Mehran Sahami
CS 106A

Handout #39A
November 15, 2017

Solution to Section #7

Based on a handout by Eric Roberts

1. Using Interactors

```
// File: BoxDiagram.java
// This program allows the user to create a set of boxes with labels
// and then drag them around in the window.

import acm.graphics.*;
import acm.program.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;

public class BoxDiagram extends GraphicsProgram {

/* Initializes the program */
   public void init() {
      contents = new HashMap<String,GObject>();
      createController();
      addActionListeners();
      addMouseListeners();
   }

/* Creates the control strip at the bottom of the window */
   private void createController() {
      nameField = new JTextField(MAX_NAME);
      nameField.addActionListener(this);    // detect ENTER pressed too
      addButton = new JButton("Add");
      removeButton = new JButton("Remove");
      clearButton = new JButton("Clear");
      add(new JLabel("Name"), SOUTH);
      add(nameField, SOUTH);
      add(addButton, SOUTH);
      add(removeButton, SOUTH);
      add(clearButton, SOUTH);
   }

/* Adds a box with the given name at the center of the window */
   private void addBox(String name) {
      GCompound box = new GCompound();
      GRect outline = new GRect(BOX_WIDTH, BOX_HEIGHT);
      GLabel label = new GLabel(name);
      box.add(outline, -BOX_WIDTH / 2, -BOX_HEIGHT / 2);
      box.add(label, -label.getWidth() / 2, label.getAscent() / 2);
      add(box, getWidth() / 2, getHeight() / 2);
      contents.put(name, box);
   }

/* Removes all boxes in the contents table */
   private void removeContents() {
      Iterator<String> iterator = contents.keySet().iterator();
      while (iterator.hasNext()) {
         remove(contents.get(iterator.next()));
      }
      contents.clear();    // Clear all entries in the HashMap
   }
```

```
/* Called in response to button actions */
   public void actionPerformed(ActionEvent e) {
      Object source = e.getSource();
      if (source == addButton || source == nameField) {
         addBox(nameField.getText());
      } else if (source == removeButton) {
         String text = nameField.getText();
         GObject obj = contents.get(text);
         if (obj != null) {
            remove(obj);
            contents.remove(text);
         }
      } else if (source == clearButton) {
         removeContents();
      }
   }

/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      currentObject = getElementAt(last);
   }

/* Called on mouse drag to reposition the object */
   public void mouseDragged(MouseEvent e) {
      if (currentObject != null) {
         currentObject.move(e.getX() - last.getX(),
                            e.getY() - last.getY());
         last = new GPoint(e.getPoint());
      }
   }

/* Called on mouse click to move this object to the front */
   public void mouseClicked(MouseEvent e) {
      if (currentObject != null) currentObject.sendToFront();
   }

/* Private constants */
   private static final int MAX_NAME = 25;
   private static final double BOX_WIDTH = 120;
   private static final double BOX_HEIGHT = 50;

/* Private instance variables */
   private HashMap<String,GObject> contents;
   private JTextField nameField;
   private JButton addButton;
   private JButton removeButton;
   private JButton clearButton;
   private GObject currentObject;
   private GPoint last;

}
```

2. Using the Debugger

Error #1:

In **`setupBall()`**, the argument **`ball`** is only a copy of the reference declared in **`run()`**:

```
private void setupBall(GOval ball) {
      ball = new GOval(BALL_RADIUS * 2, BALL_RADIUS * 2);
```

We initialize a ball in **`setupBall()`**, but when the method finishes, we no longer have any reference to the **`GOval`** we allocated. Thus, back in **`run()`**, we try moving a null **`GOval`**:

```
GOval ball = null;
…
setupLabel();
setupBall(ball);

while (true) {
      ball.move(dx, dy);
      ...
```

We might think that any changes we make to an object are saved, no matter where the changes are made. This is largely true, but only when changes are made to an *already existing* object. Like an **`int`**, **`double`**, or **`boolean`**, the reference itself is just a copy when passed to another method. Consequently, we can't change the reference itself, only data *inside* the object.

Solution(s):

Either allocate the **`ball GOval`** *before* passing it to **`setupBall()`**, or make **`ball`** an instance variable and don't even bother passing the **`GOval`** as an argument.

Error #2:

`checkForCollisions()` is supposed to change **`dx`** and **`dy`**, but we are only passing *copies* of the two variables:

```
double dx = rgen.nextDouble(MIN_DX, MAX_DX);
double dy = rgen.nextDouble(MIN_DY, MAX_DY);
…
while (true) {
      ball.move(dx, dy);
      checkForCollisions(ball, dx, dy);
```

In other words, **`checkForCollisions()`** will have its own copies of **`dx`** and **`dy`**, and any changes we make to the two variables will not affect the original variables in **`run()`**.

Solution(s):

Make **`dx`** and **`dy`** instance variables and don't even bother passing them as arguments. Not every variable should be an instance variable, of course. But as a rule of thumb, any variable that will be changed over the course of multiple methods should be an instance variable.

If you need to save any changes to an **`int`**, **`double`**, or **`boolean`**, you'll need to return that value out of the method. Of course, you can only return one variable from a method. As an alternative solution, we could have split **`checkForCollisions()`** into two separate methods **`checkForXCollision()`** and **`checkForYCollision()`** that return the updated values.

Error #3:

The switch statement in **`getRandomNewColor()`** is missing **`break`** statements at the end of each **`case`** (and the end of **`default`** too, as a formality). Because of this, **`newColor`** will always be **`Color.BLACK`**. Moreover, once the label and ball are already black, **`getRandomNewColor()`** will infinite loop because **`newColor`** is always the same as **`prevColor`** and thus we will never return from the method:

```
if (!newColor.equals(prevColor)) return newColor;
```

Thus, the program will get stuck in **`checkForCollisions()`** when calling **`getRandomNewColor()`**.

Solution(s):

Add **`break`** statements at the end of **`default`** and at the end of each **`case`**.

Error #4:

We adjust the location of the **`GLabel`** before we adjust the size:

```
double x = (getWidth() / 2.0) - (text.getWidth() / 2.0);
double y = getHeight() - TEXT_HEIGHT;
text.setLocation(x, y);
text.setFont(new  Font("Arial",  Font.BOLD,  32));
```

As a result, the location is based off the smaller, default size.

Solution(s):

Resize the **`GLabel`** first and only afterwards set the location.