Eric Roberts
CS 106A

Handout #52A
May 21, 2012

Solution to Section #7

1. Drawing box diagrams

```
/*
 * 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.awt.event.*;
import java.util.*;
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);
      nameField.setActionCommand("Add");
      add(new JLabel("Name"), SOUTH);
      add(nameField, SOUTH);
      add(new JButton("Add"), SOUTH);
      add(new JButton("Remove"), SOUTH);
      add(new JButton("Clear"), 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 the box with the given name */
   private void removeBox(String name) {
      GObject obj = contents.get(name);
      if (obj != null) {
         remove(obj);
         contents.remove(name);
      }
   }
```

```
/* Clears the screen and the stored copy in the contents hash map */
   private void clear() {
      removeAll();
      contents.clear();
      nameField.setText("");
   }

/* Called in response to button actions */
   public void actionPerformed(ActionEvent e) {
      String cmd = e.getActionCommand();
      if (cmd.equals("Add")) {
         addBox(nameField.getText());
      } else if (cmd.equals("Remove")) {
         removeBox(nameField.getText());
      } else if (cmd.equals("Clear")) {
         clear();
      }
   }

/* 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 GObject currentObject;
   private GPoint last;

}
```

2. Implementing a class

```
/*
 * File: CalendarDate.java
 * -----------------------
 * This file defines a class to support a simple month/day/year object.
 */

public class CalendarDate {

/** Creates a new date object with the specified fields */
   public CalendarDate(int mm, int dd, int yyyy) {
      month = mm;
      day = dd;
      year = yyyy;
   }

/** Extracts the month field */
   public int getMonth() {
      return month;
   }

/** Extracts the day field */
   public int getDay() {
      return day;
   }

/** Extracts the year field */
   public int getYear() {
      return year;
   }

/** Compares this date to a second one */
   public int compareTo(CalendarDate d2) {
      if (year < d2.year) return -1;
      if (year > d2.year) return +1;
      if (month < d2.month) return -1;
      if (month > d2.month) return +1;
      if (day < d2.day) return -1;
      if (day > d2.day) return +1;
      return 0;
   }

/** Converts a date to a string */
   public String toString() {
      return MONTH_NAMES[month - 1] + " " + day + ", " + year;
   }

/* Array containing names of the months */
   private static final String[] MONTH_NAMES = {
      "January", "February", "March", "April", "May", "June",
      "July", "August", "September", "October", "November", "December"
   };

/* Private instance variables */
   private int month, day, year;

}
```

3. Data structure design

```
/*
 * File: PotionManager.java
 * ------------------------
 * This file exports a class to manage potion ingredients, as in the
 * style of Harry Potter.
 */

import acm.util.*;
import java.io.*;
import java.util.*;

public class PotionManager {

/**
 * Creates a new PotionManager object and initializes it from the data
 * in the specified file.  If the file does not exists, this method throws
 * throw an ErrorException.
 */
   public PotionManager(String filename) {
      potionMap = new TreeMap<String,ArrayList<String>>();
      try {
         BufferedReader rd = new BufferedReader(new FileReader(filename));
         while (true) {
            String name = rd.readLine();
            if (name == null) break;
            potionMap.put(name, readIngredients(rd));
         }
         rd.close();
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }
   }

/**
 * Returns an array of the potion names stored in this object.  The
 * elements of the returned array must be sorted in lexicographic order.
 */
   public String[] getPotionNames() {
      String[] names = new String[potionMap.size()];
      int index = 0;
      for (String name : potionMap.keySet()) {
         names[index++] = name;
      }
      return names;
   }

/**
 * Returns an array of the ingredients for the named potion.  If the
 * potion name does not exist, this method returns null.
 */
   public String[] getIngredients(String potionName) {
      ArrayList<String> ingredientList = potionMap.get(potionName);
      if (ingredientList == null) return null;
      return ingredientList.toArray(new String[ingredientList.size()]);
   }
```

☞

```
/* Reads a list of ingredients from the reader */
   private ArrayList<String> readIngredients(BufferedReader rd) {
      try {
         ArrayList<String> ingredients = new ArrayList<String>();
         while (true) {
            String line = rd.readLine();
            if (line == null || line.length() == 0) break;
            ingredients.add(line);
         }
         return ingredients;
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }
   }

/* Instance variables */
   private TreeMap<String,ArrayList<String>> potionMap;

}
```