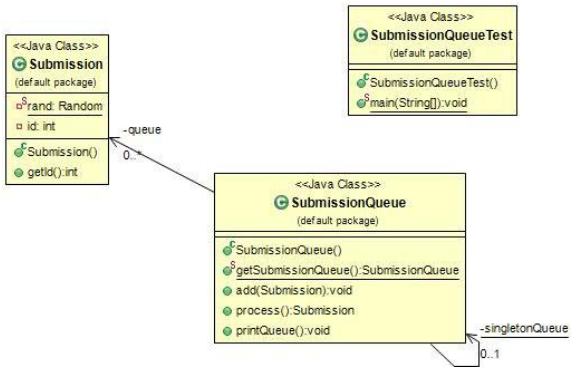
1. We employed the Singleton pattern to ensure that only one submission queue was created. The queue was implemented as a linked list. The SubmissionQueue contained the singleton pattern in the GetSubmissionQueue() method.



33.3%	Jeremy Granger	Help via peer-programming for coding and debugging
33.3%	Chris Jordan	Majority of coding
33.3%	Dan Palmer	Write-up and idea-bouncing

Submission.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);
	}
	public int getId()
	{
		return id;
	 }
}
```

SubmissionQueue.java

```
import java.util.*;

public class SubmissionQueue
{
	private Queue<Submission> queue;
	private static SubmissionQueue singletonQueue;
```

```
	public SubmissionQueue(){
		queue = new LinkedList<Submission>();
	}

	public static SubmissionQueue getSubmissionQueue()
	{
		if(singletonQueue == null)
			singletonQueue = new SubmissionQueue();
		return singletonQueue;
	}

	public void add(Submission s)
	{
		queue.add(s);
	}

	public Submission process()
	{
		return queue.remove();
	}

	public void printQueue()
	{
		Iterator<Submission> iter = queue.iterator();
		int index = 0;

		while(iter.hasNext())
		{
			Submission iterSub = iter.next();
			System.out.println("Element " + index + " in queue: " + iterSub.getId());
			index++;
		}
	}

}
```

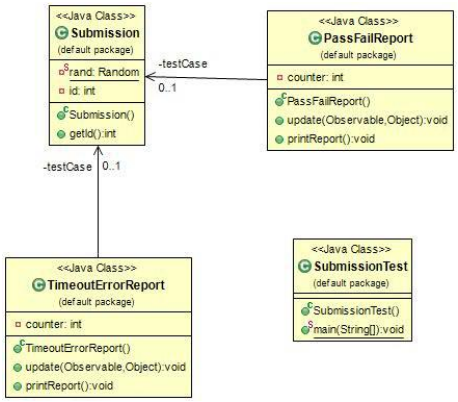
SubmissionQueueTest.java

```
public class SubmissionQueueTest
{
	public static void main(String[] args)
	{
		//Create three new assignments with class "Submission"
		Submission assignment1 = new Submission();
		Submission assignment2 = new Submission();
		Submission assignment3 = new Submission();

		//Create three new submission queues with class "SubmissionQueue"
		//This also tests the SubmissionQueue as a singleton
		SubmissionQueue subQueue1 = SubmissionQueue.getSubmissionQueue();
		SubmissionQueue subQueue2 = SubmissionQueue.getSubmissionQueue();
		SubmissionQueue subQueue3 = SubmissionQueue.getSubmissionQueue();

		//Add each assignment to each subQueue
		subQueue1.add(assignment1);
		subQueue2.add(assignment2);
		subQueue3.add(assignment3);

		System.out.println("subQueue1: ");
		subQueue1.printQueue();

		//Test the process() method for "SubmissionQueue"
		System.out.println("\nAssignment 4 is processed from subQueue3 using 'process()':");
		Submission assignment4 = subQueue3.process();
```

```
                System.out.println("Assignment 4 has id: " + assignment4.getId());
                System.out.println("Assignment 1 has id: " + assignment4.getId());

                if(assignment4 == assignment1)
                        System.out.println("Assignment 1 and Assignment 4 are equal!");

                System.out.println("\nsubQueue3: ");
                subQueue1.printQueue();

                System.out.println("\nTesting the subQueues to verify Singleton 'Submission Queue' is working:");
                if(subQueue1 == subQueue2 && subQueue2 == subQueue3)
                        System.out.println("\tAll queues equal!");

                //Here are all of the assignment ID's:
                System.out.println("Here are all assinments by ID: ");
                System.out.println("\tAssignment1 has id: " + assignment1.getId());
                System.out.println("\tAssignment2 has id: " + assignment2.getId());
                System.out.println("\tAssignment3 has id: " + assignment3.getId());
                System.out.println("\tAssignment4 has id: " + assignment4.getId());
        }
}
```

---

2. We used the observer pattern which allowed our submissionTest to pass information to PassFailReport and TimeoutErrorReport. PassFailReport kept track of the number of tests passed, while TimeoutErrorReport tracked the number of timeout errors. PassFailReport and TimeoutErrorReport are both observers of SubmissionTest.



33.3%	Jeremy Granger	Majority of coding
33.3%	Chris Jordan	Debugging
33.3%	Dan Palmer	Peer-Programming and write-up

Submission.java

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

import java.util.Random;
import java.util.Observable;

public class Submission extends Observable
{
   private Random myRandom;
            private boolean lastErrorWasTimeout;
            private boolean passed;

   // 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
                        passed = myRandom.nextBoolean();
                        if(!passed)
                        {
                           lastErrorWasTimeout = myRandom.nextBoolean();
                        }

                        setChanged();
                        notifyObservers();
            }

   public boolean wasTimeoutError()
            {
               return lastErrorWasTimeout;
            }

            public boolean didPass()
            {
                        return passed;
            }
}
```

PassFailReport.java

```
import java.util.Observer;
import java.util.Observable;

public class PassFailReport implements Observer
{
            private int counter;
            private Submission testCase;

            public PassFailReport()
            {
                        counter = 0;
            }
```

```
	@Override
	public void update(Observable obj, Object arg)
	{
		testCase = (Submission)obj;
		if(testCase.didPass())
		{
			counter++;
		}
	}

	public void printReport()
	{
		System.out.println("***Begin Pass/Fail Report***");
		System.out.println("The submission contained " + counter + " passing test cases.");
		System.out.println("***End Pass/Fail Report***\n");
	}
}
```

TimeoutErrorReport.java

```
import java.util.Observer;
import java.util.Observable;

public class TimeoutErrorReport implements Observer
{
	private int counter;
	private Submission testCase;

	public TimeoutErrorReport()
	{
		counter = 0;
	}

	@Override
	public void update(Observable obj, Object arg)
	{
		testCase = (Submission)obj;
		if(!testCase.didPass())
		{
			if(testCase.wasTimeoutError())
				counter++;
		}
	}

	public void printReport()
	{
		System.out.println("***Begin Timeout Error Report***");
		System.out.println("The submission contained " + counter + " timeout errors.");
		System.out.println("***End Timeout Error Report***\n");
	}
}
```

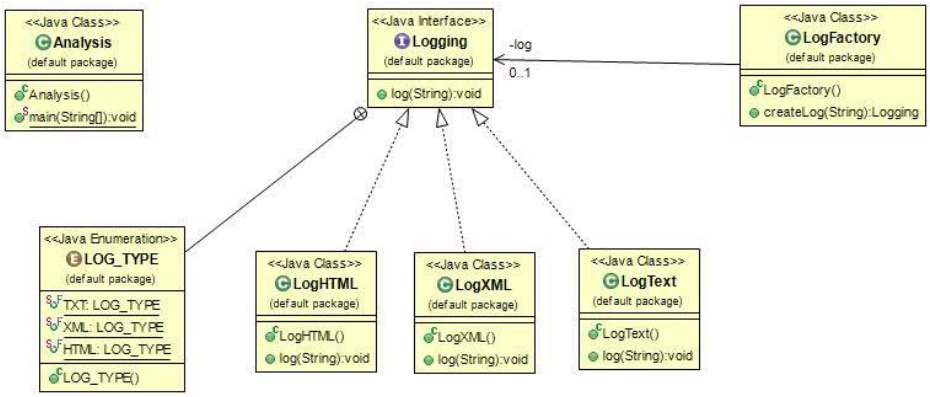
SubmissionTest.java

```
import java.util.*;

public class SubmissionTest
{
	public static void main(String[] args)
	{
		if (args.length != 1)
		{
```

```
                System.out.println("Usage: java SubmissionTest [# of tests to run]");
                System.exit(-1);
        }

        int numberOfTests = Integer.parseInt(args[0].toString());
        Submission test = new Submission();
        PassFailReport pfr = new PassFailReport();
        TimeoutErrorReport ter = new TimeoutErrorReport();

        test.addObserver(pfr);
        test.addObserver(ter);

        for(int i = 0; i < numberOfTests; i++)
        {
                test.runTestCase();
        }

        pfr.printReport();
        ter.printReport();

    }
}
```

---

3. Instead of determining which type of log file to create in the Analysis class, we used a factory to determine which type of log file to create. Depending on type, it either produces a text, xml, or html file. We added a variable LOG_TYPE to the Logging interface in order to define these types of files. The LogFactory class is the factory.



33.3%	Jeremy Granger	Peer-programming
33.3%	Chris Jordan	Peer-programming
33.3%	Dan Palmer	Peer-programming

Analysis.java

```
public 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];
		LogFactory factory = new LogFactory();
		Logging logfile = factory.createLog(type);

		logfile.log("Starting application...");

		System.out.println("... read in data file to analyze ...");
		// code...
		System.out.println("... Clustering data for analysis ...");
		// code...
		System.out.println("... Printing analysis results ...");
		// code...
	}
}
```

LogFactory.java

```
public class LogFactory {
	private Logging log;

	public Logging createLog(String type){
		if (type.equalsIgnoreCase("text"))
			log = new LogText();
		else if (type.equalsIgnoreCase("xml"))
			log = new LogXML();
		else if (type.equalsIgnoreCase("html"))
			log = new LogHTML();
		else
			log = new LogText();
		return log;
	}
}
```

Logging.java

```
public interface Logging {
	public enum LOG_TYPE {TXT, XML, HTML};
	public void log(String msg);
}
```

LogHTML.java

```
public 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>");
        }
}
```

LogText.java

```
public 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);
        }
}
```

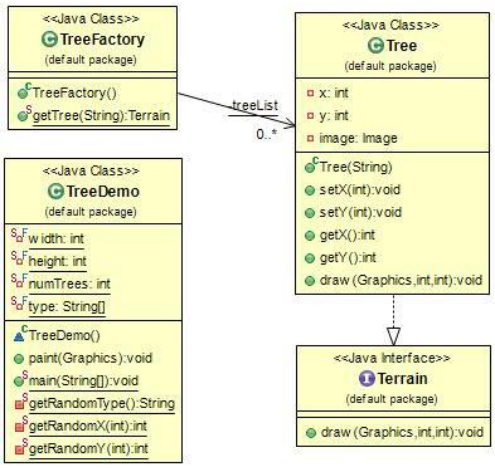
logXML.java

```
 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>");
        }
}
```

---

4. Since the main issue causing slow rendering of the trees was the fact that each tree had to be created individually, we used the flyweight pattern. Each time a new type of tree is rendered, it is stored in a hashmap. When we attempt to create a tree, we first check the hashmap to see if that tree type has already been created. If so, we do not have to create that type from scratch and rather use the tree from the hashmap. The flyweight pattern is employed in the TreeFactory class.

40%	Jeremy Granger	Peer-programming and debugging
40%	Chris Jordan	Majority of coding
20%	Dan Palmer	Write-up



Terrain.java

```
import java.awt.Graphics;

public interface Terrain {
	void draw(Graphics graphics, int x, int y);
}
```

Tree.java

```
import java.awt.Graphics;
import java.awt.Image;
import java.io.File;

import javax.imageio.ImageIO;

public 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; }

	public void draw(Graphics graphics, int x, int y)
	{
		graphics.drawImage(image, x, y, null);
	}
}
```

TreeDemo.java

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

/**
 * 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);
	}
}
```

TreeFactory.java

```
import java.util.*;

public class TreeFactory {
	private static Map<String, Tree> treeList = new HashMap<String, Tree>();

	public static Terrain getTree(String type)
	{
		Tree tree = treeList.get(type);

		if (tree == null){
			tree = new Tree(type);
			treeList.put(type, tree);
		}
```

```
                return tree;
    }
}
```