment 1

    double time = 0;                          // Simulation clock
    double total_time = 0;                    // Tracks time from beginning
    double vt = Math.sqrt(2*G*Yi);            // Terminal velocity
    double height = Yi;                       // Initial height of drop

    int dir = 0;                              // 0 down, 1 up
    double last_top = Yi;                     // Height of last drop
    double el = Math.sqrt(1.0-bLoss);         // Energy loss scale factor for velocity

    // This while loop computes height:
    // dir=0:  falling under gravity -->  height =  h0 - 0.5*g*t^2
```

```
// dir=1:  vertical projectile motion --> height = vt*t - 0.5*g*t^2

while (bState) {                          // Simulation can be halted
    if (dir == 0) {
        height = last_top - 0.5*G*time*time;
        if (height <= bSize) {
            dir=1;
            last_top = bSize;
            time=0;
            vt = vt*el;
        }
    }
    else {
        height = bSize + vt*time -0.5*G*time*time;
        if (height < last_top) {
            if (height <= bSize) break; // Stop simulation when top of
                dir=0;                  // last excursion is ball diameter.
                time=0;
        }
        last_top = height;
    }

// Determine the current position of the ball in simulation coordinates
// and update the position on the screen

    Yt = Math.max(bSize,height);                // Stay above ground!
    Xt = Xi + bVel*total_time;
    myBall.setLocation(gUtil.XtoScreen(Xt-bSize),gUtil.YtoScreen(Yt+bSize));

// Delay and update clocks

    try {
        Thread.sleep((long) (TICK*500));
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    time+=TICK;
```

```
        total_time+=TICK;

    }
    bState=false;                       // Can determine when simulation finished
}

// Class Methods

/**
 * Enable/Disable simulation asynchronously
 * @param boolean state
 * @return void
 */

void setBState(boolean state) {
    bState=state;
}

/**
 * Move the ball to specified simulation coordinates.
 * Update the position of the ball on the screen to match.
 * @param double x
 * @param double y
 * @return void
 */

void moveTo(double x, double y) {
    Xt=x;
    Yt=y;
    myBall.setLocation(gUtil.XtoScreen(Xt),gUtil.YtoScreen(Yt));
}

/**
 * Instance Variables & Class Parameters
 */

public GOval myBall;
private double Xi;
```

```
    private double Yi;
    public double bSize;
    private Color bColor;
    private double bLoss;
    private double bVel;
    public boolean bState;
    private double Xt;
    private double Yt;

    public static final double G=9.8;           // Gravitational acceleration
    private static final double TICK = 0.1; // Clock tick duration

}
```

```java
/**
 * Some helper methods to translate simulation coordinates to screen
 * coordinates
 * @author ferrie
 *
 */
public class gUtil {

    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;                 // Pixels/meter

    /**
     * X coordinate to screen x
     * @param X
     * @return x screen coordinate - integer
     */

    static int XtoScreen(double X) {
        return (int) (X * SCALE);
    }

   /**
    * Y coordinate to screen y
    * @param Y
    * @return y screen coordinate - integer
    */

    static int YtoScreen(double Y) {
        return (int) (HEIGHT - Y * SCALE);
    }

    /**
     * Length to screen length
     * @param length - double
     * @return sLen - integer
     */
```

```
    static int LtoScreen(double length) {
        return (int) (length * SCALE);
    }

    /**
     * Delay for <int> milliseconds
     * @param int time
     * @return void
     */

    static void delay (long time) {
        long start = System.currentTimeMillis();
        while (true) {
            long current = System.currentTimeMillis();
            long delta = current - start;
            if (delta >= time) break;
        }
     }
}
```

```java
/**
 * Implements a B-Tree class for storing gBall objects
 * @author ferrie
 *
 */

public class bTree {

    // Instance variables

    bNode root=null;

/**
 * addNode method - wrapper for rNode
 */

    public void addNode(gBall data) {
        root=rNode(root,data);
    }

/**
 * rNode method - recursively adds a new entry into the B-Tree
 */

    private bNode rNode(bNode root, gBall data) {
        if (root==null) {
            bNode node = new bNode();
            node.data = data;
            node.left = null;
            node.right = null;
            root=node;
            return root;
        }
        else if (data.bSize < root.data.bSize) {
            root.left = rNode(root.left,data);
        }
        else {
            root.right = rNode(root.right,data);
```

```
        }
        return root;
    }

/**
 * inorder method - inorder traversal via call to recursive method
 */

    public void inorder() {
        traverse_inorder(root);
    }

/**
 * traverse_inorder method - recursively traverses tree in order and prints each node.
 */

    private void traverse_inorder(bNode root) {
        if (root.left != null) traverse_inorder(root.left);
        System.out.println(root.data.bSize);
        if (root.right != null) traverse_inorder(root.right);
    }

/**
 * isRunning predicate - determines if simulation is still running
 */

    boolean isRunning() {
        running=false;
        recScan(root);
        return running;
    }

    void recScan(bNode root) {
        if (root.left != null) recScan(root.left);
        if (root.data.bState) running=true;
        if (root.right != null) recScan(root.right);
    }
```

```
/**
 * clearBalls - removes all balls from display
 * (note - you need to pass a reference to the display)
 *
 */

    void clearBalls(bSim display) {
        recClear(display,root);
    }

    void recClear(bSim display,bNode root) {
        if (root.left != null) recClear(display,root.left);
        display.remove (root.data.myBall);
        if (root.right != null) recClear(display,root.right);
    }

/**
 * drawSort - sorts balls by size and plots from left to right on display
 *
 */

    void moveSort() {
        nextX=0;
        recMove(root);
    }

    void recMove(bNode root) {
        if (root.left != null) recMove(root.left);
    //
    // Plot ball along baseline
    //
        double X = nextX;
        double Y = root.data.bSize*2;
        nextX = X + root.data.bSize*2 + SEP;
        root.data.moveTo(X, Y);;
        if (root.right != null) recMove(root.right);
    }
```

```
// Example of a nested class //

    public class bNode {
        gBall data;
        bNode left;
        bNode right;
    }

// Instance variables

    boolean running;
    double nextX;

// Parameters

    private static final double SEP = 0;


}
```