University of Colorado Boulder

CSCI4448: Homework 3
Design Patterns

You may work on this assignment either as a pair or as a group of 3. *Divide up your project team into two groups!*
Name the .zip with each person's lastname **Lastname1_Lastname2_Lastname3_HW3.zip**
Upon unzip, you should have:

```
ProblemSets.PDF
Problem1
        x.java
        y.java
        z.png
Problem2
        a.java
etc.
```

For each of the following problems, submit your Java code solutions into its directory, and add to a single .PDF file ProblemSets.PDF with the following:

- Each person's name in the group at the top
- For each problem
  1. Problem #
  2. A brief write-up of which pattern you used (using only ones discussed in class).
  3. A class diagram of your solution (including existing classes), so future developers can easily see how to work with your solution. *(This can be auto-generated if you find a good tool, or you can do it yourself using a digital tool – no hand-drawn diagrams).*
  4. Which classes are which participants in your design pattern (add the participant labels to class diagram from #3).
  5. Who worked on this problem shown in a table:

40%	Cookie Monster	Pair-programmed with Elmo to create interface and concrete classes
55%	Elmo Elmo	Pair-programmed with Cookie Monster to create interface and concrete classes, also implemented the abstract class
0%	Big Bird	N/A

  6. An example run of your program that *clearly* shows that <u>your design pattern implementation</u> of your code works *(not just that your code produces some output)*.
  7. A copy of your code in the .PDF. Include the test driver program for each problem!

You will be graded mostly on the design patterns, but also on good programming practices so follow good naming conventions, etc. as well as clearly demonstrating your design pattern works both in the driver and your understanding of the solutions during an individual grading interview. <u>You need to be able to answer questions on all problem sets during your interview!</u> So be sure to teach each person in your group how you solved each problem (if you break it up instead of pair-program) as you will receive a group grade.
Note, if you use the Internet be sure to site your source!

Problem #1: Auto-grader Submission Management

Scenario

You are performing an update to an auto-grader system. The auto-grader receives submissions asynchronously, runs a test script, and produces a report. Looking closely at the process, you notice that the process could be easily multi-threaded! You implement the multithreaded aspect immediately, however, you still need to figure out a good way to queue submissions. You have a server which accepts submissions via several different paths, each with its own thread, and a set of Grader objects, each running in its own thread.

Problem

Your task is to create a class which consists of a queue (i.e., FIFO list) of Submission objects. For this example, submissions can be a dummy class (see below). This class must implement two methods: *add(Submission s)* and *process( )*. Additionally, you want to ensure that only one queue exists. Otherwise, every time a submission is submitted, a new queue will be created, and a Grader object will also not know which queue to select the next submission from.

Starter Code

```java
/**
* Submission.java
*
* A basic dummy Submission object for use with the auto-grader
*/

import java.util.Random;

public class Submission
{
  private static Random rand = new Random();
  private int id;

  public Submission()
  {
    // Give this submission a unique(ish) id
    id = rand.nextInt(10000000);
  }
}
```

Problem #2: Auto-grader Reporting

Scenario

Once again, you are performing a much needed update to a particular auto-grader for a certain computer science course. This time, however, you are trying to make the reporting mechanism easier to interact with the specific student submissions. In particular, you would like to generate two types of reports for each submission--one report simply involves counting the total number of correctly passed test cases, while the other report determines the number of time-out errors in a submission. Being wary of how the auto-grader has evolved in the past, you expect extra reports to be created in the future.

Problem

Your task is to create a pair of Report classes which maintain a specific type of report for a Submission object. The Submission class has been provided, but you must create the Report classes. After every test case, the submission should inform the Report objects of the results (i.e., pass or fail). If the test case failed, the error report should ask the Submission object if the most recent test failed due to a time-out (using the *wasTimeoutError* method).
You may add methods to the submission class as needed.
Note: Since the Submission object may be running on a separate thread from the Report objects, this is not simply a matter of running a test case, and then asking for the results.

For simplicity, reports can simply involve printing to the console.

```java
/**
* Submission.java
* A representation of a Submission
*/
import java.util.Random;

public class Submission
{
    private Random myRandom;
    private boolean lastErrorWasTimeout;
    // You may add attributes to this class if necessary

    public Submission()
    {
        myRandom = new Random();
        lastErrorWasTimeout = false;
    }
    public void runTestCase()
    {
        // For now, randomly pass or fail, possibly due to a timeout
        boolean passed = myRandom.nextBoolean();
        if(!passed)
        {
            lastErrorWasTimeout = myRandom.nextBoolean();
        }
        // You can add to the end of this method for reporting purposes
    }
    public boolean wasTimeoutError()
    {
        return lastErrorWasTimeout;
    }
}
```

Problem #3: Logging System

Fix the following code based on an appropriate design pattern (Where is the ‘smelly’ code?).

```
interface Logging {
     public void log(String msg);
}
class LogText implements Logging
{
    public LogText()
    {
        System.out.println("Logging: text format");
    }
    public void log(String msg)
    {
        System.out.println("Logging text to file: " + msg);
    }
}
class LogXML implements Logging
{
    public LogXML()
    {
        System.out.println("Logging: <type>XML Format</type>");
    }
    public void log(String msg)
    {
        System.out.println("Logging text to file: log.xml" );
        System.out.println("<xml><msg>"+msg+"</msg></xml>");
    }
}
class LogHTML implements Logging
{
    public LogHTML()
    {
        System.out.println("Logging: HTML format");
    }
    public void log(String msg)
    {
        System.out.println("Logging HTML to file: log.html" );
        System.out.println("<html><body>"+msg+"</body></html>");
    }
}
class Analysis
{
    public static void main(String[] args)
    {
        if (args.length != 1)
         {
            System.out.println("Usage: java Analysis type");
            System.exit(-1);
         }
        String type = args[0];
        Logging logfile;
        if (type.equalsIgnoreCase("text"))
            logfile = new LogText();
        else if (type.equalsIgnoreCase("xml"))
            logfile = new LogXML();
        else if (type.equalsIgnoreCase("html"))
            logfile = new LogHTML();
        else
            logfile = new LogText();
        logfile.log("Starting application...");
   }
}
```

Problem #4: Video Game Trees

Scenario

You are working at Video Games Ultra and they have a game that creates a bunch of trees for the terrain. They need to make a huge forest with thousands of trees placed randomly throughout the landscape.

Problem

When they scale the program up for the thousands of trees, the program is slowed down too much and takes up too many resources. Your task is to modify the provided code so that it doesn't slow down from too much overhead of creating tons of image objects.

```java
import java.util.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import javax.imageio.*;

interface Terrain
{
    void draw(Graphics graphics, int x, int y);
}
class Tree implements Terrain
{
    private int x;
    private int y;
    private Image image;
    public Tree(String type)
    {
        System.out.println("Creating a new instance of a tree of type " + type);
        String filename = "tree" + type + ".png";
        try
        {
            image = ImageIO.read(new File(filename));
        } catch(Exception exc) { }
    }
    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
    @Override
    public void draw(Graphics graphics, int x, int y)
    {
        graphics.drawImage(image, x, y, null);
    }
}
class TreeFactory
{
    private static final ArrayList<Tree> mylist = new ArrayList<Tree>();
    public static Terrain getTree(String type)
    {
        Tree tree = new Tree(type);
        mylist.add(tree);
        return tree;
   }
}
```

```java
/**
 * Don't change anything in TreeDemo
 */
class TreeDemo extends JPanel
{
    private static final int width = 800;
    private static final int height = 700;
    private static final int numTrees = 50;
    private static final String type[] = { "Apple", "Lemon", "Blob", "Elm", "Maple" };

    public void paint(Graphics graphics)
    {
        for(int i=0; i < numTrees; i++)
        {
            Tree tree = (Tree)TreeFactory.getTree(getRandomType());
            tree.draw(graphics, getRandomX(width), getRandomY(height));
        }
    }
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.add(new TreeDemo());
        frame.setSize(width, height);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    private static String getRandomType()
    {
        return type[(int)(Math.random()*type.length)];
    }
    private static int getRandomX(int max)
    {
        return (int)(Math.random()*max );
    }
    private static int getRandomY(int max)
    {
        return (int)(Math.random()*max);
    }
}
```