0. Name (2 pts):

CS 2334: Programming Structures and Abstractions

Midterm Exam II

Solution Set

Monday, November 9, 2009

Problem	Topic	Max	Grade
0	Name	2	
1	Binary I/O	15	
2	Graphical User Interfaces	25	
3	Event-Driven Programming and MVC	35	
4	Collections Framework	25	
Total			

1. **Binary I/O** (15 pts)

(a) (5 pts) True or False: a String is represented as a binary number when it is written to an ASCII text file.

True. (Everything is represented as binary numbers – in the files and inside the computer.)

(b) (10 pts) Briefly describe the advantage of using ObjectOutputStream and ObjectInputStream over using DataOutputStream and DataInputStream.

The latter provide methods for reading/writing primitive types. The former can read/write entire objects in a single method call. Hence, the former is easier to implement (less code) and easier to maintain (adding or removing properties does not require a change to the read/write code).

2. **Graphical User Interfaces** (25 pts)

(a) (10 pts) Briefly explain why paintComponent() should be declared as "protected" and not "public".

The Java programmer should not call this method, except when calling from the paintComponent() method of a child class (which the "protected" keyword allows). Most of the time, this method is called by the Java graphics management system (a process that is largely hidden from the Java programmer).

(b) (10 pts) Briefly describe the function and benefits of using a layout manager.

A container object has a single instance of a layout manager. This manager is responsible for making decisions about where each of the child components is placed within the container's space on the screen. This process can be very complicated due to window and panel resizing and to the different space needs of all of the children. Delegating the layout management process to a standard set of layout manager classes can substantially reduce the complexity of writing a GUI.

(c) (5 pts) True or False: a layout manager implements the getPreferredSize() method.

False. (The layout manager calls the getPreferredSize() for each of the components within its associated container)

3. **Event-Driven Programming and the Model View Controller Approach** (35 pts)

(a) (10 pts) List two advantages of event-based programming over standard procedural programming.

1. Event-based programming allows us to write modular code (by coupling the listener for events directly with the source object).

2. A pure procedural programming approach can waste a substantial amount of CPU time by "busy waiting" on events to happen. Event-based programming triggers the execution of code when an event happens (note that not all busy waiting is eliminated, but it is centralized and efficient).

(b) (10 pts) List the three types of classes that play roles in event-driven programming. Briefly describe their function in the process.

1. Source objects: originators of events.

2. Event objects: the class that describes the details of an event.

3. Listener objects: are registered with the source object and provide a method (or methods) that is (are) called on the production of an event.

(c) (10 pts) True or False, and briefly explain: in the MVC approach, an event Listener is part of the Model.

False. The event listener is a part of the Viewer. The Viewer registers the Listener with the Model.

(d) (5 pts) How many JList objects can register a Listener with a single DefaultListModel?

An arbitrary number.

4. **Java Collections Framework** (25 pts)

Consider the following code:

```
public class Person implements Comparable<Person>
{
    public int ID;
    public String name;

    public Person(int ID, String name) {
        this.ID = ID;
        this.name = name;
    };

    public String toString() {
        return(name + ", " + ID);
    };

    public int compareTo(Person p) {
        return(this.name.compareTo(p.name));
    };
};
```

```
import java.util.*;

public class driver {

    public static void displayList(List list) {
        System.out.println("##");
        for(Object o: list) {
             System.out.println(o);
        };
    };

    public static void main(String[] args) {
        LinkedList<Person> list = new LinkedList<Person>();
        Person p1 = new Person(45, "Bob");
        Person p2 = new Person(25, "Zella");
        Person p3 = new Person(32, "Arnie");

        list.add(p1);
        list.add(p2);
        list.add(p3);

        Collections.sort(list);

        displayList(list);

        Collections.sort(list, new Comparator<Person>() {
                              public int compare(Person p1, Person p2){
                                  if(p1.ID > p2.ID) return(1);
                                  if(p1.ID < p2.ID) return(-1);
                                  return(0);
                              };
             });

        displayList(list);
    };
}
```

(a) (15 pts) What output does the program produce?

```
##
Arnie, 32
Bob, 45
Zella, 25
##
Zella, 25
Arnie, 32
Bob, 45
```

(b) (10 pts) True or False, and briefly explain: a LinkedHashSet would be an adequate substitute in the above code for the LinkedList.

False. We would want to be able to have multiple records that contain the same name but with different ID's. In addition, LinkedHashSet's do not allow us to impose a specific ordering of the objects (e.g., using the Collections.sort() method).