Fall 2015 (Thursday, December 3) Name: _____

CS 3331 — Advanced Object-Oriented Programming

Final Exam

This test has 5 questions and pages numbered 1 through 10.

Reminders

This test is closed-notes and closed-book. However, you may bring one page (8.5 X 11) of notes (both sides). Your notes must be your own, and they must be hand-written and turned in with your test. This test is to be done individually, and you are not to exchange or share your notes with others during the test.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. Your test will not be graded if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

Clarity is important; if your writing or code is sloppy or hard to read, you will lose points. Correct syntax also makes some difference.

There are 110 points all.

---

1. Short answers (30 points)

   (a) (2 points) A Software Design Pattern is a schematic description of a design solution to a recurring problem in software design. It is generic and reusable but doesn't have to be implemented in the same way.

   (b) (2 points) The Template design pattern defines a skeleton algorithm by deferring some steps to subclasses. It allows the subclasses to redefine certain steps of the algorithm, often called *hook methods* or *hot spots*.

   (c) (2 points) The Strategy design pattern defines a family of algorithms by encapsulating each member of the family and making the members interchangeable. It should be used when many related classes differ only in their behavior or when different variants of an algorithm is needed.

   (d) (2 points) This design pattern should be used to access the contents of a collection without exposing its internal representation, to support multiple traversal of a collection, and to provide a uniform interface for traversing different collections.

      i. Template method
      ii. Strategy
      iii. Iterator
      iv. Abstract factory

(e) (2 points) The Java GUI framework consists of several categories of classes. The main categories are the followings except for:

i. GUI components (or widgets)
ii. Layout managers
iii. Events and event listeners
iv. Graphics and imaging classes
v. Collection classes

(f) (4 points) The design of Java GUI framework uses several design patterns, including:

- It uses the Composite design pattern to compose GUI components into tree structures to represent part-whole hierarchies.
- Each container has a *layout manager*, that handles the layout of the components contained in the container. The design of the layout managers uses the Strategy design pattern.

(g) (2 points) This layout manager arranges as many as five components in five positions identified as North, South, East, West, and Center. Which layout manager is described?

i. BorderLayout
ii. FlowLayout
iii. GridLayout
iv. GridBagLayout

(h) (2 points) Which of the followings is not the naming convention used throughout AWT and Swing?

i. The names of event classes end with the suffix `Event`.
ii. For event class *Xyz*`Event`, the associated listener interface is usually named *Xyz*`Listener`.
iii. There is always an associated adapter class for a listener interface *Xyz*`Listener`, and its name is *Xyz*`Adapter`.
iv. For a listener that implements interface *Xyz*`Listener`, the name of the method to register the listener to its source is `add`*Xyz*`Listener`.

(i) (2 points) There are several different implementations of the `List` interface, each providing different time and space complexity for various operations. The LinkedList implementation (class) is preferred if a list is dynamic or volatile in that it grows and shrinks a lot.

(j) (2 points) The design of the Java IO classes uses the Decorator design pattern to enhance, or provide add-on features to, the basic IO capability through object composition.

(k) (2 points) All the following statements about a server socket are true except for:

i. It waits for connection requests from clients.

ii. It listens at a specific port. The port number is necessary to distinguish different servers running on the same host.

iii. It is an instance of the `java.net.ServerSocket` class.

iv. It can be used to send and receive data.

v. It must be running on the server host before its clients initiate contact.

vi. The `accept()` method of a server socket blocks the caller until a request is received.

(l) (2 points) A client socket is an instance of the `java.net.Socket` class and has a pair of an `InputStream object` and an `OutputStream` object for receiving and sending data. There are two ways to obtain a client socket: on the client side, it can be created using the constructor `Socket(String, port)`, and on the server side, it is returned by the _____() method of the `ServerSocket` class.

(m) (4 points) Threads can be created and declared in one of two ways:

- by directly extending the `java.lang.`Thread class, and
- by implementing the `java.lang.`Runnable interface.

(n) (2 points) Synchronization ensures the mutual exclusion of two or more threads in the critical regions, but it does not address cooperation among threads. The most common technique or pattern for supporting thread cooperation is known as Guarded Suspension in which a method call and a calling thread are suspended until a precondition (acting as a guard) is satisfied.

2. (5 points) If the `synchronized` modifier is removed from the following code snippet, what difference does it make? Be specific about your answer.

```
public synchronized void put(Object e) {
    if (!isFull()) {
       back = (back + 1) % size;
       rep[back] = e;
       count++;
    }
}
```

Without the synchronized modifier, this method will no longer be atomic, meaning it will not run uninterrupted anymore.

3. (10 points) Consider the following socket-based game server class, named `GameServer`, which is supposed to support multiple clients concurrently. There is a serious problem in supporting multiple clients. Explain the problem, and rewrite the code to fix it.

```
import java.net.*;

public class GameServer {
    /** Launch this server. */
    public start() {
        try {
            ServerSocket server = new ServerSocket(8000);
            while (true) {
                Socket socket = server.accept();
                serveClient(socket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /** Serve the client on the given socket. */
    private void serveClient(Socket socket) { ... }
}
```

The serveClient() method should be run on a different thread so the server can go back to accepting new clients. Below is a modified version that should avoid this problem:

```
public class GameServer extends Thread{
    Socket socket;
    public startServer(){
        try{
            ServerSocket server = new ServerSocket(8000);
            while(true){
                socket = server.accept();
                this.start(); //calls the run method on this instance of GameServer
            }
        } catch (IOException e) {
                e.printStackTrace();
    }

    public void run(){
        serveClient(socket); //This method will be called on a seperate thread
    }

    private void serveClient(Socket socket){…}
}
```

4. (total 20 points) You are to define a class named `Book` and ordering among its instances. A book may have many attributes, but here you will consider only two: the title (of type `String`) and the publication year (of type `int`). Below you will define two different orderings for books so that they can be sorted in different orders.

   (a) (10 points) Define a `Book` class that implements a natural ordering among its instances. The natural order of books is based on the ascending order of the title and the publication year. A book, b_1, precedes another book, b_2, if the title of b_1 precedes that of b_2 in the lexicographic order. If both have the same title, the book published earlier precedes the other.

```
public class Book implements Comparable<Book>{ // <--- YOUR CODE HERE!
    private String title;
    private int year;

    public Book(String title, int year) {
        this.title = title;
        this.year = year;
    }

    public String title() {
        return title;
    }

    public int year() {
        return year;
    }

    // >>>WRITE YOUR CODE HERE<<<
```

```
public int compareTo(Book b2){
    if(b2 == null) //make sure the book is not null
        return 0;
    if(this.title() == null || b2.title() == null) //make sure both books have titles
        return 0;
    if(this.title().equals(b2.title())){
        if(this.year() == null || b2.year() == null) //make sure both books have years
            return 0;
        return this.year()-b2.year();
    }
    return (this.title().compareTo(b2.title()); //return the string comparison of titles
}
```

```
} // end of class Product
```

(b) (10 points) Define a comparator class, named `YearComparator`, for the `Book` class to order books based on their publication years. A book published earlier precedes a book published later. If two books are published in the same year, the lexicographic order of their titles determines the order.

```
public class YearComparator implements Comparator<Book>{
    public int compare(Book b1, Book b2){
        if(b1 == null || b2 == null)//make sure neither book is null
            return 0;
        if(b1.year() == null || b2.year() == null) { //make sure both books have a year
            if(b1.title() == null || b2.title() == null)//if no year, check titles
                return 0;
            return b1.title().compareTo(b2.title());
        }

        if(b1.year() - b2.year() == 0){ //if the year is the same check the title
            if(b1.title() == null || b2.title() == null) //make sure they have title
                return 0;
            return b1.title().compareTo(b2.title()); //
            }
        return b1.year()-b2.year();
    }
}
```

5. (total 35 points) You are to write a simple applet named `PhotoAlbumApplet` that can display a list of digital images (see Figure 1 below). The applet provides two control buttons to view previous and next images. Your applet should download digital images from the same directory where the applet is located, and the images are assumed to be stored in the files named `Picture1.jpg`, `Picture2.jpg`, ..., `Picture`N`.jpg`, where N is given by an applet parameter `numberOfPhotos`.

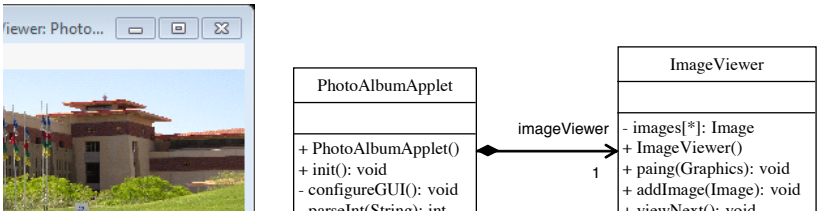

Figure 1: Sample screenshot (left) and detailed design (right)

(a) (20 points) Define the `PhotoAlbumApplet` class by completing the partial code given below. Do not add any more fields or methods. You can use `Image getImage(URL, String)` method of the `Applet` class to download an image, e.g., `getImage(getCodeBase(), "Picture1.jpg")`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PhotoAlbumApplet extends java.applet.Applet {

  /** A special panel that can display a list of images. */
  private final ImageViewer imageViewer = new ImageViewer();

  /** Parse a given string and return an int value. */
  private int parseInt(String s) {
      int result = 0;
      if (s != null) {
          try {
              result = Integer.parseInt(s);
          } catch (NumberFormatException e) {
          }
      }
      return result;
  }

  /** Create a new instance. */
  public PhotoAlbumApplet() {
      configureGUI();
  }

  // MORE CODE ON THE NEXT PAGE.
```

```
/** Configure GUI consisting of an image viewer and two buttons. */
private void configureGUI() {
    setLayout(new BorderLayout());
    add(imageViewer, BorderLayout.CENTER);

    JButton prevButton = new JButton("<Prev");
    JButton nextButton = new JButton("Next>");

  // >>>WRITE YOUR CODE HERE<<<
  // (Hint: refer to the ImageViewer class in part (b) of this question.)
```

JPanel buttons = new JPanel(new FlowLayout()); //create new flow layout for buttons

buttons.add(prevButton); //add prevButton to buttons
prevButton.addActionListener(h -> viewPrev()); //add event handler to prevButton

buttons.add(nextButton); //add nextButton to buttons
nextButton.addActionListener(h -> viewNext()); //add event handler to nextButton

add(buttons, BorderLayout.SOUTH); //the the flow layout below the imageviewer

```
  }

  /** Downloads pictures and add them to the image viewer. */
  @Override
  public void init() {
    // >>>WRITE YOUR CODE HERE<<<
    // (Hint: refer to part (b) of this question.)

      Image k;
      for(int i = 1; i<numberOfPhotos; i++) {
          k = getImage(getCodeBase(), "Picture" + i + ".jpg");
          addImage(k);
      }




  }
}
```

(b) (15 points) Define the `PhotoViwer` class used by the `PhotoAlbumApplet` class by completing its partial code given below. It is a special JPanel class that can contain a list of images and display them one at a time.

```
import java.awt.*;
import javax.swing.*;
import java.util.List;
import java.util.ArrayList;

/** A special panel that can display a list of images, one at a time. */
class ImageViewer extends JPanel {

  /** The list of images to display. */
  private final List<Image> images = new ArrayList<>();

  /** The 0-based index of the current image to display. */
  private int index = 0;

  /** Create a new instance. */
  public ImageViewer() {
      super(true); // enable double-buffering
  }

  /** Overridden here to paint the current image. */
  @Override
  public void paint(Graphics g) {
      if (images.size() > 0) {
          Dimension d = getSize();
          Image img = images.get(index);
          g.drawImage(img, 0, 0, d.width, d.height, Color.WHITE, this);
      }
  }

  /** Add the given image to this viewer. */
  public void addImage(Image image) {
      // >>>WRITE YOUR CODE HERE<<<
        images.add(image);









  }

  // MORE CODE ON THE NEXT PAGE.
```

```
  /** Display the next image. If there is no next image, display the first
   * one. */
  public void viewNext() {
      // >>>WRITE YOUR CODE HERE<<<
        index = (index+1)%imagres.size();
        repaint();









  }

  /** Display the previous image. If there is no previous image, display
   * the last one. */
  public void viewPrev() {
      // >>>WRITE YOUR CODE HERE<<<

       if(index = 0){
           index = images.size();
       }
       index - -;
       repaint();









  }
}
```