Reminders

This test is open book. You may also bring one page (8.5 × 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, including 10 points of extra credits.

1. (5 points) Answer briefly the following questions.

(a) One of the most distinguishing characteristics of object-oriented application frameworks is “inversion of control”. Explain what the inversion of control means.

(b) Explain the benefits of design patterns.
2. (total 20 points) In this problem you are to define a class Product that describes the products sold by an Internet shopping mall. The products may have many attributes, but in this problem you will consider only two attributes: the manufacturer (of type String) and the price (of type int). In the subproblems below you will define two different orderings for the products so that the customer can browse or list the products in different orders, e.g., sorted by manufacturers or by prices.

(a) (10 points) Define the class Product. The Product class should implement the Comparable interface to define a natural ordering on the products. The natural order of the product objects is based on the ascending order of the manufacturer and price attributes. That is, a product, \( p_1 \), precedes a product, \( p_2 \), if the manufacturer’s name of \( p_1 \) precedes the manufacturer’s name of \( p_2 \) in the lexicographic order. If both \( p_1 \) and \( p_2 \) are from the same manufacturer, the product with a lower price precedes the other.

public class Product implements Comparable {
    private String manufacturer;
    private int price;

    public Product(String manufacturer, int price) {
        this.manufacturer = manufacturer;
        this.price = price;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public int getPrice() {
        return price;
    }

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

} // end of class Product
(b) (10 points) Define a comparator class, named `PriceComparator`, for the class `Product` to sort the products based on their prices. A product with a lower price precedes a product with a higher price. If two products have the same price, the lexicographic order of their manufacturers determines the order.

3. (total 30 points) You are to define a new kind of sets called a `SieveSet`, which behaves like a regular set except that it has “holes” through which objects may fall; i.e., often objects are not added to and thus stored in a `SieveSet`. The `SieveSet` class defines a method, `boolean fallsThrough(Object)`, that returns true if the given object will “fall through” the sieve and hence is not stored. As a good OO practitioner, you decided to implement your `SieveSet` class as a direct subclass of the `HashSet` class. You also learned that the `addAll` method that your class inherits from its superclasses is defined as follows.

```java
/**
 * Adds all elements of the given collection c to this collection
 * and returns true if this collection changed as a result of the call.
 */
public boolean addAll(Collection c) {
    boolean result = false;
    for (Iterator i = c.iterator(); i.hasNext(); ) {
        result = result || add(i.next());
    }
    return result;
}
```

Therefore, you decided to override only the `add` method in your `SieveSet` class; i.e., you decided not to override the inherited `addAll` method.

(a) (5 points) Which design pattern are you using? Justify your answer by explaining the roles that the `add` and `addAll` methods play in the design pattern. (Hint: Consider part (b) of this question.)
(b) (15 points) Define the \texttt{SieveSet} class by completing the skeleton code given below. You also need to write, in JML, the pre and postconditions for the \texttt{add} method. (Caution: The return type of the \texttt{add} method is \texttt{boolean}.)

```java
public class SieveSet extends HashSet {

    /** Returns true iff the given object o falls through. */
    protected /*@ pure @*/ boolean fallsThrough(Object o) {
        return Math.random() < 0.5;
    }

    /**
     * Adds the given object o to this set if it is not already present
     * and o does not fall through; otherwise, do not add the given object.
     * Returns true if the object o was successfully added to this sieve set.
     */
    /*@ >>>WRITE YOUR PRE AND POSTCONDITION HERE<<< */

    public boolean add(Object o) {
        // >>>WRITE YOUR CODE HERE<<<
    }
}
```
(c) (10 points) Write a new class called `ControllableSieveSet` that behaves like a sieve set except that one can now specify the percentage that added items fall through the sieve. For example, `new ControllableSieveSet(30)` creates a new sieve set in which 30 percents of added items fall through. You can assume that a value between 0 and 100, inclusive, is passed as an argument to the constructor. You also need to provide a default constructor that creates a new sieve set in which half of the added items fall through. If you have duplicate code, you will be penalized for that. (Hint: You only need to define two simple constructors and one simple method. The static method `Math.random()` returns a random `double` number between 0 and 1.)

```java
public class ControllableSieveSet extends SieveSet {

    /** The percentage that the added items fall through. */
    private int percentage;

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

}
4. (total 30 points) You are to write a simple applet, called PhotoViewer, that can display a list of digital images (see Figure 1 below). The applet provides two control buttons to view previous and next images, respectively. Your applet should download digital images from the same directory where the applet is located. The images are assumed to be stored in the files Picture1.jpg, Picture2.jpg, …, PictureN.jpg, where N is given by the applet parameter number_of_pictures.

Figure 1: A sample screen of the PhotoViewer applet

(a) (15 points) Define the PhotoViewer class by completing the skeleton code given below. Do not add any more fields or methods; if you do, you will be penalized for that! (Hint: Consider part (b) of this question below before answering. You can use Image getImage(URL, String) method of the Applet class to download an image.)

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

public class PhotoViewer extends java.applet.Applet {

/**
 * A panel containing the list of pictures to view. The class PhotoAlbum
 * will be defined in part (b) of this question.
 */
private PhotoAlbum album = new PhotoAlbum();

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

// MORE CODE ON THE NEXT PAGE.
```
/** Creates a new instance by composing its GUI. */
public PhotoViewer() {
    JButton prevButton = new JButton("<Prev");
    JButton nextButton = new JButton("Next");

    // >>>WRITE YOUR CODE HERE<<<
    // (Hint: refer to part (b) of this question that define
    // the PhotoAlbum class.)
}

/** Downloads pictures and adds them to the album. */
public void init() {
    // >>>WRITE YOUR CODE HERE<<<
    // (Hint: refer to part (b) of this question.)
}
(b) (15 points) Define the PhotoAlbum class used by the PhotoViewer applet class.

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

public class PhotoAlbum extends JPanel {

    /**
     * The list of pictures to display. Each element is supposed to be
     * an object of type Image.
     */
    private List photos = new ArrayList();

    /** The index of the current picture to be displayed. */
    private int index = 0;

    public PhotoAlbum() {
        super(true); // enable double-buffering
    }

    /** Paints the current picture by scaling it. */
    public void paint(Graphics g) {
        if (photos.size() > 0) {
            Dimension d = getSize();
            Image p = (Image) photos.get(index);
            g.drawImage(p, 0, 0, d.width, d.height, Color.WHITE, this);
        }
    }

    /** Adds the given image to this album. */
    public void addPhoto(Image image) {
        // >>>WRITE YOUR CODE HERE<<<
        // (Hint: one line of code should be enough.)
    }
}
```

// MORE CODE ON THE NEXT PAGE.
/**
 * Displays the next picture. If there is no next picture (i.e., if
 * the current is the last picture), then displays the first picture.
 */
public void viewNext() {
    // >>>WRITE YOUR CODE HERE<<<

} 

/**
 * Displays the previous picture. If there is no previous picture (i.e., if
 * the current is the first picture), then displays the last picture.
 */
public void viewPrev() {
    // >>>WRITE YOUR CODE HERE<<<

} 
}
5. (total 15 points) In this problem you will write a simple timer class, called MyTimer, that fires one or more action events after a specified delay. For example, an animation object can use this class as the trigger for drawing its frames. Each timer has one or more action listeners and a delay (the time between action events). When delay milliseconds have passed, the timer fires an action event to its listeners; this cycle repeats forever.

(a) (10 points) Complete the MyTimer class below by adding the missing part.

```java
public class MyTimer {

    /** A list of listeners that have registered interest for
     * notification on this timer. */
    private List listeners = new ArrayList();

    /** A background thread that manages this timer’s tick and fires
     * action events after a specified delay. */
    private Thread timerThread = null;

    /**
     * Creates a timer that will notify its listeners every delay
     * milliseconds. The <code>delay</code> is assumed to be a positive
     * number. If <code>listener</code> is not null, it’s registered
     * as an action listener on the timer.
     * public MyTimer(final int delay, ActionListener listener) {
     *     addActionListener(listener);
     *     // event to be fired after every delay milliseconds.
     *     final ActionEvent e = new ActionEvent(this, 0, "time out");
     *     // >>>WRITE YOUR CODE HERE<<<
    }

} // end of the constructor
```
/** Adds an action listener to this timer. */
public void addActionListener(ActionListener listener) {
    if (listener != null && !listeners.contains(listener)) {
        listeners.add(listener);
    }
}

/** Removes the given action listener from this timer. */
public void removeActionListener(ActionListener listener) {
    listeners.remove(listener);
}

/**
 * Starts this timer, causing it to start sending action events
 * to its listeners. *
 */
public void start() {
    timerThread.start();
}

/**
 * Notifies all listeners that have registered interest for
 * notification on this timer. *
 */
protected void fireActionPerformed(ActionEvent e) {
    for (Iterator i = listeners.iterator(); i.hasNext(); ) {
        ((ActionListener) i.next()).actionPerformed(e);
    }
}

(b) (5 points) Which design pattern does the above code use? Justify your answer.
6. (total 10 points of extra credits) The MyTimer class on page 10 has a hidden data structure to store all the listeners. Suppose that some clients of the MyTimer class need to access these listeners, but as a good OO programmer you don’t want expose the internal data structure to the clients.

(a) (2 points) Which design pattern will you use to address this? That is, how will you let the clients access all the listeners sequentially but without exposing the internal data structure used to store the listeners?

(b) (8 points) Implement your approach by adding additional fields, methods, or classes to the MyTimer class. You only need to write the the additional code that you introduce.