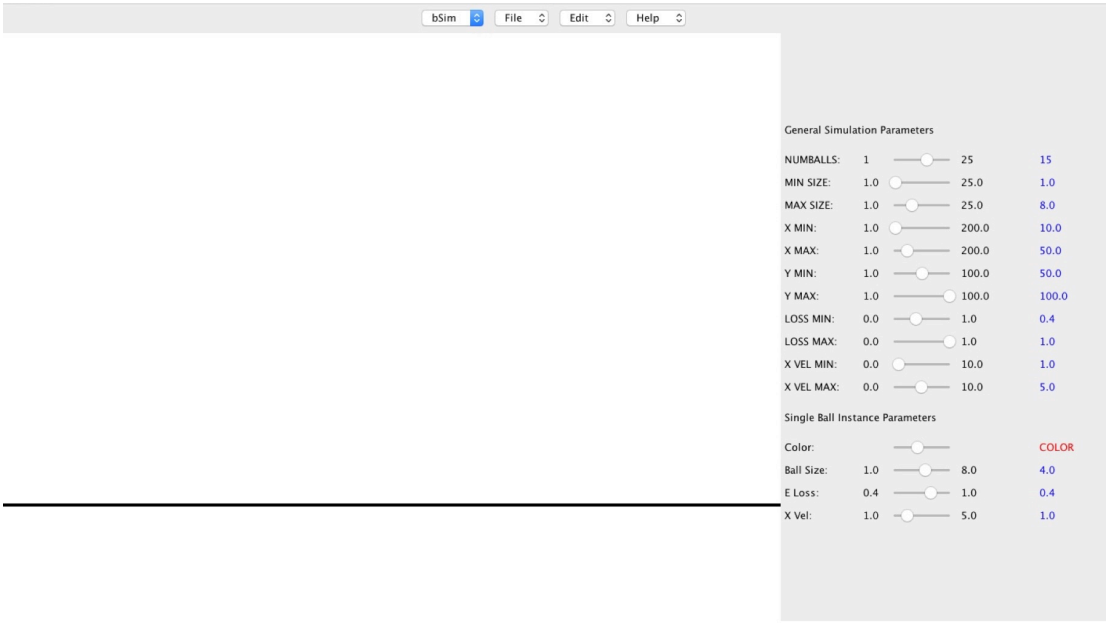
ECSE 202 – Introduction to Software Development
Assignment 4
Update

Background

After talking to many of you and answering questions about Assignment 4, we've decided to issue this update to clarify a number of points that you have raised, as well as to re-scope the assignment to be feasible for students without prior coding experience.

Requirements (100 points)



Your application should present something similar to the screenshot shown above. There should be a set of sliders that control the parameters of the simulation that were fixed in Assigment 3. In the screenshot, these are listed under General Simulation Parameters. There should be at least one JcomboBox for selecting the operations that can be performed and should include Run, to start a new simulation, and Quit, to exit the program. Selecting Run should start a new simulation that erases the previous output and begins a new simulation using the parameters selected using using the sliders. If you wish, you can include an explicit Clear option to force the clear at the discretion of the user (e.g. to abort the currently running simulation). In effect, A4 adds a graphical user interface to A3, permitting the user to interact with the program.

Bonus (25 points)

The bonus part of the question takes user interaction one step further. At any time the user should be able to select any ball, moving or not, and drag it to a new position on the screen. When released, the ball's parameters change to values provided by a second set of sliders, the Single Ball Instance Parameters, shown in the figure above. When released, the ball simulation runs forward from that point in time until it runs out of energy and halts. If the selected ball is released before the main simulation has completed, it should appear in the sort in proper order, otherwise, it simply runs independently until it stops. This mode of operation continues until the simulation is re-started.

Hints

1. Consider writing a sliderBox class that assembles a JLabel identifying the box next to a JLabel showing the minimum value, followed by a JSlider, followed by another JLabel to identify the max value, and finally a final JLabel to display the selected value. As the slider moves, the readout should change accordingly as shown below.

To perform this assembly, you can create a JPanel, which operates exactly the same way as the display canvas. You can also tell JPanel which layout manager to use so that you can control alignment more precisely. Here is partial code for a sliderBox class.

```
public sliderBox(String name, Integer min, Integer dValue,
Integer max) {        //  Integer values
      myPanel = new JPanel();
      nameLabel = new JLabel(name);
      minLabel = new JLabel(min.toString());
      maxLabel = new JLabel(max.toString());
      mySlider = new JSlider(min,max,dValue);
      sReadout = new JLabel(dValue.toString());
      sReadout.setForeground(Color.blue);
      myPanel.setLayout(new TableLayout(1,5));
```

```
        myPanel.add(nameLabel,"width=100");
        myPanel.add(minLabel,"width=25");
        myPanel.add(mySlider,"width=100");
        myPanel.add(maxLabel,"width=100");
        myPanel.add(sReadout,"width=80");
        imin=min;
        imax=max;
}
```

This code corresponds to the constructor for this class (which does most of the heavy lifting). Since you'll need to handle both integer and double values, you will need a second constructor to handle doubles. Finally you will have to write get and set methods for both cases inorder to read back values and update the display.

Note that you can use the same approach to put multiple slider boxes inside a JPanel, all precisely aligned using the TableLayout manager, and then add that panel to the default canvas.

2. Once you have your set of slider boxes, you will need to set up listeners,

```
MaxSize.mySlider.addChangeListener((ChangeListener) this);
XMin.mySlider.addChangeListener((ChangeListener) this);
XMax.mySlider.addChangeListener((ChangeListener) this);
etc…
```

   and then implement the stateChanged method to catch events generated by the sliders,

```
public void stateChanged(ChangeEvent e) {
JSlider source = (JSlider)e.getSource();

      if (source==NumBalls.mySlider) {
            PS_NumBalls=NumBalls.getISlider();
            NumBalls.setISlider(PS_NumBalls);
      }
      else if (source==MinSize.mySlider) {
            PS_MinSize=MinSize.getFSlider();
            MinSize.setFSlider(PS_MinSize);
      }
      etc…
}
```

3. The other interface element that you will need is a JComboBox to implement the chooser menus (by the way, it's OK to use a JComboBox to select the colors if you wish – I choose not to for aesthetic reasons).

```
void setChoosers() {
      bSimC = new JComboBox<String>();
      bSimC.addItem("bSim");
      bSimC.addItem("Run");
      bSimC.addItem("Clear");
      bSimC.addItem("Stop");
      bSimC.addItem("Quit");
      add(bSimC,NORTH);
}
```

You don't have to follow the example literally – this is just to show you how to use this element. Next, you need to set up an item change listener,

```
void addJComboListeners() {
      bSimC.addItemListener((ItemListener)this);
      etc…
}
```

and then implement the item change listener method,

```
public void itemStateChanged(ItemEvent e) {
      JComboBox source = (JComboBox)e.getSource();

      if (source==bSimC) {
            if (bSimC.getSelectedIndex()==1) {
                  System.out.println("Starting simulation");
                  this.SimRunning=true;
            }
      etc…
      }
```

4. Implementing the Bonus will require some additional features for the bTree class, namely methods for i) searching for the gBall instance that corresponds to a specific GOval returned on a muse click, ii) re-sorting the bTree in the event that the size parameters of one or more gBalls have changed. The first is a straightforward traversal comparing an input GOval reference with those at each node of the tree. The second is easily done by writing a copy function that traverses the bTree to be re-sorted, using the addNode() method to build aa new tree in the process. Once this traversal is complete, the reference to the new bTree is copied over to the old.
5. When a particular GOval instance is selected, it is often easier to kill its corresponding thread and create a new gBall instance (that is then attached to the GOval), rather than modifying the gBall class.

Final Points

The program is being judged on functionality, so as long as it performs the functions outlined above, all is good. However, there is one exception. Since this is supposed to be built on A1, A2, and A3, you may not change the underlying storage method , i.e., you may not use arrays to store gBalls. If you do, there will be a penalty. Have a look at the posted solutions for A3.

Fpf/November 1, 2018

```
import java.awt.Color;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;

import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import acm.graphics.GLabel;
import acm.graphics.GObject;
import acm.graphics.GOval;
import acm.graphics.GPoint;
import acm.graphics.GRect;
import acm.gui.TableLayout;
import acm.program.GraphicsProgram;
import acm.util.RandomGenerator;

/**
 * The main class in Assignment 4
 * bSim has been modified by inclusion of an interactive component that allows
 * the user to
 * 1. Override the default parameters for the simulation
 * 2. Intervene at any time by grabbing a ball, moving it to a
 *    new location, and dropping it.
 * 3. Ending or restrting the simulation by pressing a button.
 * 4. Bonus:
 *
 * @author ferrie
 *
 */

public class bSim extends GraphicsProgram implements ChangeListener, ItemListener {

// Parameters used in this program
```

```
    private static final int WIDTH = 1400;                          // 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 = 0.2;                         // Minimum loss coefficient
    private static final double EMAX = 1.0;                         // Maximum loss coefficient
    private static final double VMIN = 1.0;                         // Minimum X velocity
    private static final double VMAX = 5.0;                         // Maximum Y velocity

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

    public void init() {
        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

        rgen = RandomGenerator.getInstance();

// Set up user interface

        setSliders();                               // Parameter sliders
        setChoosers();                              // Pull-down menus
```

```
// Listeners

        addActionListeners();                           // Slider events
        addMouseListeners();                            // Mouse events
        addJSliderListeners();                          // Add change listener for JSliders
        addJComboListeners();                           // Add item item change listners for JComboBox

/**
 *  Main simulation loop
 */

        while(true) {
            pause(200);
            if (SimRunning) {
                runSimulation();
            }
        }
    }


/**
 * Simulation is placed in its own method
 */

    void runSimulation() {

        myTree = new bTree();

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

        for (int i=0; i<PS_NumBalls; i++) {
            double Xi = rgen.nextDouble(PS_XMin,PS_XMax);               // Current Xi
            double Yi = rgen.nextDouble(PS_YMin,PS_YMax);               // Current Yi
            double iSize = rgen.nextDouble(PS_MinSize,PS_MaxSize);      // Current size
            Color iColor = rgen.nextColor();                            // Current color
            double iLoss = rgen.nextDouble(PS_EMin,PS_EMax);            // Current loss coefficient
            double iVel = rgen.nextDouble(PS_VMin,PS_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

// Draw balls in sort order/

            GLabel myLabel = new GLabel("Click mouse to continue");
            myLabel.setLocation(WIDTH-myLabel.getWidth()-500,HEIGHT-myLabel.getHeight());
            myLabel.setColor(Color.RED);
            add(myLabel);
            waitForClick();
            myTree.moveSort();                  // Lay out balls from left to right in size order
            myLabel.setLabel("");
            SimRunning=false;                   // Halt the simulator
        }

/**
 * Method to set up sliders for simulation parameters.  Uses the sliderBox class
 * with the TableLayout manager to create a single JPanel that can be added
 * to the simple window manager used in GraphicsProgram.
 *
 */

    void setSliders() {
//
// Best done using the Table Layout Manager inside of a JPanel which
// subsequently gets added to the right side of the screen.
//
// First layout the general simulation parameters
//
        JPanel myPanel = new JPanel();
```

```
        myPanel.setLayout(new TableLayout(30, 1));
        myPanel.add(new JLabel("General Simulation Parameters"));
        myPanel.add(new JLabel("                                    "));
        NumBalls = new sliderBox("NUMBALLS: ",1,NUMBALLS,25);       // Number of balls
        myPanel.add(NumBalls.myPanel);
        MinSize = new sliderBox("MIN SIZE: ",1.0,MINSIZE,25.0);     // Minimum ball size
        myPanel.add(MinSize.myPanel);
        MaxSize = new sliderBox("MAX SIZE: ",1.0,MAXSIZE,25.0);     // Maximum ball size
        myPanel.add(MaxSize.myPanel);
        XMin = new sliderBox("X MIN: ",1.0,XMIN,200.0);             // X min
        myPanel.add(XMin.myPanel);
        XMax = new sliderBox("X MAX: ",1.0,XMAX,200.0);             // X max
        myPanel.add(XMax.myPanel);
        YMin = new sliderBox("Y MIN: ",1.0,YMIN,100.0);             // Y min
        myPanel.add(YMin.myPanel);
        YMax = new sliderBox("Y MAX: ",1.0,YMAX,100.0);             // Y max
        myPanel.add(YMax.myPanel);
        EMin = new sliderBox("LOSS MIN: ",0.0,0.4,1.0);             // Minimum energy loss
        myPanel.add(EMin.myPanel);
        EMax = new sliderBox("LOSS MAX: ",0.0,EMAX,1.0);            // Maximum energy loss
        myPanel.add(EMax.myPanel);
        VMin = new sliderBox("X VEL MIN: ",0.0,VMIN,10.0);          // Minimum X velocity
        myPanel.add(VMin.myPanel);
        VMax = new sliderBox("X VEL MAX: ",0.0,VMAX,10.0);          // Number of balls
        myPanel.add(VMax.myPanel);
//
// And now the ball instance parameters
//
        myPanel.add(new JLabel("                                    "));
        myPanel.add(new JLabel("Single Ball Instance Parameters"));
        myPanel.add(new JLabel("                                    "));
        CChooser = new sliderBox("Color: ",Color.red);
        myPanel.add(CChooser.myPanel);
        BSize = new sliderBox("Ball Size:",MINSIZE,PI_BSize,MAXSIZE);
        myPanel.add(BSize.myPanel);
        ELoss = new sliderBox("E Loss:",EMIN,PI_ELoss,EMAX);
        myPanel.add(ELoss.myPanel);
        XVel = new sliderBox("X Vel:",VMIN,PI_XVel,VMAX);
```

```
        myPanel.add(XVel.myPanel);
        add(myPanel,EAST);
    }

/**
 * Method to set up chooser-like pull down menus across the top
 * of the display.
 */

    void setChoosers() {
        bSimC = new JComboBox<String>();
        bSimC.addItem("bSim");
        bSimC.addItem("Run");
        bSimC.addItem("Clear");
        bSimC.addItem("Stop");
        bSimC.addItem("Quit");
        add(bSimC,NORTH);
        FileC = new JComboBox<String>();
        FileC.addItem("File");
        add(FileC,NORTH);
        EditC = new JComboBox<String>();
        EditC.addItem("Edit");
        add(EditC,NORTH);
        HelpC = new JComboBox<String>();
        HelpC.addItem("Help");
        add(HelpC,NORTH);
    }

/**
 * Method to set up change listener for each slider box
 */

    void addJSliderListeners() {
        NumBalls.mySlider.addChangeListener((ChangeListener) this);
        MinSize.mySlider.addChangeListener((ChangeListener) this);
        MaxSize.mySlider.addChangeListener((ChangeListener) this);
        XMin.mySlider.addChangeListener((ChangeListener) this);
        XMax.mySlider.addChangeListener((ChangeListener) this);
```

```
        YMin.mySlider.addChangeListener((ChangeListener) this);
        YMax.mySlider.addChangeListener((ChangeListener) this);
        EMin.mySlider.addChangeListener((ChangeListener) this);
        EMax.mySlider.addChangeListener((ChangeListener) this);
        VMin.mySlider.addChangeListener((ChangeListener) this);
        VMax.mySlider.addChangeListener((ChangeListener) this);
        CChooser.mySlider.addChangeListener((ChangeListener) this);
        BSize.mySlider.addChangeListener((ChangeListener) this);
        ELoss.mySlider.addChangeListener((ChangeListener) this);
        XVel.mySlider.addChangeListener((ChangeListener) this);
    }

/**
 * Method to set up item change listeners for JComboBoxes
 */

    void addJComboListeners() {
        bSimC.addItemListener((ItemListener)this);
        FileC.addItemListener((ItemListener)this);
        EditC.addItemListener((ItemListener)this);
        HelpC.addItemListener((ItemListener)this);
    }

/**
 * Method to handle dispatch for JComboBox
 */

    public void itemStateChanged(ItemEvent e) {
        JComboBox source = (JComboBox)e.getSource();

        if (source==bSimC) {                                    // Only bSim active at the moment
            if (bSimC.getSelectedIndex()==1) {
                System.out.println("Starting simulation");
                this.SimRunning=true;
            }
            else if (bSimC.getSelectedIndex()==2) {
                System.out.println("Clearing simulation");
                myTree.clearBalls(this);
```

```
            }
            else if (bSimC.getSelectedIndex()==3) {
                System.out.println("Stopping simulation");
                this.SimRunning=false;
            }
            else if (bSimC.getSelectedIndex()==4) {
                System.out.println("Shutting down");
                System.exit(0);
            }
        }
    }

/**
 * Method to handle dispach for sliders
 */

    public void stateChanged(ChangeEvent e) {
        JSlider source = (JSlider)e.getSource();

        if (source==NumBalls.mySlider) {                    // Global parameter sliders
            PS_NumBalls=NumBalls.getISlider();
            NumBalls.setISlider(PS_NumBalls);
        }
        else if (source==MinSize.mySlider) {
            PS_MinSize=MinSize.getFSlider();
            MinSize.setFSlider(PS_MinSize);
        }
        else if (source==MaxSize.mySlider) {
            PS_MaxSize=MaxSize.getFSlider();
            MaxSize.setFSlider(PS_MaxSize);
        }
        else if (source==XMin.mySlider) {
            PS_XMin=XMin.getFSlider();
            XMin.setFSlider(PS_XMin);
        }
        else if (source==XMax.mySlider) {
            PS_XMax=XMax.getFSlider();
            XMax.setFSlider(PS_XMax);
```

```
}
else if (source==YMin.mySlider) {
    PS_YMin=YMin.getFSlider();
    YMin.setFSlider(PS_YMin);
}
else if (source==YMax.mySlider) {
    PS_YMax=YMax.getFSlider();
    YMax.setFSlider(PS_YMax);
}
else if (source==EMin.mySlider) {
    PS_EMin=EMin.getFSlider();
    EMin.setFSlider(PS_EMin);
}
else if (source==EMax.mySlider) {
    PS_EMax=EMax.getFSlider();
    EMax.setFSlider(PS_EMax);
}
else if (source==VMin.mySlider) {
    PS_VMin=VMin.getFSlider();
    VMin.setFSlider(PS_VMin);
}
else if (source==VMax.mySlider) {
    PS_VMax=VMax.getFSlider();
    VMax.setFSlider(PS_VMax);
}
else if (source==CChooser.mySlider) {               // Instance parameter settings
    PI_Color=CChooser.getCSlider();
    CChooser.setCSlider(PI_Color);
}
else if (source==BSize.mySlider) {
    PI_BSize=BSize.getFSlider();
    BSize.setFSlider(PI_BSize);
}
else if (source==ELoss.mySlider) {
    PI_ELoss=ELoss.getFSlider();
    ELoss.setFSlider(PI_ELoss);
}
else if (source==XVel.mySlider) {
```

```
            PI_XVel=XVel.getFSlider();
            XVel.setFSlider(PI_XVel);
        }
    }

/**
 * Methods to handle mouse events
 */

    public void mousePressed(MouseEvent e) {
        last = new GPoint(e.getPoint());
        obj = getElementAt(last);
        match=null;
        if (obj != null) {
            match = myTree.nSearch((GOval) obj);
            if (match != null) {
                match.bState=false;         // Kill thread
            }
        }
    }

    public void mouseDragged(MouseEvent e) {
        if (match != null) {
            match.myBall.move(e.getX() - last.getX(), e.getY() - last.getY());
            last=new GPoint(e.getPoint());
        }
    }

    public void mouseReleased(MouseEvent e) {
        if (match!=null) {
            updateSimParams((int)e.getX(),(int)e.getY());
        }
    }

/**
 * Method to update simulation parameters
 */
```

```
    void updateSimParams(int X, int Y) {
        double Xt=gUtil.ScreentoX(X)+match.bSize;
        double Yt=gUtil.ScreentoY(Y)-match.bSize;

        // Substitute a new simulation instance

        gBall repl = new gBall(Xt,Yt,PI_BSize,PI_Color,PI_ELoss,PI_XVel);   // New sim
        repl.myBall=match.myBall;                                           // Attach to GOval
        myTree.replace(match,repl);                                         // Substitute in tree
        match=repl;
        myTree.reorder();                                                   // Reorder tree

        // Update the screen display

        match.myBall.setFillColor(PI_Color);
        match.myBall.setSize(gUtil.LtoScreen(2*PI_BSize),gUtil.LtoScreen(2*PI_BSize));
        match.myBall.setLocation(gUtil.XtoScreen(Xt-match.bSize),gUtil.YtoScreen(Yt+match.bSize));
        match.start();
    }

/**
 * Instance variables - simulation parameters
 */

    private int PS_NumBalls=NUMBALLS;       // Global simulation parameters for the rgen
    private double PS_MinSize=MINSIZE;
    private double PS_MaxSize=MAXSIZE;
    private double PS_XMin=XMIN;
    private double PS_XMax=XMAX;
    private double PS_YMin=YMIN;
    private double PS_YMax=YMAX;
    private double PS_EMin=EMIN;
    private double PS_EMax=EMAX;
    private double PS_VMin=VMIN;
    private double PS_VMax=VMAX;

    private Color PI_Color=Color.red;
    private double PI_BSize=4;
```

```
    private double PI_ELoss=0.4;
    private double PI_XVel=VMIN;

/**
 * Instance variables - slider boxes
 *
 */

    private sliderBox NumBalls;
    private sliderBox MinSize;
    private sliderBox MaxSize;
    private sliderBox XMin;
    private sliderBox XMax;
    private sliderBox YMin;
    private sliderBox YMax;
    private sliderBox EMin;
    private sliderBox EMax;
    private sliderBox VMin;
    private sliderBox VMax;
    private sliderBox CChooser;
    private sliderBox BSize;
    private sliderBox ELoss;
    private sliderBox XVel;

/**
 * Instance variables - JCombo boxes
 *
 */

        private JComboBox<String> bSimC;
        private JComboBox<String> FileC;
        private JComboBox<String> EditC;
        private JComboBox<String> HelpC;

/**
 * Instance variables - simulation
 */
```

```
RandomGenerator rgen;
bTree myTree;
boolean SimRunning=false;
GObject obj;
gBall match;
GPoint last;









}
```

```
import acm.graphics.GOval;

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

public class bTree {

/**
 * 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);
        root.data.bState=false;         // Kill controlling thread
        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.Xt=X;
        root.data.Yt=Y;
        root.data.moveTo(X, Y);
        if (root.right != null) recMove(root.right);
    }
```

```
/**
 * gBall nSearch - node search routine.  Returns the matching gBall object
 * @param - GOval ball, GOval object within the gBall being searched for
 * @author ferrie
 *
 */

    gBall nSearch(GOval ball) {
        match=null;
        rnSearch(root,ball);
        return match;
    }

    void rnSearch(bNode root,GOval ball) {
        if (root.left != null) rnSearch(root.left,ball);
        if (root.data.myBall==ball) match=root.data;
        if (root.right != null) rnSearch(root.right,ball);
    }

/**
 * Replace a node on the tree
 * @author ferrie
 */

    void replace(gBall Ball, gBall rBall) {
        rreplace(root,Ball,rBall);
    }

    void rreplace(bNode root, gBall Ball, gBall rBall) {
        if (root.left != null) rreplace(root.left,Ball,rBall);
        if (root.data==Ball) root.data=rBall;
        if (root.right != null) rreplace(root.right,Ball,rBall);

    }

/**
 * Method to access the root node of a bTree
```

```
 */

    bNode getRoot() {
        return root;
    }

/**
 * Method to reorder the tree in sort order
 * @author ferrie
 */

    void reorder() {
        bTree copy = new bTree();
        rreorder(root,copy);
        root=copy.getRoot();
    }

    void rreorder(bNode root, bTree copy)
    {
        if (root.left != null) rreorder(root.left,copy);
        copy.addNode(root.data);
        if (root.right != null) rreorder(root.right,copy);
    }
// Example of a nested class //

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

// Instance variables

    bNode root=null;
    boolean running;
    double nextX;
    gBall match;
```

```
// Parameters

    private static final double SEP = 0;

}
```

```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;            // All public - allows state to be
```

```
    public double Xi;               // modified given a pointer to
    public double Yi;               // this object.
    public double bSize;
    public Color bColor;
    public double bLoss;
    public double bVel;
    public boolean bState;
    public double Xt;
    public 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 = 1400;                          // n.b. screen coordinates
    private static final int HEIGHT = 600;
    private static final int OFFSET = 200;
    private static final double SCALE = HEIGHT/100.0;                   // Pixels/meter

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

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

    /**
     * Screen x to X
     */

    static double ScreentoX(int x) {
        return x/SCALE;
    }

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

    static int YtoScreen(double Y) {
```

```
        return (int) (HEIGHT - Y * SCALE);
    }

    /**
     * Screen to Y
     */

    static double ScreentoY(int y) {
        return (HEIGHT-y)/SCALE;
    }

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

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

```java
import java.awt.Color;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;

import acm.graphics.GRect;
import acm.gui.TableLayout;

/**
 * This class creates a sliderBox object consisting of a JSlider with a set of
 * JLabels for defining minimum and maximum values, and a numerical readout
 * corresponding to the current position of the slider.  Accessor methods
 * are provided for each.
 * @author ferrie
 *
 */
public class sliderBox {

/**
 * This constructor creates an instance of a JPanel using the default
 * layout manager, and populates it as follows:
 *      Parameter: JLabel --- JSlider --- JLabel      JLabel
 *  @param min - Integer, min slider value
 *  @param dValue - Integer, default slider value
 *  @param max - Integer, max slider value
 */

    public sliderBox(String name, Integer min, Integer dValue, Integer max) {       //  Integer values
        myPanel = new JPanel();
        nameLabel = new JLabel(name);
        minLabel = new JLabel(min.toString());
        maxLabel = new JLabel(max.toString());
        mySlider = new JSlider(min,max,dValue);
        sReadout = new JLabel(dValue.toString());
        sReadout.setForeground(Color.blue);
        myPanel.setLayout(new TableLayout(1,5));
        myPanel.add(nameLabel,"width=100");
```

```
    myPanel.add(minLabel,"width=25");
    myPanel.add(mySlider,"width=100");
    myPanel.add(maxLabel,"width=100");
    myPanel.add(sReadout,"width=80");
    imin=min;
    imax=max;
}

public sliderBox(String name, Double min, Double dValue, Double max) {          // Floating point
 values
    myPanel = new JPanel();
    nameLabel = new JLabel(name);
    minLabel = new JLabel(min.toString());
    maxLabel = new JLabel(max.toString());
    mySlider = new JSlider(JSMIN,JSMAX,(int) ((JSMAX-JSMIN)*(dValue/(max-min))));
    sReadout = new JLabel(dValue.toString());
    sReadout.setForeground(Color.blue);
    myPanel.setLayout(new TableLayout(1,5));
    myPanel.add(nameLabel,"width=100");
    myPanel.add(minLabel,"width=25");
    myPanel.add(mySlider,"width=100");
    myPanel.add(maxLabel,"width=100");
    myPanel.add(sReadout,"width=80");
    fmin=min;
    fmax=max;
}

public sliderBox(String name, Color dValue) {                                   // Color chooser
    myPanel = new JPanel();
    nameLabel = new JLabel(name);
    mySlider = new JSlider(JSMIN,Clut.length,Clut.length/2);
    colorDisp = new JLabel("COLOR");
    colorDisp.setForeground(dValue);
    myPanel.setLayout(new TableLayout(1,5));
    myPanel.add(nameLabel,"width=100");
    myPanel.add(new JLabel("   "),"width=25");
    myPanel.add(mySlider,"width=100");
    myPanel.add(new JLabel("   "),"width=100");
```

```
        myPanel.add(colorDisp,"width=80");
    }

/**
 * Accessor methods to read back slider values as reals and integers.
 */

    public int getISlider() {                                   // Get value at slider - int
        return mySlider.getValue();
    }

    void setISlider(Integer arg) {
        sReadout.setText(arg.toString());                       // Set value on readout - int
    }

    public double getFSlider() {                                // Get slider and do linear
     interplation
        double scale = ((double)mySlider.getValue())/JSMAX;     // to get floating point
        return fmin+scale*(fmax-fmin);
    }

    void setFSlider(Double arg) {                               // Set value on readout - double
        sReadout.setText(arg.toString());
    }

    public Color getCSlider() {                                 // Get color slider
        return Clut[mySlider.getValue()];
    }

    void setCSlider(Color arg) {                                // Set color slider
        colorDisp.setForeground(arg);
    }


/**
 * Instance Variables for this class
 *
 */
```

```
    JPanel myPanel;
    JLabel nameLabel;
    JLabel minLabel;
    JLabel maxLabel;
    JLabel sReadout;
    JSlider mySlider;
    JLabel colorDisp;
    int imin;
    int imax;
    double fmin;
    double fmax;
    Color Clut[] = {Color.red, Color.green, Color.blue, Color.magenta, Color.cyan, Color.yellow,
                    Color.black, Color.white, Color.gray, Color.darkGray, Color.lightGray,
                    Color.orange, Color.pink
    };

/**
 * Parameters
 */

    private static final int JSMIN=1;
    private static final int JSMAX=100;

}
```