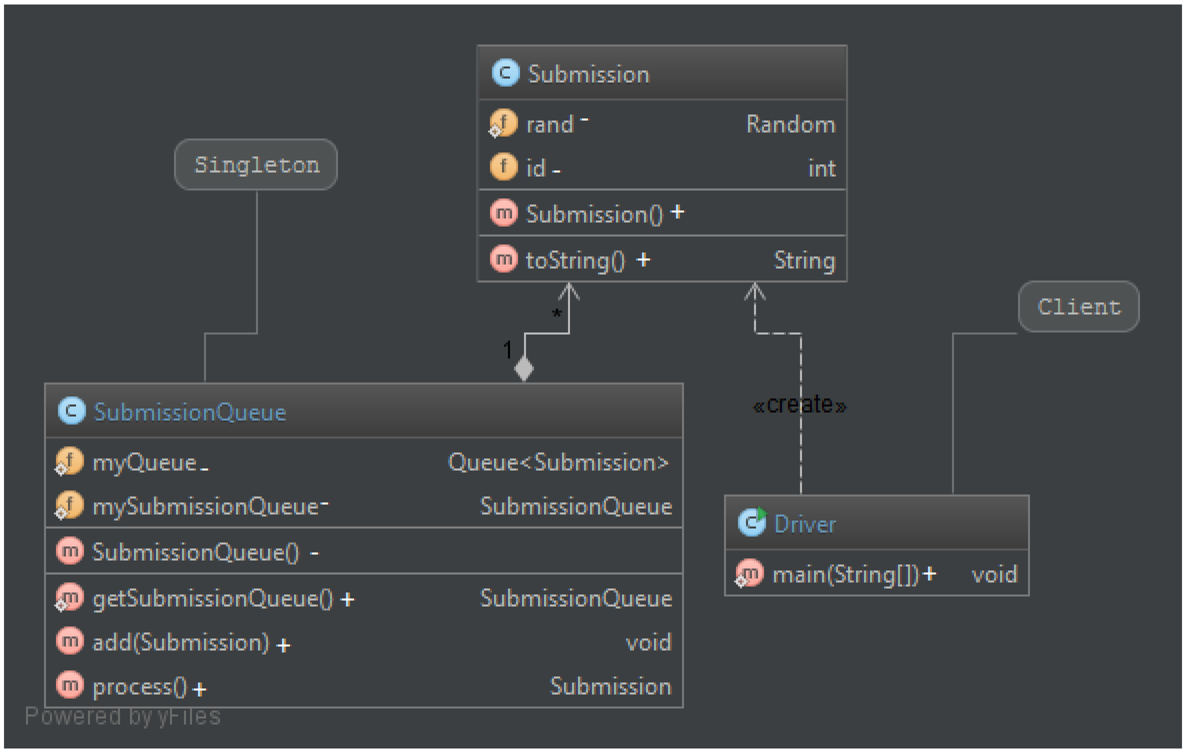
Brian Newsom & Dominic Tonozzi
CSCI 4448 HW3 - Design Patterns
Due 3-6-15

**Problem 1**

**Problem Number:**
Problem 1

**Design Pattern Used:**
We used the singleton design pattern because we wanted to have only one queue with a consistent state, allowing access from multiple threads without concurrency issues. This design pattern successfully ensures that there can never be multiple queues of submissions as the constructor is private, only allowing access from the getSubmissionQueue() method which will always return the same instance (or a new instance if none has been created) of the queue of submissions.

**Class diagram:**



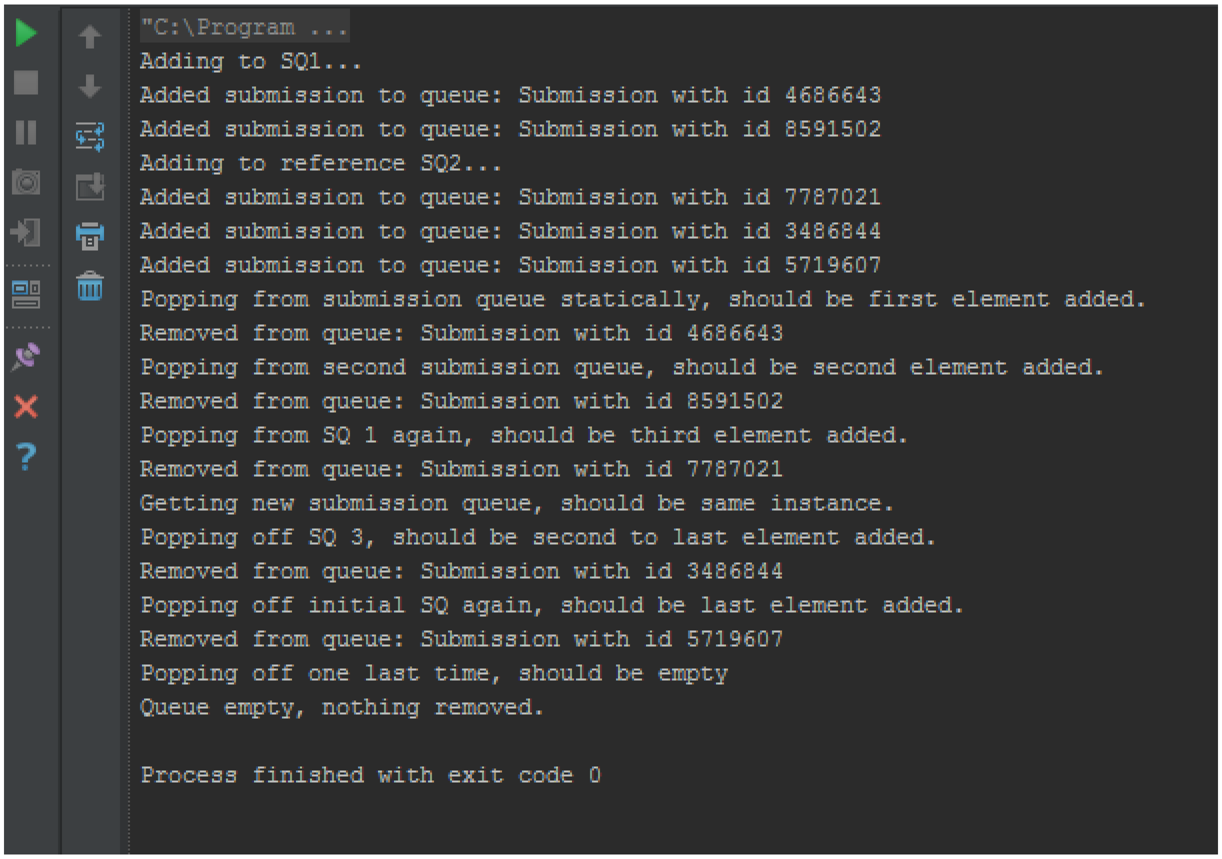
A simple class diagram of the important classes, note that the diamond indicates a static method.

**Participants:**
Singleton: SubmissionQueue
Client: Driver

**Contributors:**

|  |  |  |
|---|---|---|
| 60 % | Brian Newsom | Setup singleton, made diagrams |
| 40 % | Dominic Tonozzi | revised singleton, helped with driver |

**Example Run:**



```
"C:\Program ...
Adding to SQ1...
Added submission to queue: Submission with id 4686643
Added submission to queue: Submission with id 8591502
Adding to reference SQ2...
Added submission to queue: Submission with id 7787021
Added submission to queue: Submission with id 3486844
Added submission to queue: Submission with id 5719607
Popping from submission queue statically, should be first element added.
Removed from queue: Submission with id 4686643
Popping from second submission queue, should be second element added.
Removed from queue: Submission with id 8591502
Popping from SQ 1 again, should be third element added.
Removed from queue: Submission with id 7787021
Getting new submission queue, should be same instance.
Popping off SQ 3, should be second to last element added.
Removed from queue: Submission with id 3486844
Popping off initial SQ again, should be last element added.
Removed from queue: Submission with id 5719607
Popping off one last time, should be empty
Queue empty, nothing removed.

Process finished with exit code 0
```

The above driver run displays the functionality of the singleton.  We continually try to get new submission queues but always end up with the singleton instance expected, adding and processing the same elements in the order we want.

**Code:**

```
////////////////////////Submission.Java//////////////////////
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);
   }
   public String toString(){
       // Cleaner printing for debugging
       return "Submission with id " + this.id;
   }
}
/////////////////////////SubmissionQueue.Java////////////////////////
import java.util.LinkedList;
import java.util.Queue;

public class SubmissionQueue {
   // Actual FIFO queue to store submissions
   private static Queue<Submission> myQueue;
   private static SubmissionQueue mySubmissionQueue;

   //Empty Constructor
   private SubmissionQueue() {
     myQueue = new LinkedList<Submission>();
   }


   public static SubmissionQueue getSubmissionQueue() {
       // If we really want to be correct I think we have to do some java magic here to allow
multithreading.
       if (mySubmissionQueue == null){
           mySubmissionQueue = new SubmissionQueue();
       }
       return mySubmissionQueue;
   }

   public void add(Submission s){
       getSubmissionQueue().myQueue.add(s);
       System.out.println("Added submission to queue: " + s);
   }

   public Submission process(){
       // Get next element
       Submission s = getSubmissionQueue().myQueue.poll();
       if(s == null){
           System.out.println("Queue empty, nothing removed.");
       } else {
           System.out.println("Removed from queue: " + s);
       }
       return s;
   }
}
/////////////////////////Driver.Java////////////////////////
public class Driver {

   public static void main(String[] args) {
       SubmissionQueue sq1 = SubmissionQueue.getSubmissionQueue();
       SubmissionQueue sq2 = SubmissionQueue.getSubmissionQueue();

       System.out.println("Adding to SQ1...");
       for(int i = 0 ; i < 2 ; i++){
           sq1.add(new Submission());
       }
       System.out.println("Adding to reference SQ2...");
       for(int i= 0 ; i < 3 ; i++){
           sq2.add(new Submission());
       }

       // Should equal first submission
       System.out.println("Popping from submission queue statically, should be first element
added.");
       sq1.process();

       System.out.println("Popping from second submission queue, should be second element
added.");
       sq2.process();
       System.out.println("Popping from SQ 1 again, should be third element added.");
       sq1.process();
       // Get new submission queue, should return the same instance
       System.out.println("Getting new submission queue, should be same instance.");
       SubmissionQueue sq3 = SubmissionQueue.getSubmissionQueue();
       System.out.println("Popping off SQ 3, should be second to last element added.");
       sq3.process();
       System.out.println("Popping off initial SQ again, should be last element added.");
       sq1.process();
       System.out.println("Popping off one last time, should be empty");
       sq2.process();
   }
}
```

## <u>Problem 2</u>

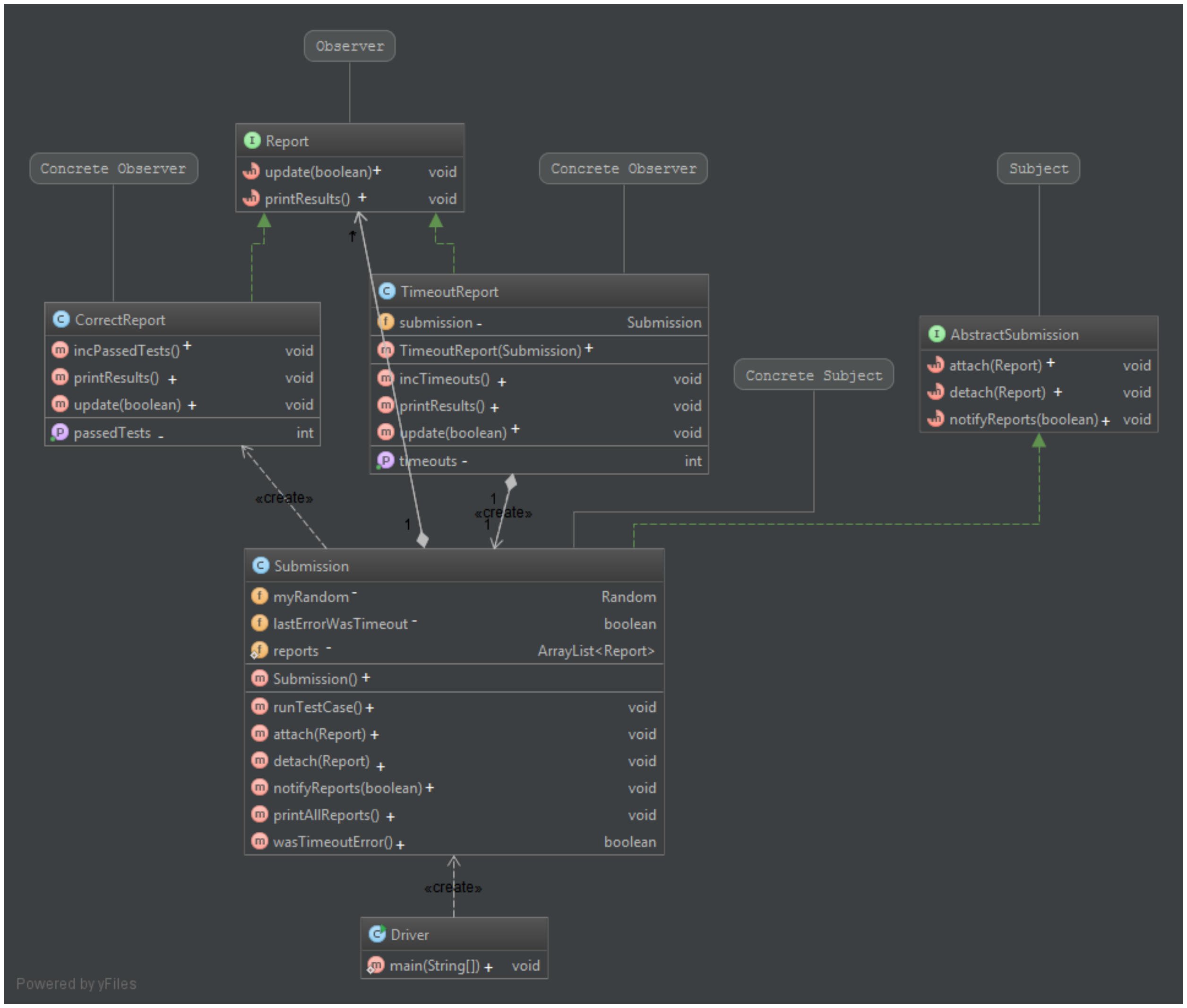
**Problem Number:**

Problem 2

**Design Pattern Used:**

We used the observer design pattern to facilitate communication and updating of the report and submission classes. This pattern allows the submissions to notify each report type on certain events, in this case the occurrence of each test. Then by passing a submission into the constructor of a TimeoutReport, we are able to access the most recent information from the subject from within the observer. This facilitates the necessary communication and meets the spec of the observer pattern exactly.

**Class diagram:**



A diagram of the observer design pattern used. Note that a diamond indicates a static method.

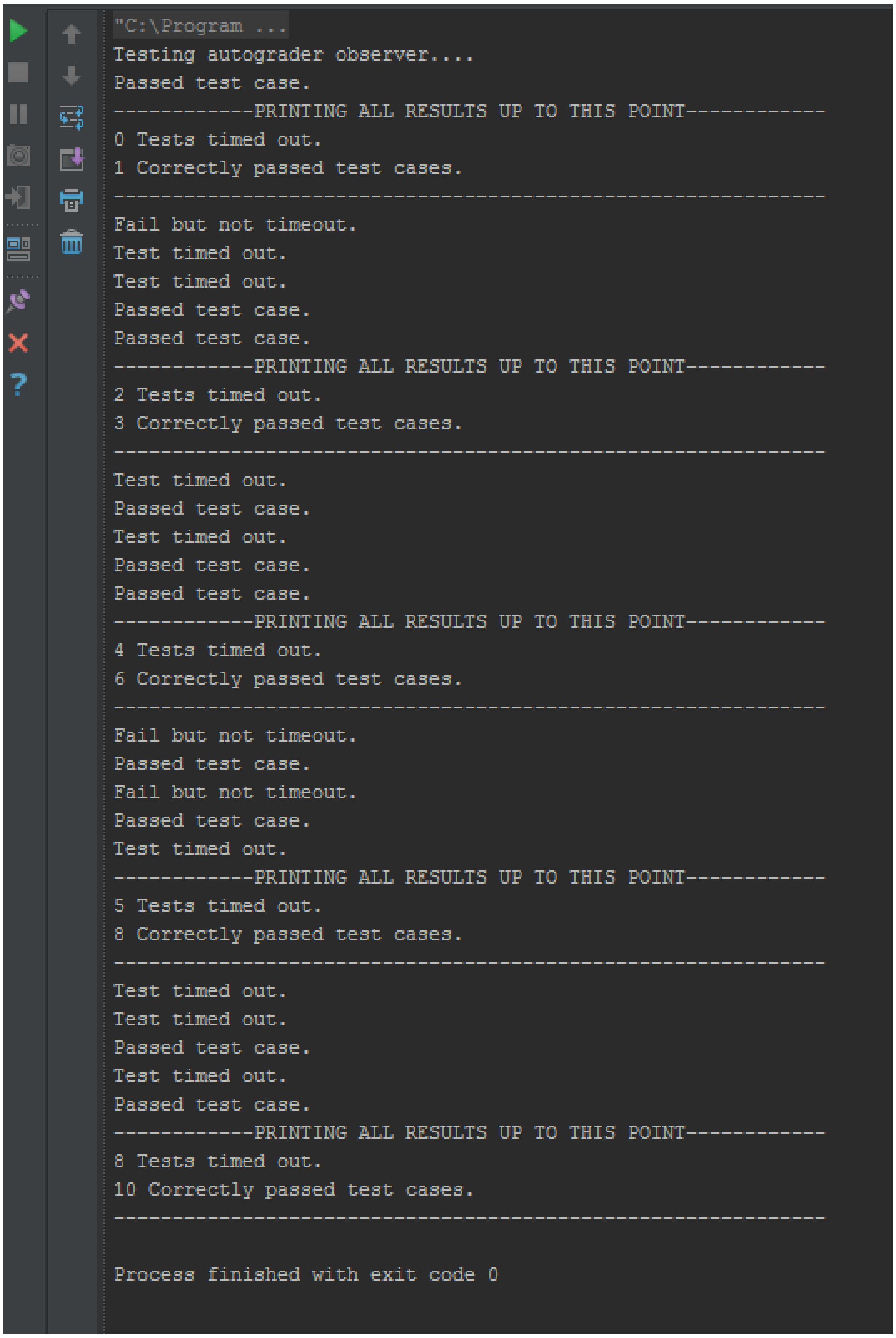
**Participants:**

Subject: AbstractSubmission
Concrete Subject: Submission
Observer: Report
Concrete Observers: CorrectReport, TimeoutReport

**Contributors:**

| | | |
|---|---|---|
| 60 % | Brian Newsom | Write observer pattern, diagrams |
| 40 % | Dominic Tonozzi | Pair programmed, driver |

**Example Run:**



```
"C:\Program ...
Testing autograder observer....
Passed test case.
------------PRINTING ALL RESULTS UP TO THIS POINT------------
0 Tests timed out.
1 Correctly passed test cases.
-------------------------------------------------------------
Fail but not timeout.
Test timed out.
Test timed out.
Passed test case.
Passed test case.
------------PRINTING ALL RESULTS UP TO THIS POINT------------
2 Tests timed out.
3 Correctly passed test cases.
-------------------------------------------------------------
Test timed out.
Passed test case.
Test timed out.
Passed test case.
Passed test case.
------------PRINTING ALL RESULTS UP TO THIS POINT------------
4 Tests timed out.
6 Correctly passed test cases.
-------------------------------------------------------------
Fail but not timeout.
Passed test case.
Fail but not timeout.
Passed test case.
Test timed out.
------------PRINTING ALL RESULTS UP TO THIS POINT------------
5 Tests timed out.
8 Correctly passed test cases.
-------------------------------------------------------------
Test timed out.
Test timed out.
Passed test case.
Test timed out.
Passed test case.
------------PRINTING ALL RESULTS UP TO THIS POINT------------
8 Tests timed out.
10 Correctly passed test cases.
-------------------------------------------------------------

Process finished with exit code 0
```

The above driver displays the correct functionality of the observer pattern implemented, as the submissions are continually run and notify the report types, and in the case of the TimeoutReport, it is successfully able to check if the last test was a time out, printing “fail but not timeout” in this case.  In then aggregates the results through a call to printResults() of the reports

**Code:**

```
/////////////////////////AbstractSubmission.Java////////////////////////

public interface AbstractSubmission {
   public void attach(Report r);
   public void detach(Report r);
   public void notifyReports(boolean pass);
}

/////////////////////////Submission.Java////////////////////////
/**
* Submission.java
*
* A representation of a Submission
*/

import java.util.ArrayList;
import java.util.Random;

public class Submission implements AbstractSubmission
{
   private Random myRandom;
  private boolean lastErrorWasTimeout;
   private static ArrayList<Report> reports = new ArrayList<Report>();

   // 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();
     }
```

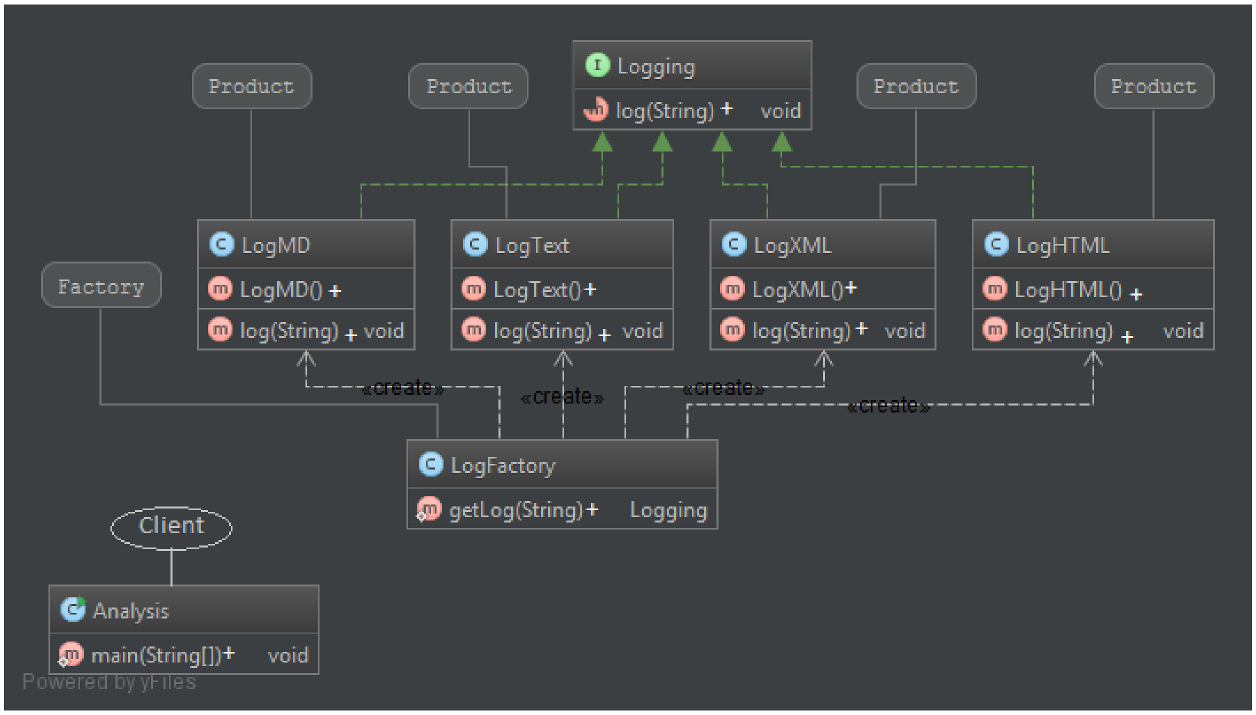
```
        // Notify in pass or fail case
        notifyReports(passed);
      // You can add to the end of this method for reporting purposes
   }

    public void attach(Report r){
        reports.add(r);
    }

    public void detach(Report r){
        reports.remove(r);
    }

    public void notifyReports(boolean pass){
        for(Report r : reports){
            r.update(pass);
        }
    }

    public void printAllReports(){
        System.out.println("------------PRINTING ALL RESULTS UP TO THIS POINT------------");
        for(Report r : reports){
            r.printResults();
        }
        System.out.println("-------------------------------------------------------------");
    }

    public boolean wasTimeoutError()
   {
       return lastErrorWasTimeout;
   }
}

////////////////////////Report.Java///////////////////////

// Observer
public interface Report {
   // Generic data for each report.
   public void update(boolean pass);
   public void printResults();

}

////////////////////////TimeoutReport.Java///////////////////////

public class TimeoutReport implements Report {
   private int timeouts = 0;
   private Submission submission;

   public TimeoutReport(Submission s){
       this.submission = s;
   }

   public void incTimeouts(){
       System.out.println("Test timed out.");
       timeouts++;
   }

   public int getTimeouts(){
       return timeouts;
   }

   public void printResults(){
       System.out.println(this.getTimeouts() + " Tests timed out.");
   }

   public void update(boolean pass){
       if(!pass){
           // Check if timeout error
           boolean timeout = this.submission.wasTimeoutError();
           if(timeout){
               incTimeouts();
           } else{
               System.out.println("Fail but not timeout.");
           }
       }
   }

}

/////////////////////////CorrectReport.Java///////////////////////

public class CorrectReport implements Report {
   private int passedTests = 0;

   public int getPassedTests() {
       return passedTests;
   }

   public void incPassedTests() {
       System.out.println("Passed test case.");
       this.passedTests++;
   }

   public void printResults() {
       System.out.println(this.getPassedTests() + " Correctly passed test cases.");
   }

   public void update(boolean pass){
       if(pass){
           incPassedTests();
       }
   }

}

/////////////////////////Driver.Java////////////////////////

public class Driver {

   public static void main(String[] args) {
       System.out.println("Testing autograder observer....");
       Submission s = new Submission();
       Report cr = new CorrectReport();
       Report tor = new TimeoutReport(s);
       s.attach(cr);
       s.attach(tor);
       for (int i = 0 ; i < 21 ; i++){
           s.runTestCase();
           if(i % 5 == 0) s.printAllReports();
       }
   }
}
```

## Problem 3

**Design Pattern Used:**

We used the factory pattern to abstract the logic concerning the format of the log file into the factory class. Now the user of the factory (main in this case) simply calls the factory method with the string representing the type of log, and doesn't need to concern itself with formatting details.

**Class diagram:**



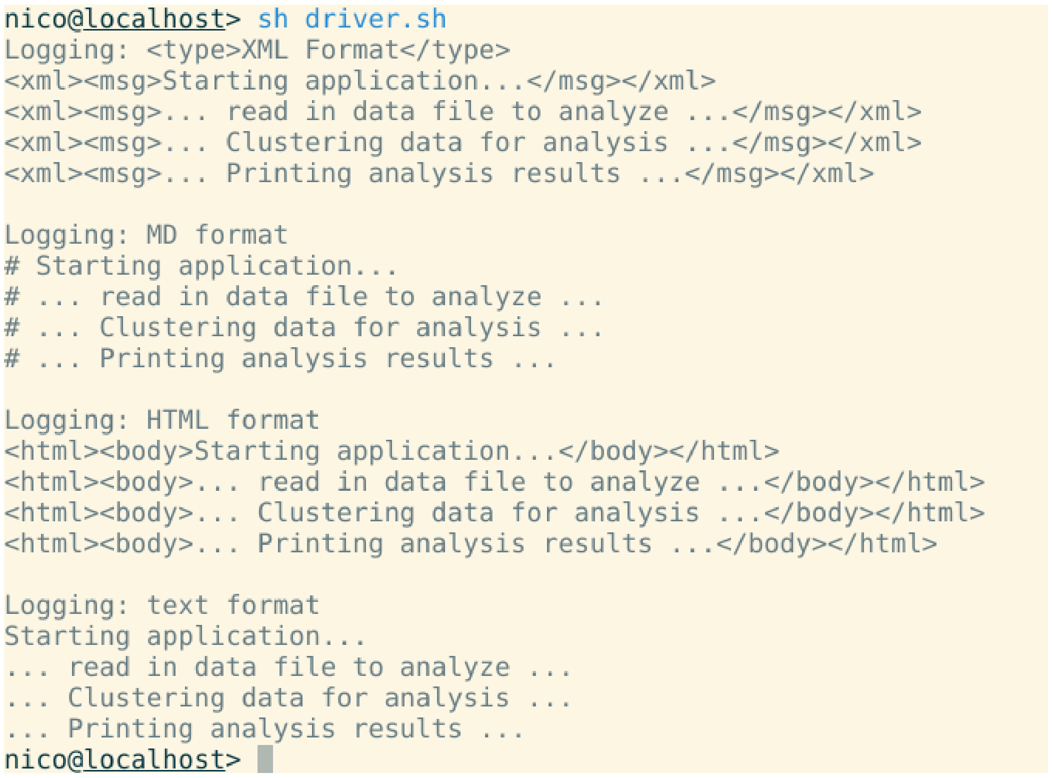
The class diagram for the factory pattern - notice the LogFactory class creates all realizations of Logging.

**Participants:**
Client: Analysis
Factory: LogFactory
Products: LogXML, LogHTML, LogMD, LogText. All of these classes implement the logging interface.

**Contributors:**

| | | |
|---|---|---|
| 50 % | Brian Newsom | Create outline of code, diagrams |
| 50 % | Dominic Tonozzi | Implement factory pattern, write driver, test code |

**Example Run:**



```
nico@localhost> sh driver.sh
Logging: <type>XML Format</type>
<xml><msg>Starting application...</msg></xml>
<xml><msg>... read in data file to analyze ...</msg></xml>
<xml><msg>... Clustering data for analysis ...</msg></xml>
<xml><msg>... Printing analysis results ...</msg></xml>

Logging: MD format
# Starting application...
# ... read in data file to analyze ...
# ... Clustering data for analysis ...
# ... Printing analysis results ...

Logging: HTML format
<html><body>Starting application...</body></html>
<html><body>... read in data file to analyze ...</body></html>
<html><body>... Clustering data for analysis ...</body></html>
<html><body>... Printing analysis results ...</body></html>

Logging: text format
Starting application...
... read in data file to analyze ...
... Clustering data for analysis ...
... Printing analysis results ...
nico@localhost> 
```

**Code:**

File Analysis.java:

```
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(msg);
        }
}
class LogXML implements Logging
{
        public LogXML()
        {
                System.out.println("Logging: <type>XML Format</type>");
        }
        public void log(String msg)
        {
                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("<html><body>"+msg+"</body></html>");
        }
}
class LogMD implements Logging
{
        public LogMD(){
                System.out.println("Logging: MD format");
        }
        public void log(String msg)
        {
                System.out.println("# "+msg);
        }
}

class LogFactory{
        public static Logging getLog(String type){
        Logging logfile;
                if (type.equalsIgnoreCase("xml"))
                        logfile = new LogXML();
                else if (type.equalsIgnoreCase("html"))
                        logfile = new LogHTML();
                else if (type.equalsIgnoreCase("md"))
                        logfile = new LogMD();
                else
                        logfile = new LogText();
                return logfile;
        }
}

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 = LogFactory.getLog(type);

                logfile.log("Starting application...");
                logfile.log("... read in data file to analyze ...");
                logfile.log("... Clustering data for analysis ...");
                logfile.log("... Printing analysis results ...");
        }
}
```

File driver.sh:

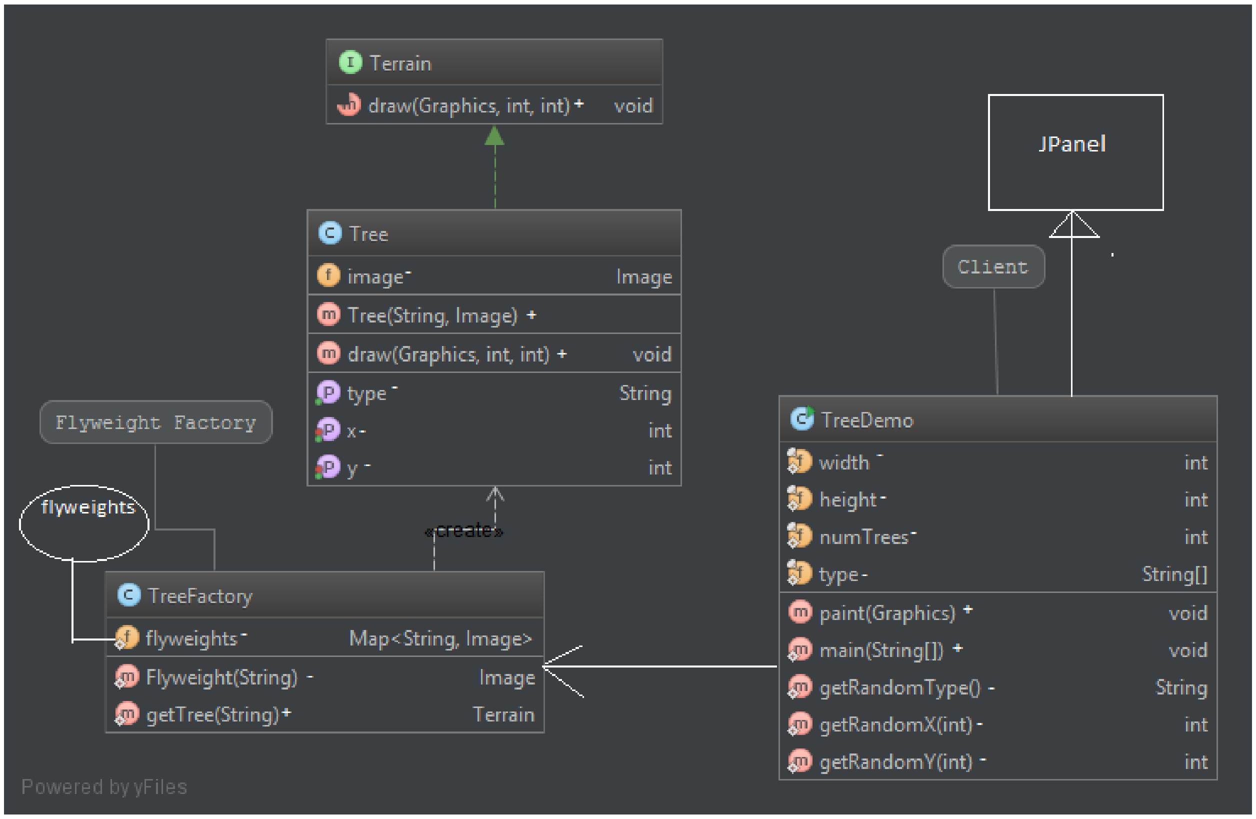
```
#!/usr/bin/env bash
javac Analysis.java
java Analysis xml
echo
java Analysis md
echo
java Analysis html
echo
java Analysis asdf
```

__**Problem 4**__

**Design Pattern Used:**

We used the flyweight pattern in this example to allow us a to use thousands of objects, without duplicating the IO and memory costs of reading an image thousands of times. The intrinsic state that can be shared between the trees are the images. The Trees still have their own objects, with important and possibly unique information like position. It's pretty challenging to show that the design pattern is effective with a screenshot, but it's much quicker now. To benchmark the change, I changed the number of trees from 50 trees to 5000. It takes 1.904 seconds to create 5000 trees with the flyweight design pattern, but 14.760 seconds with the original version.

**Class diagram:**



The class diagram is included here for the flyweight pattern. Notice that the symbol in the upper left corner of a method or object indicates final. Notice also that JPanel is blackboxed as TreeDemo extends a compiled library.

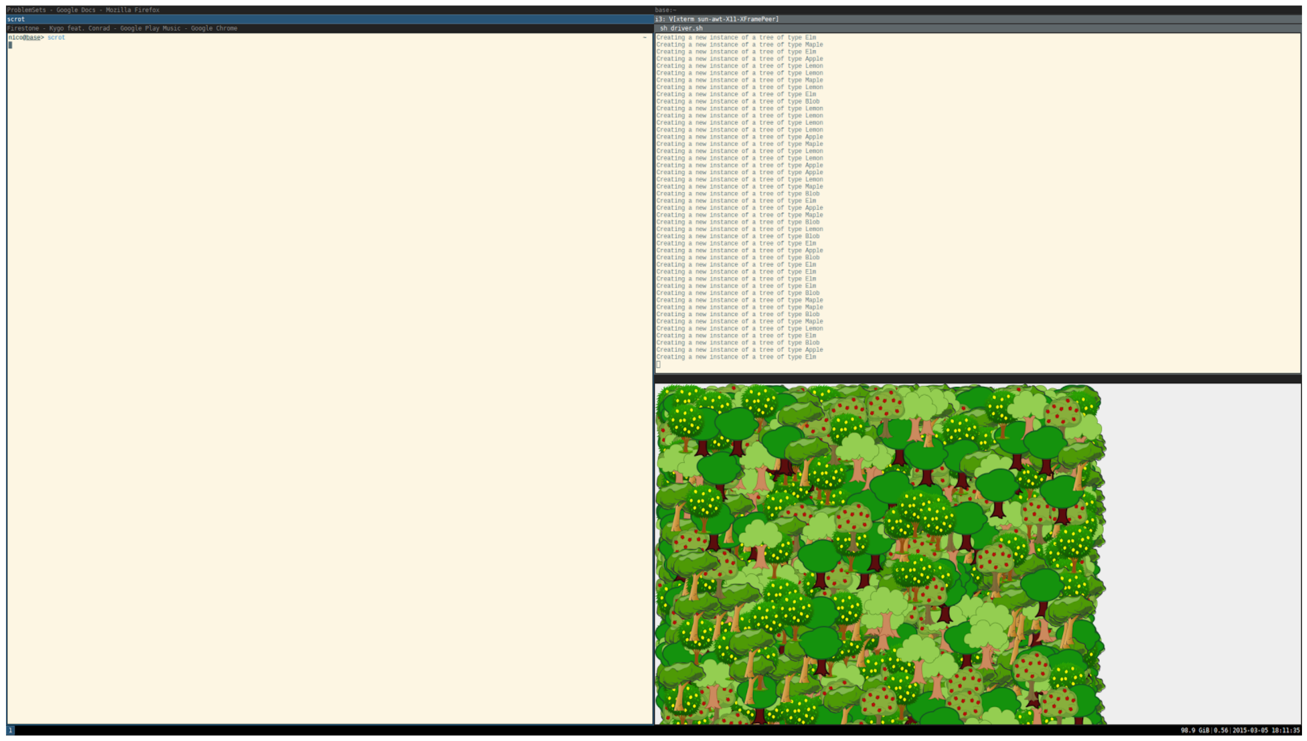
**Participants:**

Client: TreeDemo
Flyweight Factory: TreeFactory
Concrete Flyweight: flyweights

**Contributors:**

|  |  |  |
|---|---|---|
| 40 % | Brian Newsom | Help with driver, diagrams |
| 60 % | Dominic Tonozzi | Write core design pattern, explain conceptually |

## Example Run:


ProblemSets - Google Docs - Mozilla Firefox
scrot
Firestone - Kygo feat. Conrad - Google Play Music - Google Chrome
base:~
i3: V[xterm sun-awt-X11-XFramePeer]
sh driver.sh
98.9 GiB | 0.56 | 2015-03-05 18:11:35

```
nico@base> scrot
```

```
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Maple
Creating a new instance of a tree of type Lemon
Creating a new instance of a tree of type Elm
Creating a new instance of a tree of type Blob
Creating a new instance of a tree of type Apple
Creating a new instance of a tree of type Elm
```

## Code:

```
File TreeDemo.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;
    private String type;
    public Tree(String type, Image image)
    {
        System.out.println("Creating a new instance of a tree of type " + type);
        this.type = type;
        this.image = image;
    }
    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; }
    public String getType(){
        return type;
    }
    @Override
    public void draw(Graphics graphics, int x, int y)
    {
        graphics.drawImage(image, x, y, null);
    }
}

class TreeFactory
{
    private static Map<String, Image> flyweights = new HashMap<String, Image>();
    private static Image Flyweight (String type){
        String filename = "tree" + type + ".png";
        Image image = flyweights.get(filename);
        if (image == null) {
            try {
                image = ImageIO.read(new File(filename));
                flyweights.put(filename, image);
            } catch(Exception exc) {
                image = null;
            }
        }
        return image;
    }
    public static Terrain getTree(String type)
    {
        Image image = Flyweight(type);
        Tree tree = new Tree(type, image);
        return tree;
    }
}

/**
 * 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);
    }
}
```

File driver.sh:

```
#!/usr/bin/env bash
javac TreeDemo.java
time java TreeDemo
```