Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 4
Interacting with Objects

Introduction

In the last assignment, you implemented a data structure (a binary-tree) to organize and store your data (the ball objects you had created). In this assignment, you will learn how to implement tools that allow a user to interact **asynchronously** with the simulation program, while still taking advantage of the architecture surrounding the objects and data structure you have previously created. Previous assignments in the course asked for user intervention (e.g. entering parameter values) at particular points in the flow of the program (e.g. the beginning, or once all balls have stopped moving). An asynchronous interface allows the user to intercede at any point, usually by using the mouse or the keyboard to signal that a particular action should be triggered. As explained in the chapter on "Object-Oriented Graphics" in your textbook *The Art and Science of Java*:

> Events that occur asynchronously with respect to the program operation—mouse clicks, key strokes, and the like—are represented using a structure called an **event**. When an event occurs, the response is always the invocation of a method in some object that is waiting to hear about that event. Such an object is called a **listener**.

You will implement **listeners** and **events** to control parameters that already exist in your simulation, but in a way that allow you users to interact with the simulation as it runs. For this assignment, you will rely on concepts in the chapter on "Object-Oriented Graphics" in the textbook, as well as the previous material in the text, and mostly the lecture notes on "Object Oriented Graphics" and "Event Driven Programs".

Problem Description

Extend the program from Assignment 3 to allow the following asynchronous user interactions.

1) **Global, initial simulation parameters for whole simulation**. User interaction: items on top border of window – North control strip.
2) **Individual simulation parameters for a single ball (thread)**. User interaction: items on right border of window – East control strip; Mouse interactions with objects (balls) displayed in windows – clicking, clicking and dragging.

For 1) The global parameters that the user will set are
Start Button that clears screen and starts simulation with global parameters and randomly generated individual parameters for balls (as in previous assignment)
Number of balls to simulate (1 to 30) Slider
Sort – as in assignment 3, once all threads are no longer moving, pressing this button will display the sorted balls.
(All other global parameters are set as in the previous assignment)

For 2) The individual parameters

Clicking on a ball or clicking and dragging a ball will record the final x and y position of the mouse (on click or on click drag and release) and apply this starting x and y position to that particular ball as well as any individual parameters entered in the below interactive components
Starting X position – Field that displays x position (captured from mouse)
Starting Y position - – Field that displays y position (captured from mouse)
Ball size – (1 to 8) Slider
Colour – colour of ball. Combo box
Loss coefficient – text field
HVel – Horizontal velocity (.5 to 3) Slider

Design Approach

As mentioned above, you will add appropriate listeners and events to the main class of your assignment (bSim) to handle the asynchronous user interactions required above. The global parameters will set off the overall bSim loop using these entered global parameters. The local parameters will affect the individual clicked ball, stop the simulation for that thread, set the parameters as entered by the user, and restart the simulation for that thread. This will all happen on the release of the mouse click, so the user must enter in the local parameters first and then click on the ball that he/she would like to apply the parameters to.

You will be able to use the code you have written for Assignment 3. You will need to add methods related to listening and events (use the examples in the text and slide as starting points). You will need to modify some existing methods to accommodate the new interactions.
To give you some context, you will find some example code below for mouse clicks based on the textbook examples. You can use similar examples from the text and notes to set up the user interactions using the different menu items.

The principal class that will run the simulation (as in assignment 3), **bSim**

```
public class bSim extends GraphicsProgram{
.
.
//parameters used previously in the program
.
.
.
// new example parameters
private GObject gobj; /* The object being dragged */
private double lastX; /* The last mouse X position */
private double lastY; /* The last mouse Y position */
private GOval clickedOval; /* will cast GObject to this
variable*/
private bNode clickedNode; /* you will pass clickedOval to a
function that will return a node containing the underlying ball
object that this GOval belongs to */
```

btree

```
private bTree myTree; /*You might have previously declared this
as a variable within the run function but you can have it as a
class member variable to be accessed by other functions */
     .
     .
     public void run(){
     .
     .
     addMouseListeners(); /*You will need to listen for mouse
events */
     .
     .
   }

     /** Method called on mouse press to record the coordinates
of the click */
     public void mousePressed(MouseEvent e) {
     lastX = e.getX();
     lastY = e.getY();
     gobj = getElementAt(lastX, lastY);
     if (!(gobj==null)) {
          clickedOval = (GOval)gobj; //casting to GOval
          /* next find the object in the BTree and pass back the
containing node - this example uses a method called findNode -
but you can traverse the tree as you like, possibly using
existing methods you have previously written*/
          clickedNode = myTree.findNode(clickedOval);
          //change parameters of object as needed
          /* The following is an example that changes the border
colour of a ball - you of course will change other parameters */
          clickedNode.data.myBall.setColor(Color.RED);
          /* the following is an example that stops the
simulation for the associated ball thread*/
          clickedNode.data.setBState(false);
     }
     else {
          println("null"); /* show in console that click was not
on an object */
     }


     }
}
```

bNode

```
/* if you had the node class nested in another class you will
want it public to access from BSim */
public class bNode {

     gBall data;
     bNode left;
     bNode right;


}
```

bTree

```
public class bTree {
.
.
.
.

/*
 * Recursive methods related to finding the gBall in the bTree
with a particular member GOval
*/
     public bNode findNode(GOval value) {
         return findNodeRecursive(root, value);
     }

     private bNode findNodeRecursive(bNode current, GOval value)
{
     .
         /*code to traverse tree and find GOval that matches
instance variable in data of particular node and return that
node */
     .
     }
}
```

Instructions

1. Modify the bTree class, bTree.java, to accommodate the new design of returning the node of a matched object and separating bNode into a separate class.
2. Modify the gBall class, gBall.java, only if your particular implementation requires it - it might not be necessary.
3. Modify the bSim class, bSim.java, to complete the program design allowing asynchronous user interaction.
4. Before submitting your assignment, run your own tests to verify correct operation.

To Hand In:

1) The source java files.
2) A screenshot file (pdf) showing the simulation in action.
3) A screenshot file showing the balls in sorted order

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 action.jpg and sorted.jpg 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/pl Oct 2018