Report

Applied Software Engineering

Joe Sykes

21223675

Contents

[Introduction 2](#_Toc468648241)

[System Requirements 3](#_Toc468648242)

[Unified Modelling Language 4](#_Toc468648243)

[Use Case 4](#_Toc468648244)

[Class Diagram 5](#_Toc468648245)

[Sequence Diagram 6](#_Toc468648246)

[Architecture Diagram 8](#_Toc468648247)

[Implementation 10](#_Toc468648248)

[Use of design patterns 10](#_Toc468648249)

[How the system works 10](#_Toc468648250)

[How to run the software 11](#_Toc468648251)

[Evidence of running software 11](#_Toc468648252)

[With the wrong activation date: 11](#_Toc468648253)

[With the right activation date: 12](#_Toc468648254)

[Testing 13](#_Toc468648255)

[jUnit Testing 13](#_Toc468648256)

[System Testing 14](#_Toc468648257)

[Evaluation 15](#_Toc468648258)

[Code Listing 16](#_Toc468648259)

[Main.java 16](#_Toc468648260)

[MainSystem.java 18](#_Toc468648261)

[Region.java 19](#_Toc468648262)

[FactoryRegion.java 19](#_Toc468648263)

[Site.java 20](#_Toc468648264)

[NorthEast.java 21](#_Toc468648265)

[NorthWest.java 22](#_Toc468648266)

[SouthEast.java 22](#_Toc468648267)

[SouthWest.java 23](#_Toc468648268)

[Midlands.java 23](#_Toc468648269)

[London.java 24](#_Toc468648270)

[Tests.java 24](#_Toc468648271)

# Introduction

Saxon Heritage is a company that maintains buildings and structures on behalf of the government. These buildings and structures are called sites and they bring visitors every year who pay to see them. Each year the number of visitors at these sites must be counted and categorised. The lowest performing sites (the ones with the least visitors in the year) are prioritised for marketing campaigns in the following year to drum up attention and PR so people will be more inclined to visit them.

The system being described in this report works to automate this procedure. Every year when the date reaches 30th December, the new visitor numbers will be retrieved for each site. These sites will then be categorised either bronze, silver or gold depending on how many visitors it attracted in the past year. Any site that received a bronze rating will be categorised for a marketing campaign. This is all done automatically in the system using object oriented development and the java programming language.

This report will explain how the system links together, how certain areas of the system communicate with others and exactly what the system is doing. This will be demonstrated through UML diagrams. Testing will also be shown as well as providing a critical evaluation of the program.

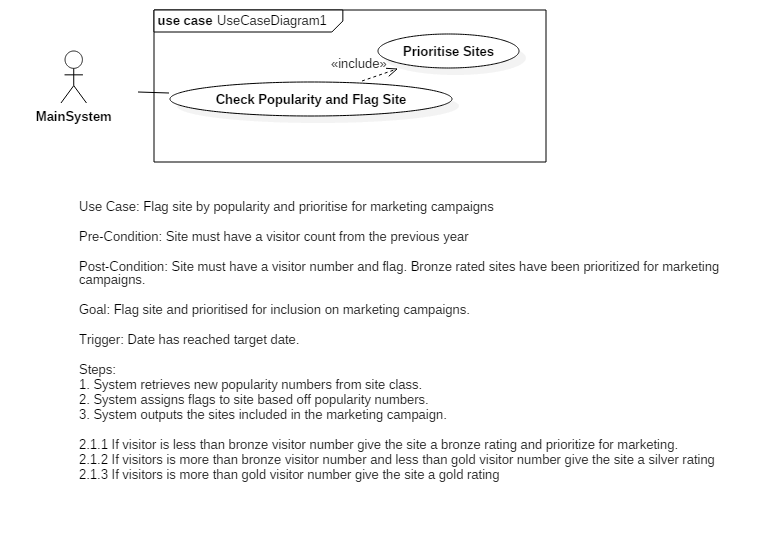
# System Requirements

The use case that has been selected for this assignment is updating the site popularity numbers, assigning them a rating based off those numbers and creating a priority marketing list for those sites with the lowest numbers. To do this the system is going to need a class to make the sites, a class to make the regions for the sites and array lists to hold the regions, sites and priority sites. It also needs to be able to update the visitor numbers in a site as well as linking a rating to the site based off these numbers.

The system must be display all the information to the user. This includes the current site information such as the name, visitor number and rating as well as the updated site information with the new visitor number and rating. It should also display the priority list with any sites that are rated bronze being included for an additional marketing campaign.

# Unified Modelling Language

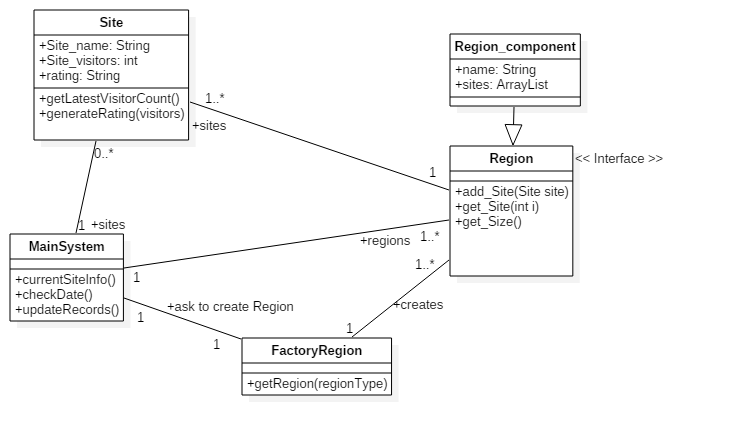
## Use Case



The use case in this system is to flag the site by popularity and priorities it for a marketing campaign. The main system is the actor in this scenario because it is an automated system that has no real user interaction therefore no actor can be used. The system has 3 steps to achieving the goal of the use case: retrieve the new popularity numbers, assign ratings based on off those numbers and add the sites to a priority list for the marketing campaign. The trigger for the use case is when the date reaches the target date which means a year has passed and the numbers need to be recounted.

The use case does not use any actual dates or number to aid in abstraction. This means the use case is still relevant and does not need to be changed if there are changes to the specific details of the case. For example, if the highest visitor number for a bronze classification were to change this would not need to be updated in the use case as it is simply listed as ‘bronze visitor number’.

## Class Diagram

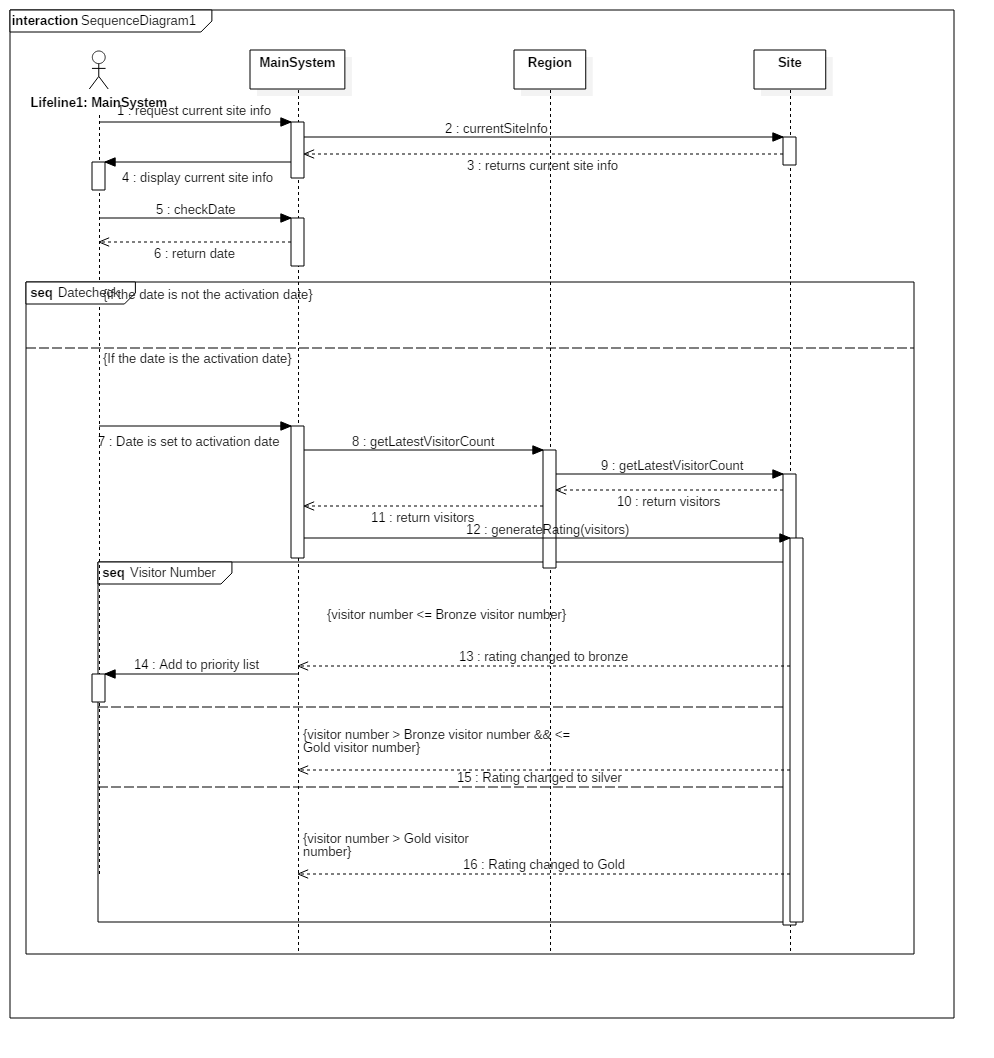


The class diagram for this system contains five separate classes. The main system class is where the main methods of the system are held, this includes getting the current site info, checking the current date and updating the records in the system. This links to the site class because it initiates the site object, it links to the regions class because it initiates the regions object and it links to the factor region class because it uses that to initiate the correct region object.

The site class contains the constructor for the site object consisting of a name, a number of visitors and a rating. It also has two methods associated with it. GetLatestVisitorCount() randomly generates a number that will be the number of visitors that have been to this site in the last year and returns the number. generateRating(visitors) will assign a rating to the site based off the visitor number.

The region class is an interface class which means it is a collection of abstract methods. An abstract method is one which can apply to and be used for a lot of different cases without having to be changed. The region class has three abstract methods within it: add\_Site, get\_Site and get\_Size. The add site method takes a site and adds it to a region. The get site method gets a site from a region and the get size method returns how many sites are in a region. The region class is an interface for the region component, which is not one specific class but rather a collection of classes that implement the region class. Each of these classes contain a name and an arraylist called sites. Each of these classes also contain the methods of the region class within them. Finally, the factory region class creates the region objects for the main system to use. This is part of the factory pattern design and helps with encapsulation because the region objects are being accessed via another class instead of being called directly.

## Sequence Diagram



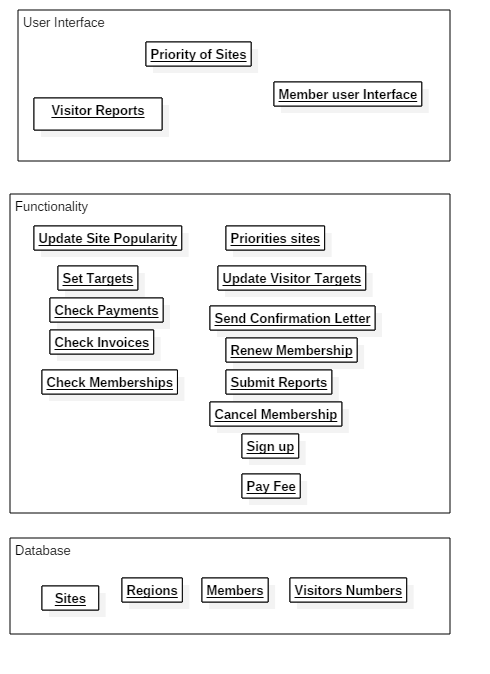
The sequence diagram is a chronological list of the events and processes that take place throughout the system life cycle. There are four life lines that represent each different element of the system and how they interact with each other. These lifelines are the main system class, the main system as an actor, the region class and the site class. The main system is where the main methods and processes that keep the process moving are stored while the actor is basically the trigger for these processes. An actor version of the system is used because this is an automated service that has no user interaction but it is easier to demonstrate how the system is triggered using the actor lifeline.

The region and site classes are basically just sources of information. This is where the main system sends messages to collect information about objects which is returned by these classes.

The system begins by requesting the current information of the sites from the site class, this is then returned to the system class where it is displayed for the user. The actor system then triggers the function to check whether the current date matches up with the trigger date. If it is not a match, the system terminates. If it is a match however the main system actor triggers the main system to get the latest visitor count for the sites. It does this by communicating with the site object and running the function to get the latest visitor count (which essentially just returns a random number). This number is then returned to the main system. It is then used to generate a rating for each site by using the generateRating method with a parameter for the number of visitors. The rating is then changed and the information returned to the main system.

This is where a combined fragment begins in the sequence diagram. Depending on the visitor number given to the site class a few different things can happen. If the visitor number is below the lowest threshold a bronze rating is given to the site and the main system then adds it to an array list called the priority list. If the visitor number is higher than the lowest threshold but lower than the highest threshold it is given a silver rating. Finally, if it higher than the highest threshold it is given a gold rating.

## Architecture Diagram



The architecture diagram represents the different areas of the system and how they interact with each other. In the Saxon system there are three different elements: the user interface, the functionality and the database. The user interface handles everything the client see’s. This includes the reports; the prioritisation of the sites being displayed and the user interface for the members to administrate their account.

The second layer to the system is the functionality. This includes any processes the system does such as renewing a membership, checking invoices, updating site popularity or sending a confirmation letter. These are all things the system must do either upon request or automatically. These are linked to the user interface layer because anything the user may select to be done will be processes in the functionality layer and any results from those processes will be displayed in the user interface layer.

The final layer of the system is the database layer which holds all the information for the entire system. This includes all the sites, regions members account information and the visitor numbers for each site and region. This layer is extremely important to the functionality layer because it will constantly call for information about certain elements such as sites or regions to complete its process.

# Implementation

## Use of design patterns

Design patterns help people understand how the program works and therefore speeds up the development of a program because others do not need to be ran through every details of the program to figure out how it is working. It also provides a paradigm to work by so even when creating an entirely new program, it has been done before and the problems and errors have already been experienced and resolved.

The system of this assignment is using the factory design pattern. I chose this pattern because it encapsulates the data very well and increases abstraction. The system only creates one instance of each region and doesn’t even do that directly, it does it through the factory class so the data is well encapsulated and cannot be manipulated or corrupted. Abstraction is increased because if the details of a region were to change for any reason it can be changed simply in the specific class and this will be updated everywhere throughout the system. Because the regions are created in other classes they do not have to be updates with the changes.

## How the system works

The system first creates the array lists, regions and sites before adding those sites to their respective regions. It then calls the currentSiteInfo() method which runs a for loop to get each instance of a region. With each region, it runs a second for loop to get the current site and displays it to the user. Once this process is finished t runs the check date function that checks if the date provided matches the trigger date returning a Boolean.

If the Boolean returns true, the updateRecords() method is ran which gets the sites in the same way as the currentSiteInfo() method but updates the site using the getLatestVisitorCount() method which generates a random number to the site to simulate the number of visitors in the last year. After this it gets the site rating and runs the getSite\_rating() function that will apply a rating to it depending on what the visitor number is. Finally, it runs an if function to check if the site rating is bronze and if so adds it to the priority array list. It also displays the information of the new updated site to the user and displays the priority sites as well.

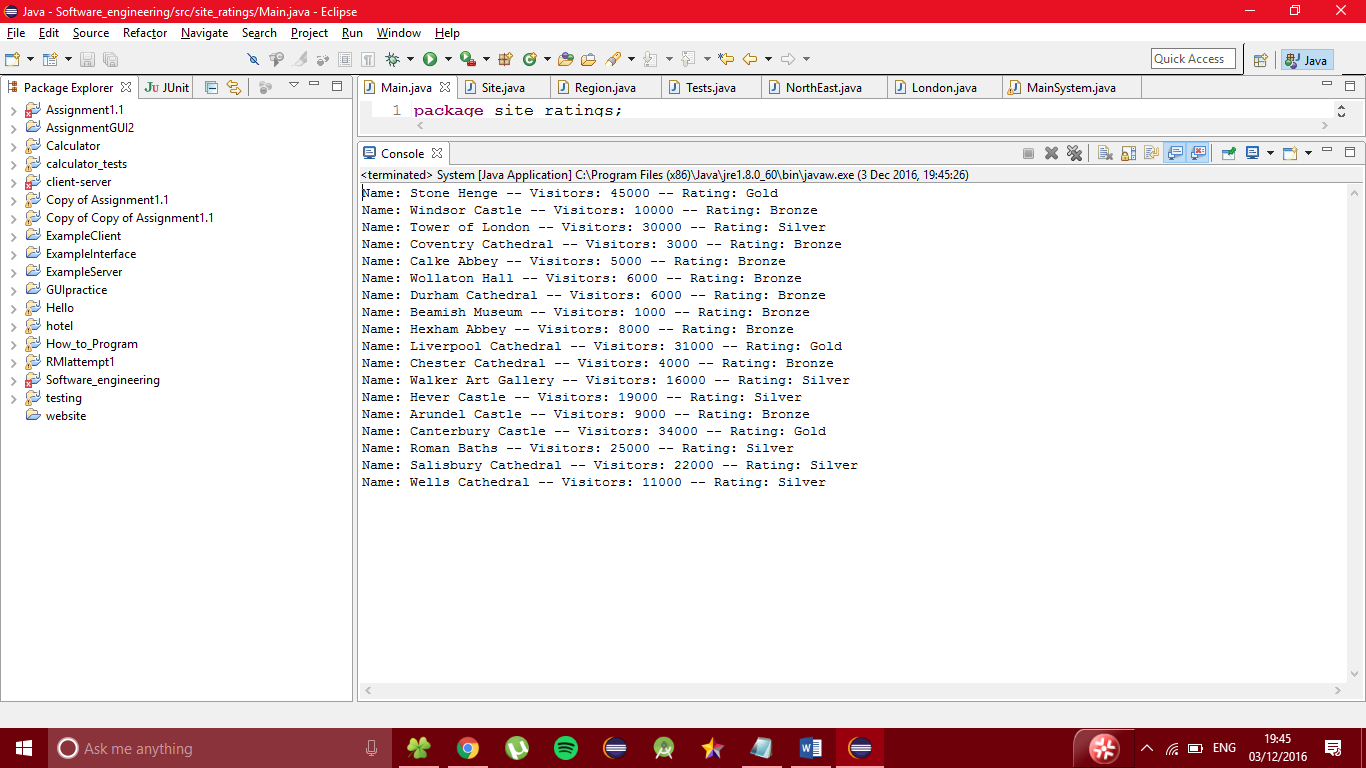
The sites use a class with a constructor which is called by the program to create it and the regions are created by calling a region factory that calls a specific region. It does this because the region class is just an interface that holds all of the other abstract methods for each region. The factory then call the region and creates an instance of an object of it which it used in the system. This is the factory design pattern.

## How to run the software

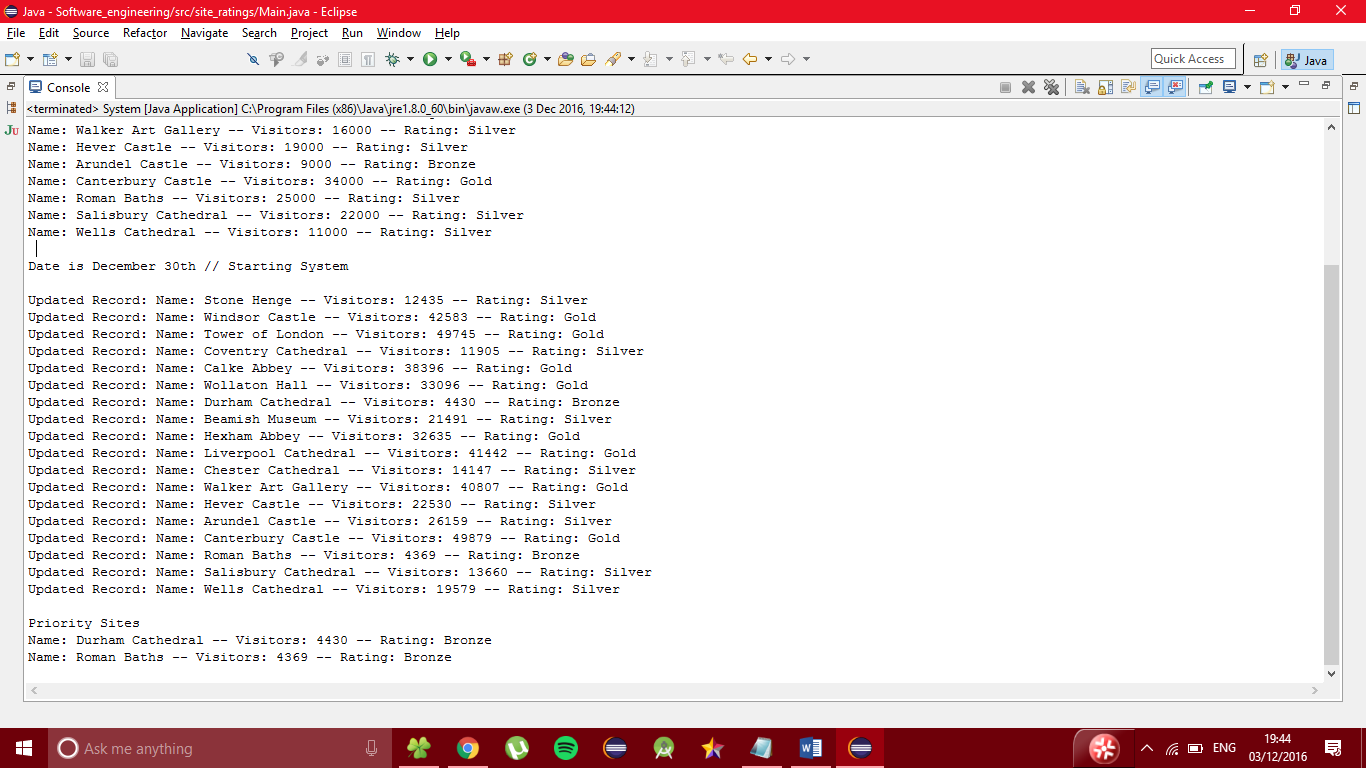
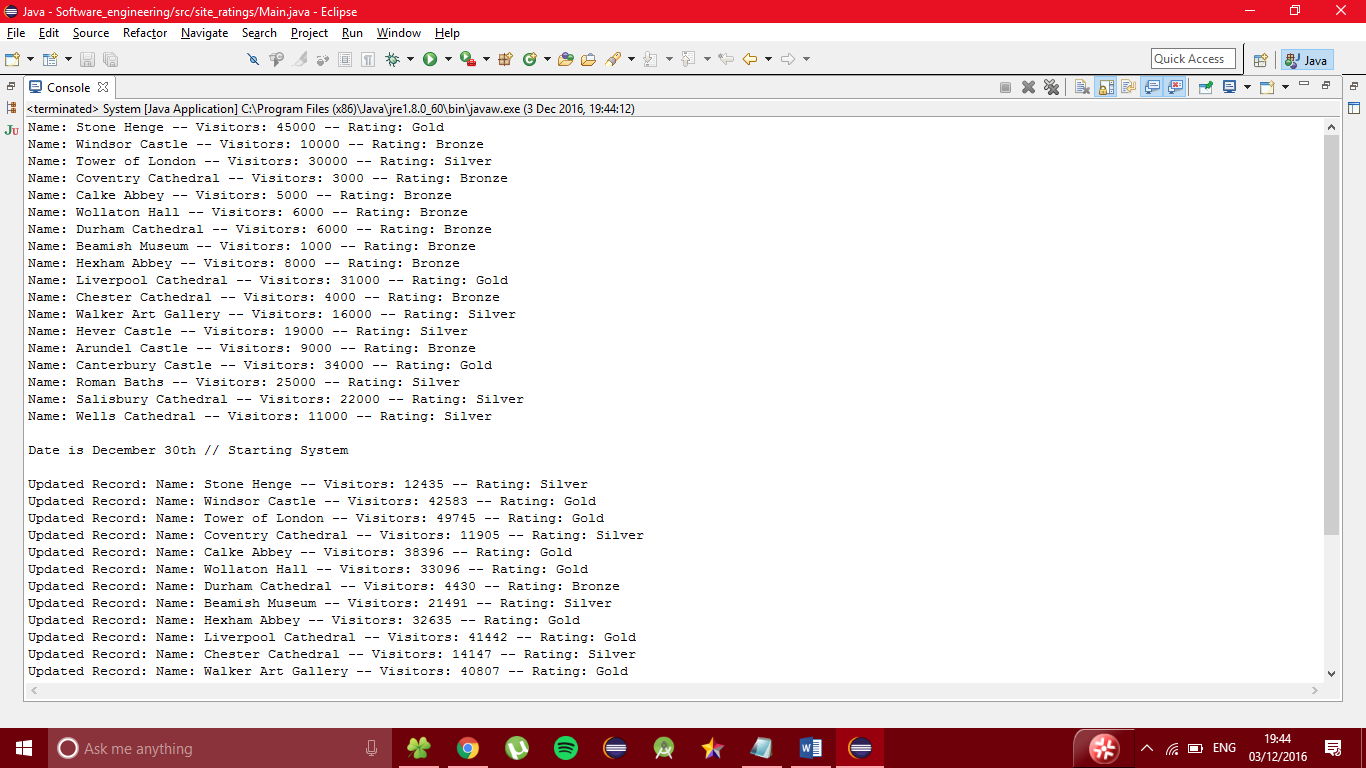
The software is ran using the java language and can be compiled and ran using any java compiling method. The system will run differently depending what the date entered is. If the date entered it the same as the trigger date it will update all of the sites with new number and ratings and display them.

## Evidence of running software

### With the wrong activation date:



### With the right activation date:

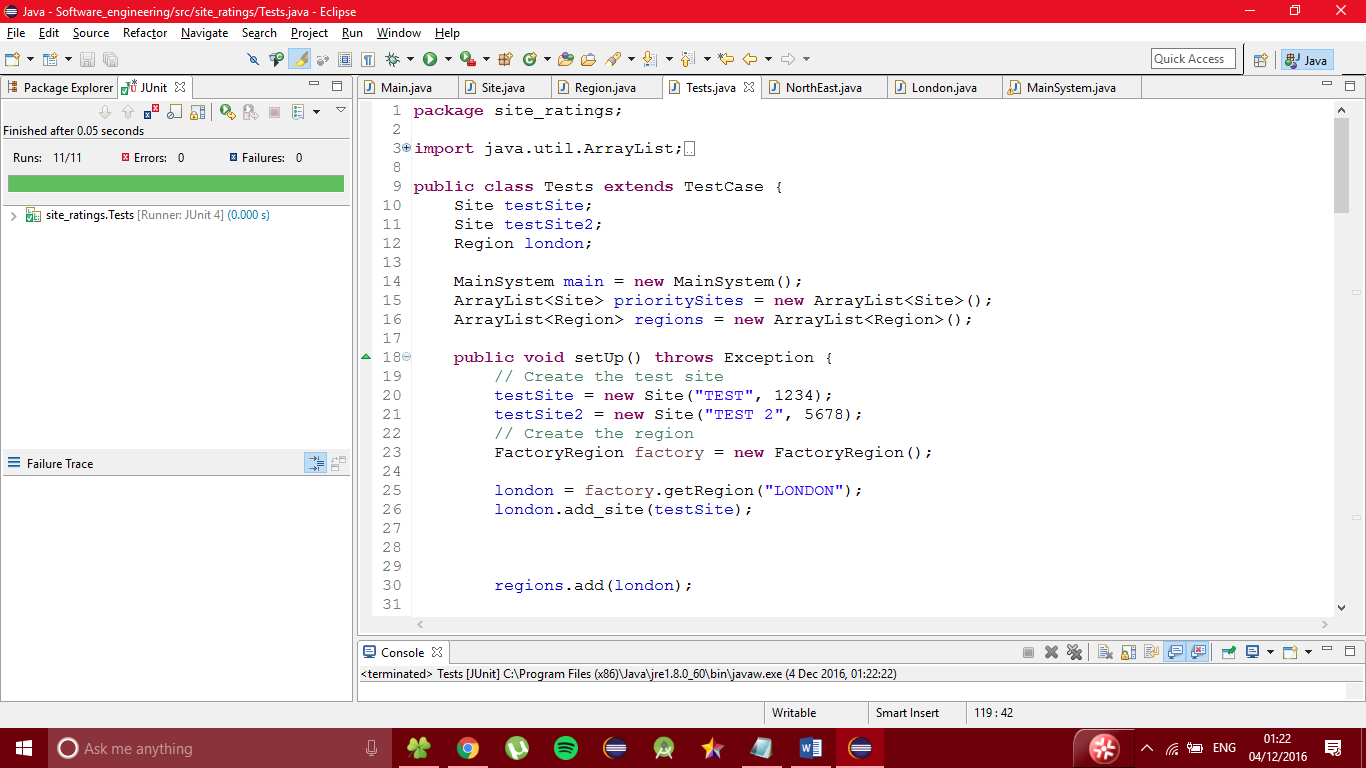


# Testing

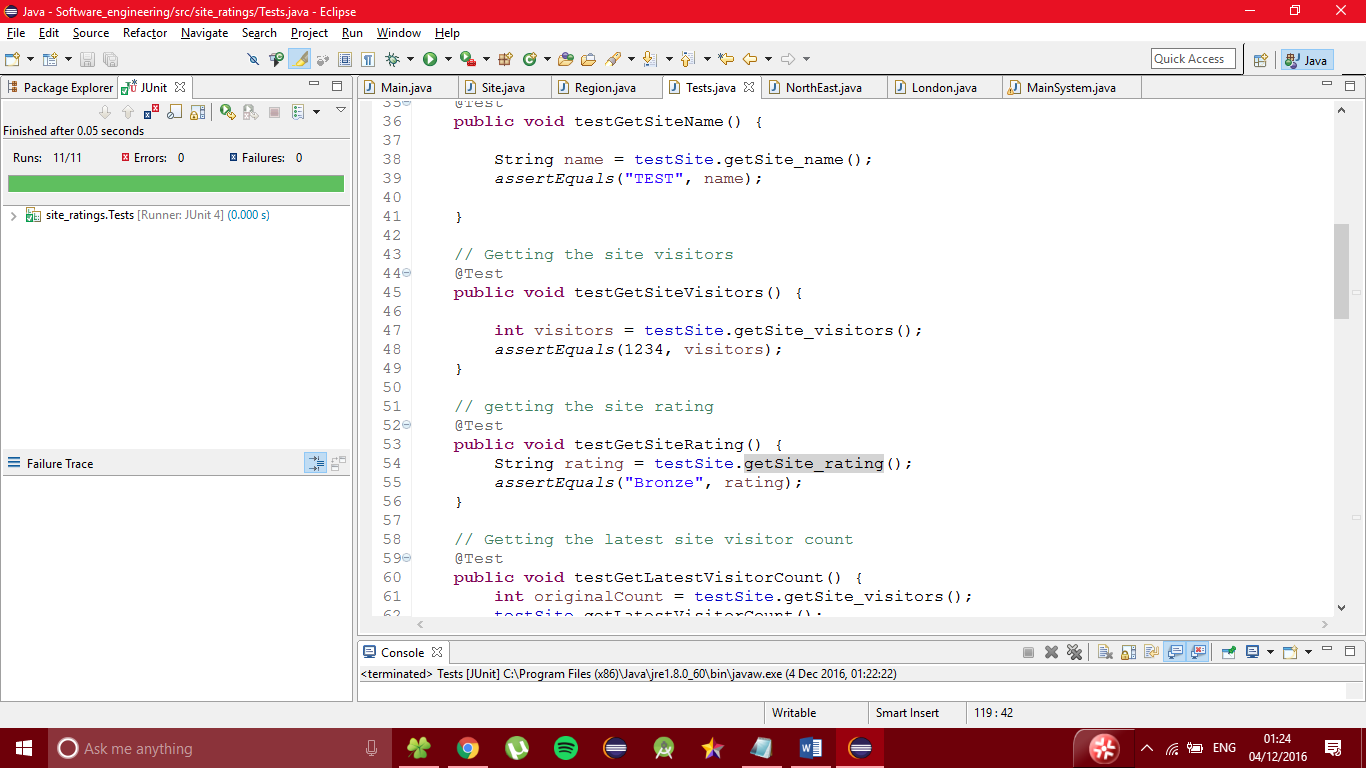
## jUnit Testing

Junit testing tests the different methods of the system using simple true or false statements. This is needed to make sure each aspect of the system is working properly so if the system isn’t functioning correctly you can check the Junit tests to quickly see what went wrong. This is called test driven development (TDD), the basic process is to develop code and test it at the same time. This means when a new feature or element is added to the system, a test is created to make sure it is working correctly and if it is not the code must be corrected, a new test written and tested again.

In this system, I tested every method of every class apart from the setters and getters.



The tests are setup using a setup method. This creates the sites and regions as well as establishing a main system object and the array lists. After this the tests can begin using this logic.



These are typical tests measuring the retrieval of the site name, getting the site visitors and getting the site rating. The test will be a void method and will have a function at the end of it such as assert equals with two parameters following it. This means if these two parameters are the same, the test will pass otherwise it will fail. All the tests run like this or in a very similar fashion. In total, there are 11 tests all of which run as soon as the system is compiled and ran.

## System Testing

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Result** | **Actual Result** | **Success?** |
| Add site to region | The site will now be included in the region array list | The site is now part of the region array list | Yes |
| Add region to regions array list | The region will now be included in the regions array list | The region is now part of the regions array list | Yes |
| Add site to sites array list | The site will now be included in the sites array list | The site is now part of the sites array list | Yes |
| Add to priority list | The object will now be included in the priority array list | The object is now part of the priority array list | Yes |
| Creating instance of region object | The region object is successfully called | The region object was successfully called | Yes |
| Creating instance of site object | The site object is successfully called | The site object was successfully called | Yes |
| Changing the date to activation date | The program updates the sites | The program updates the sites | Yes |
| Changing the date to not being activation date | The program does not update the sites | The program did not update the sites | Yes |

# Evaluation

This report has looked at a system for Saxon heritage which maintain buildings and historical sites in the different regions of England. The system can check a data and based off that information decide whether to update the entire sites catalogue with new visitor numbers and ratings as well as put the lowest performing ones into a priority list.

This system has a few key elements of software engineering that make it a good system. First of all, it has the factory design pattern which increase encapsulation and abstraction improving the validity and overall flexibility of the system. It also has jUnit testing using the test-driven development method increasing the reliability of the system. These things lead me to believe, at least from a software engineering standpoint that this system is a good system that uses strong practices.

A possible improvement for this system would be more encapsulation and abstraction. The way sites are made uses no design pattern and the way they are stored is quite vulnerable to possible malicious attack. Also, the way the priority list is created, it is simply an array list and there is no design pattern meaning it can be quite difficult to understand and there are certain problems that could be faced that have not been faced in the past simply because it is not using a tested architecture and structure.

Overall however I believe the system is strong and uses good software engineering practices and techniques.

# Code Listing

## Main.java

public class Main **{**

public static void main**(**String**[]** args**)** **{**

MainSystem main **=** **new** MainSystem**();**

// The current Date

String date **=** "30/11"**;**

// Array list to hold the priority Sites for marketing

ArrayList**<**Site**>** prioritySites **=** **new** ArrayList**<**Site**>();**

// Create Regions

FactoryRegion factory **=** **new** FactoryRegion**();**

Region london **=** factory**.**getRegion**(**"LONDON"**);**

Region midlands **=** factory**.**getRegion**(**"Midlands"**);**

Region southEast **=** factory**.**getRegion**(**"SouthEast"**);**

Region southWest **=** factory**.**getRegion**(**"SouthWest"**);**

Region northEast **=** factory**.**getRegion**(**"NorthEast"**);**

Region northWest **=** factory**.**getRegion**(**"NorthWest"**);**

// Add regions to array list

ArrayList**<**Region**>** regions **=** **new** ArrayList**<**Region**>();**

regions**.**add**(**london**);**

regions**.**add**(**midlands**);**

regions**.**add**(**northEast**);**

regions**.**add**(**northWest**);**

regions**.**add**(**southEast**);**

regions**.**add**(**southWest**);**

// Create Sites

// London Sites

Site stoneHenge **=** **new** Site**(**"Stone Henge"**,** 45000**);**

Site windsorCastle **=** **new** Site**(**"Windsor Castle"**,** 10000**);**

Site towerOfLondon **=** **new** Site**(**"Tower of London"**,** 30000**);**

london**.**add\_site**(**stoneHenge**);**

london**.**add\_site**(**windsorCastle**);**

london**.**add\_site**(**towerOfLondon**);**

// Midlands Sites

Site coventryCathedral **=** **new** Site**(**"Coventry Cathedral"**,** 3000**);**

Site calkeAbbey **=** **new** Site**(**"Calke Abbey"**,** 5000**);**

Site wollatonHall **=** **new** Site**(**"Wollaton Hall"**,** 6000**);**

midlands**.**add\_site**(**coventryCathedral**);**

midlands**.**add\_site**(**calkeAbbey**);**

midlands**.**add\_site**(**wollatonHall**);**

// Northeast Sites

Site durhamCathedral **=** **new** Site**(**"Durham Cathedral"**,** 6000**);**

Site beamishMuseum **=** **new** Site**(**"Beamish Museum"**,** 1000**);**

Site hexhamAbbey **=** **new** Site**(**"Hexham Abbey"**,** 8000**);**

northEast**.**add\_site**(**durhamCathedral**);**

northEast**.**add\_site**(**beamishMuseum**);**

northEast**.**add\_site**(**hexhamAbbey**);**

// NorthWest Sites

Site liverpoolCathedral **=** **new** Site**(**"Liverpool Cathedral"**,** 31000**);**

Site chesterCathedral **=** **new** Site**(**"Chester Cathedral"**,** 4000**);**

Site walkerArtGallery **=** **new** Site**(**"Walker Art Gallery"**,** 16000**);**

northWest**.**add\_site**(**liverpoolCathedral**);**

northWest**.**add\_site**(**chesterCathedral**);**

northWest**.**add\_site**(**walkerArtGallery**);**

// SouthEast Sites

Site heverCastle **=** **new** Site**(**"Hever Castle"**,** 19000**);**

Site arundelCastle **=** **new** Site**(**"Arundel Castle"**,** 9000**);**

Site canterburyCastle **=** **new** Site**(**"Canterbury Castle"**,** 34000**);**

southEast**.**add\_site**(**heverCastle**);**

southEast**.**add\_site**(**arundelCastle**);**

southEast**.**add\_site**(**canterburyCastle**);**

// SouthWest Sites

Site romanBaths **=** **new** Site**(**"Roman Baths"**,** 25000**);**

Site salisburyCathedral **=** **new** Site**(**"Salisbury Cathedral"**,** 22000**);**

Site wellsCathedral **=** **new** Site**(**"Wells Cathedral"**,** 11000**);**

southWest**.**add\_site**(**romanBaths**);**

southWest**.**add\_site**(**salisburyCathedral**);**

southWest**.**add\_site**(**wellsCathedral**);**

// Print all current sites before update

main**.**currentSiteInfo**(**regions**);**

Boolean dateCheck **=** main**.**checkDate**(**date**);**

**if(**dateCheck **==** **true){**

main**.**updateRecords**(**regions**,** prioritySites**);**

**}**

**}**

**}**

## MainSystem.java

public class MainSystem **{**

//Main method for the system

public void currentSiteInfo**(**ArrayList regions**){**

// For loop to get each region

**for** **(**int f **=** 0**;** f **<** regions**.**size**();** f**++)** **{**

Region current\_region **=** **(**Region**)** regions**.**get**(**f**);**

// For loop to get each site

**for** **(**int i **=** 0**;** i **<** current\_region**.**get\_size**();** i**++)** **{**

Site current\_site **=** current\_region**.**get\_site**(**i**);**

System**.**out**.**println**(**current\_site**);**

**}**

**}**

**}**

public boolean checkDate**(**String date**){**

**if(**date **==** "30/12"**){**

**return** **true;**

**}else** **{**

**return** **false;**

**}**

**}**

public void updateRecords**(**ArrayList regions**,** ArrayList prioritySites**){**

// If function for to check if the date is december 30th

System**.**out**.**println**(**" "**);**

System**.**out**.**println**(**"Date is December 30th // Starting System"**);**

System**.**out**.**println**(**""**);**

// For loop to get current region

**for** **(**int f **=** 0**;** f **<** regions**.**size**();** f**++)** **{**

Region current\_region **=** **(**Region**)** regions**.**get**(**f**);**

// for loop that gets each site and prints it

// If the site is rated bronze it is added to the priority

// arrayList

**for** **(**int i **=** 0**;** i **<** current\_region**.**get\_size**();** i**++)** **{**

Site current\_site **=** current\_region**.**get\_site**(**i**);**

current\_site**.**getLatestVisitorCount**();**

System**.**out**.**println**(**"Updated Record: " **+** current\_site**);**

String siteRating **=** current\_site**.**getSite\_rating**();**

**if** **(**siteRating **==** "Bronze"**)** **{**

prioritySites**.**add**(**current\_site**);**

**}**

**}**

**}**

// Lists the Priority Sites

System**.**out**.**println**(**""**);**

System**.**out**.**println**(**"Priority Sites"**);**

**for** **(**int i **=** 0**;** i **<** prioritySites**.**size**();** i**++)** **{**

System**.**out**.**println**(**prioritySites**.**get**(**i**));**

**}**

**}**

**}**

## Region.java

public interface Region **{**

public void add\_site**(**Site site**);**

public Site get\_site**(**int i**);**

public int get\_size**();**

**}**

## FactoryRegion.java

public class FactoryRegion **{**

public Region getRegion**(**String regionType**)** **{**

**if** **(**regionType **==** **null)** **{**

**return** **null;**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"LONDON"**))** **{**

**return** **new** London**();**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"SOUTHEAST"**))** **{**

**return** **new** SouthEast**();**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"SOUTHWEST"**))** **{**

**return** **new** SouthWest**();**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"NORTHWEST"**))** **{**

**return** **new** NorthWest**();**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"NORTHEAST"**))** **{**

**return** **new** NorthEast**();**

**}**

**if** **(**regionType**.**equalsIgnoreCase**(**"MIDLANDS"**))** **{**

**return** **new** Midlands**();**

**}**

**return** **null;**

**}**

**}**

## Site.java

public class Site **{**

// saves name of site

// has a random number generator for visiors in previous year.

// Paramaters

private String Site\_name**;**

private int Site\_visitors**;**

private String rating**;**

// Constructor

public Site**(**String n**,** int v**)** **{**

**this.**setSite\_name**(**n**);**

**this.**setSite\_visitors**(**v**);**

**this.**setSite\_rating**(**generateRating**(**Site\_visitors**));**

**}**

public Site**()** **{**

Site\_name **=** "????"**;**

Site\_visitors **=** 0**;**

rating **=** "????"**;**

**}**

// Setters and Getters

public String getSite\_name**()** **{**

**return** Site\_name**;**

**}**

public void setSite\_name**(**String site\_name**)** **{**

Site\_name **=** site\_name**;**

**}**

public int getSite\_visitors**()** **{**

**return** Site\_visitors**;**

**}**

public void setSite\_visitors**(**int site\_visitors**)** **{**

Site\_visitors **=** site\_visitors**;**

**}**

public String getSite\_rating**()** **{**

**return** rating**;**

**}**

public void setSite\_rating**(**String site\_rating**)** **{**

rating **=** site\_rating**;**

**}**

public void getLatestVisitorCount**()** **{**

Random rn **=** **new** Random**();**

int visitors **=** rn**.**nextInt**(**50000**)** **+** 1000**;**

Site\_visitors **=** visitors**;**

rating **=** generateRating**(**Site\_visitors**);**

**}**

public String generateRating**(**int Site\_visitors**)** **{**

**if** **(**Site\_visitors **<=** 10000**)** **{**

rating **=** "Bronze"**;**

**}** **else** **if** **((**Site\_visitors **>** 10000**)** **&&** **(**Site\_visitors **<=** 30000**))** **{**

rating **=** "Silver"**;**

**}** **else** **if** **(**Site\_visitors **>** 30000**)** **{**

rating **=** "Gold"**;**

**}**

**return** rating**;**

**}**

public String toString**()** **{**

**return** "Name: " **+** Site\_name **+** " -- Visitors: " **+** Site\_visitors **+** " -- Rating: " **+** rating**;**

**}**

**}**

## NorthEast.java

public class NorthEast **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public NorthEast**(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## NorthWest.java

public class NorthWest **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public NorthWest**(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## SouthEast.java

public class SouthEast **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public SouthEast**(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## SouthWest.java

public class SouthWest **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public SouthWest **(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## Midlands.java

public class Midlands **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public Midlands**(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## London.java

public class London **implements** Region **{**

private String name**;**

private ArrayList**<**Site**>** sites**;**

public London **(){**

sites **=** **new** ArrayList**<**Site**>();**

**}**

public void add\_site**(**Site site**){**

sites**.**add**(**site**);**

**}**

public Site get\_site**(**int i**){**

Site siteRef **=** sites**.**get**(**i**);**

**return** siteRef**;**

**}**

public int get\_size**(){**

**return** sites**.**size**();**

**}**

public String toString**()** **{**

**return** name **+** sites**;**

**}**

**}**

## Tests.java

public class Tests **extends** TestCase **{**

Site testSite**;**

Site testSite2**;**

Region london**;**

MainSystem main **=** **new** MainSystem**();**

ArrayList**<**Site**>** prioritySites **=** **new** ArrayList**<**Site**>();**

ArrayList**<**Region**>** regions **=** **new** ArrayList**<**Region**>();**

public void setUp**()** **throws** Exception **{**

// Create the test site

testSite **=** **new** Site**(**"TEST"**,** 1234**);**

testSite2 **=** **new** Site**(**"TEST 2"**,** 5678**);**

// Create the region

FactoryRegion factory **=** **new** FactoryRegion**();**

london **=** factory**.**getRegion**(**"LONDON"**);**

london**.**add\_site**(**testSite**);**

regions**.**add**(**london**);**

**}**

// getting the site name

@Test

public void testGetSiteName**()** **{**

String name **=** testSite**.**getSite\_name**();**

assertEquals**(**"TEST"**,** name**);**

**}**

// Getting the site visitors

@Test

public void testGetSiteVisitors**()** **{**

int visitors **=** testSite**.**getSite\_visitors**();**

assertEquals**(**1234**,** visitors**);**

**}**

// getting the site rating

@Test

public void testGetSiteRating**()** **{**

String rating **=** testSite**.**getSite\_rating**();**

assertEquals**(**"Bronze"**,** rating**);**

**}**

// Getting the latest site visitor count

@Test

public void testGetLatestVisitorCount**()** **{**

int originalCount **=** testSite**.**getSite\_visitors**();**

testSite**.**getLatestVisitorCount**();**

int newCount **=** testSite**.**getSite\_visitors**();**

assertTrue**(**originalCount **!=** newCount**);**

**}**

// Test the rating generation method.

@Test

public void testGenerateRating**()** **{**

// Test Bronze rating

int visitors **=** 5000**;**

String rating **=** testSite**.**generateRating**(**visitors**);**

assertEquals**(**"Bronze"**,** rating**);**

// Test Silver rating

visitors **=** 15000**;**

rating **=** testSite**.**generateRating**(**visitors**);**

assertEquals**(**"Silver"**,** rating**);**

// Test Gold rating

visitors **=** 35000**;**

rating **=** testSite**.**generateRating**(**visitors**);**

assertEquals**(**"Gold"**,** rating**);**

**}**

// REGION TESTS

//Testing the region size

@Test

public void testRegionSize**()** **{**

int size **=** london**.**get\_size**();**

assertEquals**(**1**,** size**);**

**}**

@Test

public void testSiteAddition**()** **{**

// LONDON.ADD\_SITE(TESTSITE);

Site testing **=** london**.**get\_site**(**0**);**

assertEquals**(**testSite**,** testing**);**

**}**

@Test

public void testRegionSize2**()** **{**

london**.**add\_site**(**testSite2**);**

int size **=** london**.**get\_size**();**

assertEquals**(**2**,** size**);**

**}**

//MAIN SYSTEM TESTS

@Test

public void testUpdateRecords**(){**

int visitors **=** testSite**.**getSite\_visitors**();**

main**.**updateRecords**(**regions**,** prioritySites**);**

int visitors2 **=** testSite**.**getSite\_visitors**();**

assertTrue**(**visitors **!=** visitors2**);**

**}**

public void testPrioritySites**(){**

int size **=** prioritySites**.**size**();**

main**.**updateRecords**(**regions**,** prioritySites**);**

String rating **=** testSite**.**getSite\_rating**();**

int size2 **=** prioritySites**.**size**();**

**if(**rating **==** "Bronze"**){**

assertTrue**(**size **!=** size2**);**

**}else{**

assertEquals**(**size**,** size2**);**

**}**

**}**

public void testCheckDate**(){**

String date **=** "30/12"**;**

Boolean result **=** main**.**checkDate**(**date**);**

assertTrue**(**result **==** **true);**

date **=** "30/11"**;**

result **=** main**.**checkDate**(**date**);**

assertTrue**(**result **==** **false);**

**}**

**}**