

REPORT

Applied Software Engineering



Joe Sykes
21223675

Contents

Introduction	2
System Requirements	3
Unified Modelling Language	4
Use Case	4
Class Diagram	5
Sequence Diagram	6
Architecture Diagram	8
Implementation	10
Use of design patterns	10
How the system works	10
How to run the software	11
Evidence of running software	11
With the wrong activation date:	11
With the right activation date:	12
Testing	13
JUnit Testing	13
System Testing	14
Evaluation	15
Code Listing	16
Main.java	16
MainSystem.java	18
Region.java	19
FactoryRegion.java	19
Site.java	20
NorthEast.java	21
NorthWest.java	22
SouthEast.java	22
SouthWest.java	23
Midlands.java	23
London.java	24
Tests.java	24

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.

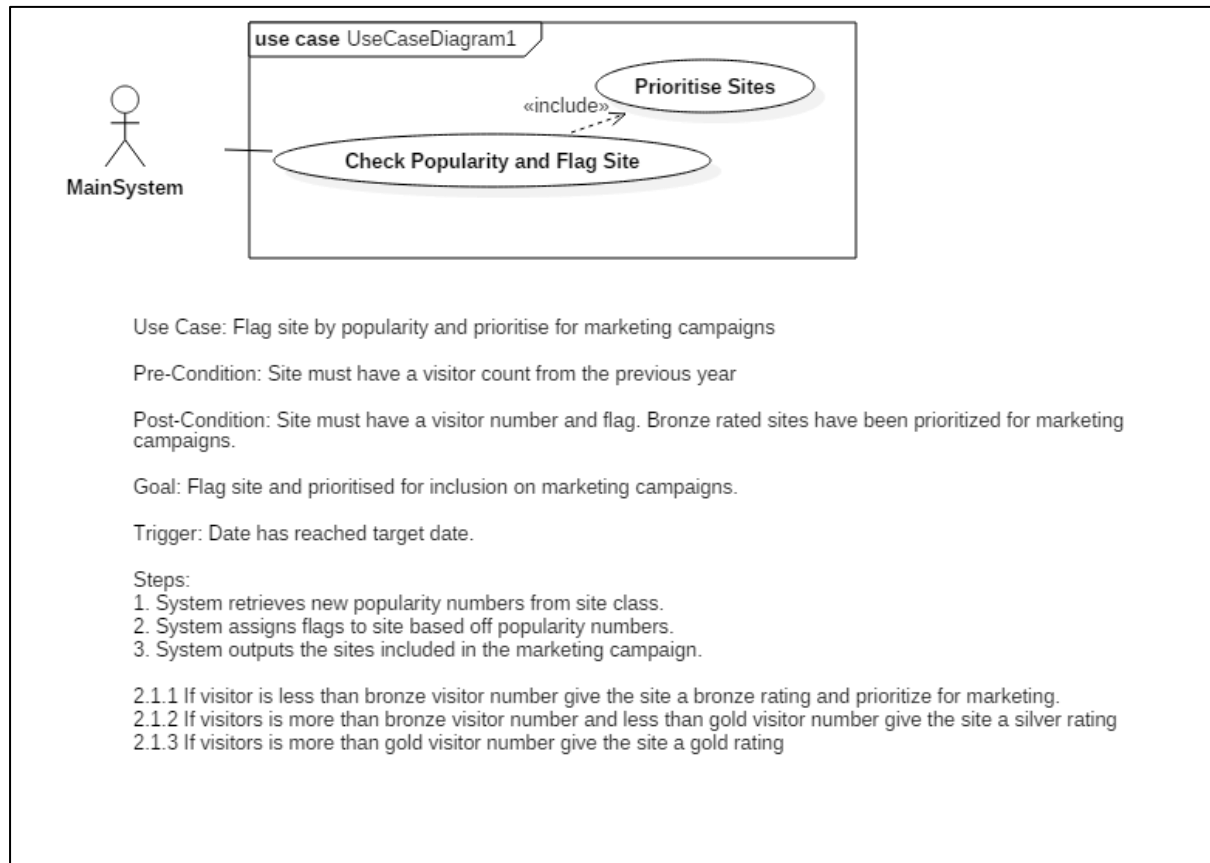
System Requirements

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

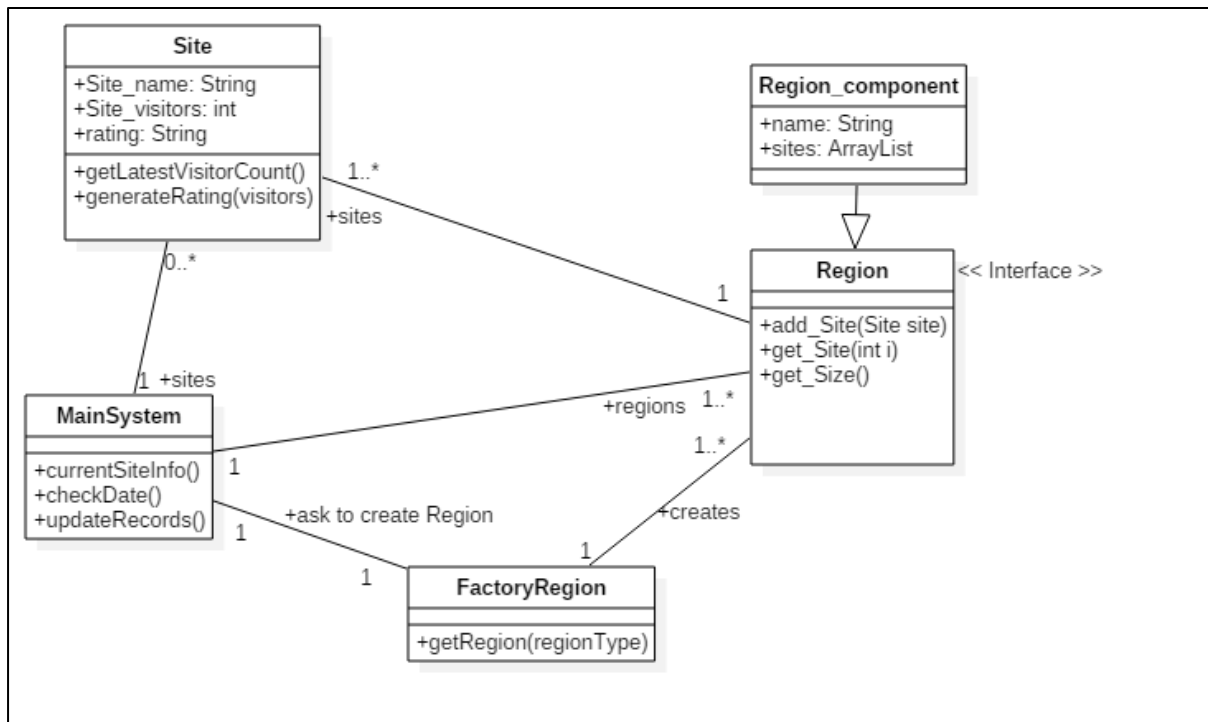
Use Case



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



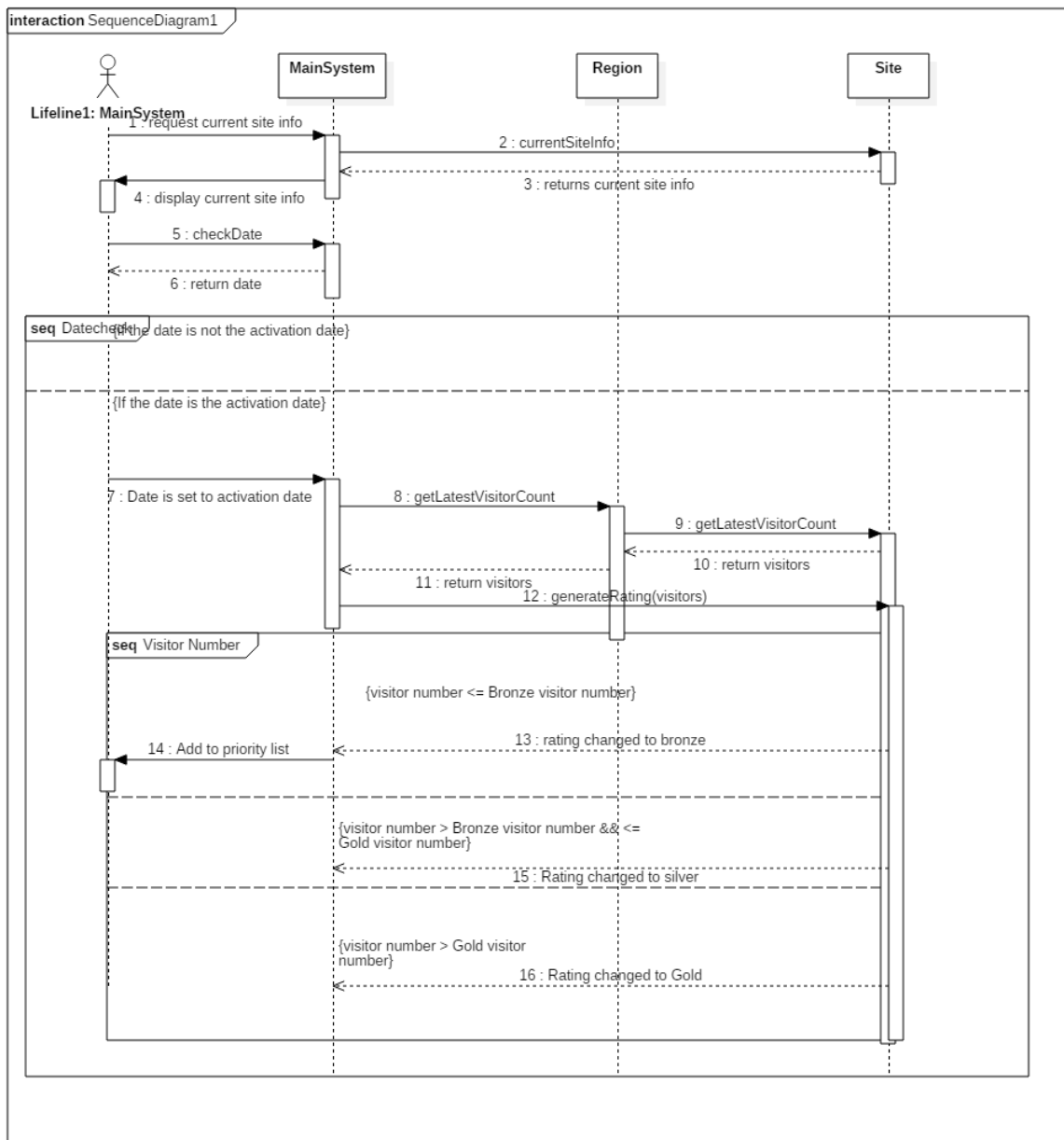
The class diagram for this system contains five separate classes. The main system class is where the main methods of the system are held, this includes getting the current site info, checking the current date and updating the records in the system. This links to the site class because it initiates the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

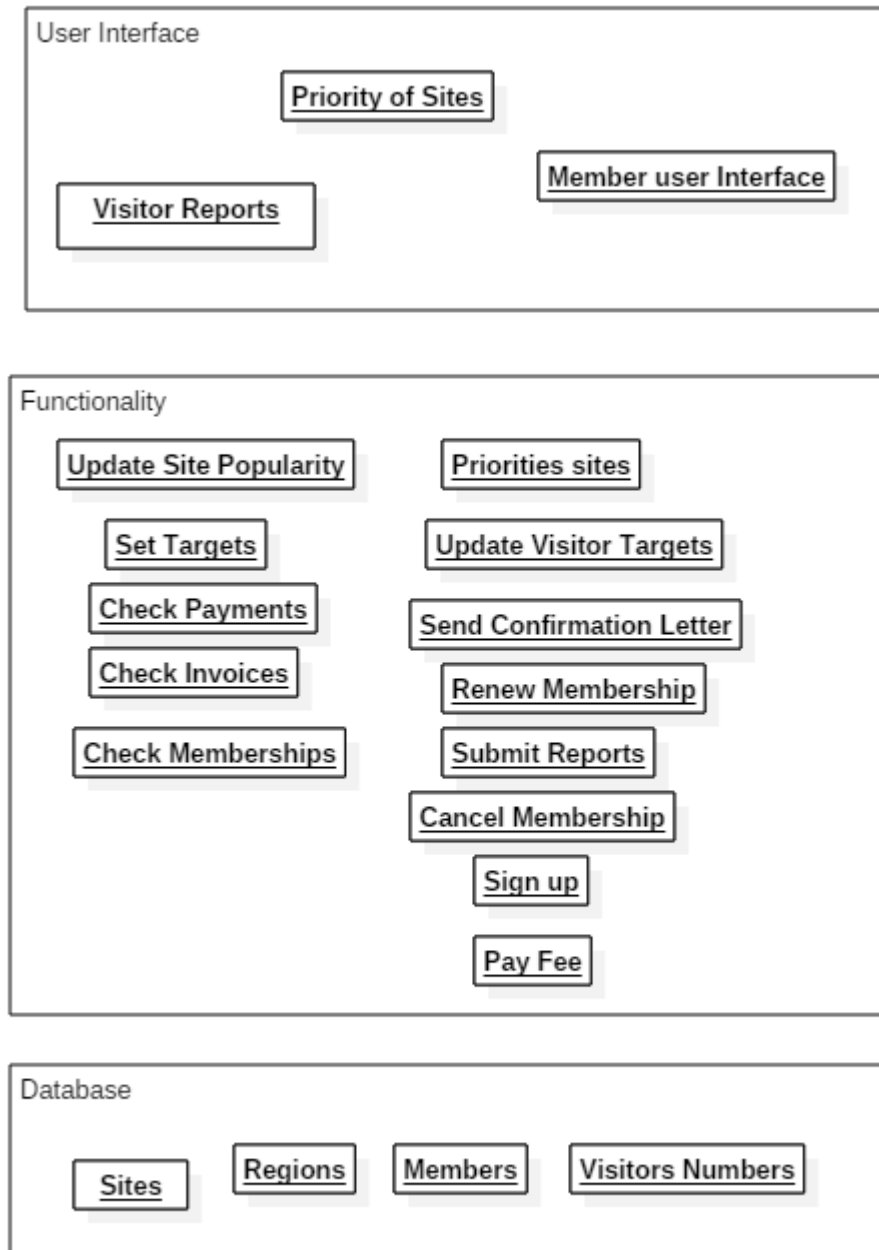
service that has no user interaction but it is easier to demonstrate how the system is triggered using the actor lifeline.

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 it 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

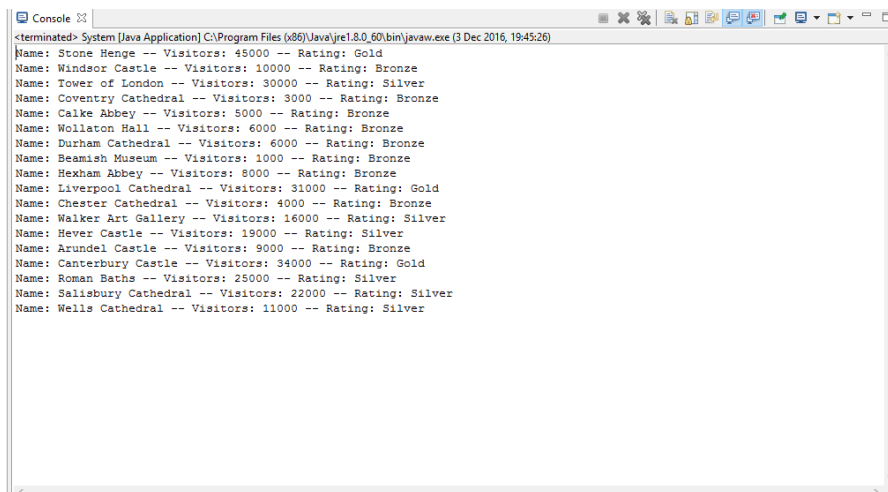
instance of an object of it which it used in the system. This is the factory design pattern.

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:



```

<terminated> System [Java Application] C:\Program Files (x86)\Java\jre1.8.0_60\bin\javaw.exe (3 Dec 2016, 19:45:26)
Name: Stone Henge -- Visitors: 45000 -- Rating: Gold
Name: Windsor Castle -- Visitors: 10000 -- Rating: Bronze
Name: Tower of London -- Visitors: 30000 -- Rating: Silver
Name: Coventry Cathedral -- Visitors: 3000 -- Rating: Bronze
Name: Calke Abbey -- Visitors: 5000 -- Rating: Bronze
Name: Wollaton Hall -- Visitors: 6000 -- Rating: Bronze
Name: Durham Cathedral -- Visitors: 6000 -- Rating: Bronze
Name: Beamish Museum -- Visitors: 1000 -- Rating: Bronze
Name: Hexham Abbey -- Visitors: 8000 -- Rating: Bronze
Name: Liverpool Cathedral -- Visitors: 31000 -- Rating: Gold
Name: Chester Cathedral -- Visitors: 4000 -- Rating: Bronze
Name: Walker Art Gallery -- Visitors: 16000 -- Rating: Silver
Name: Hever Castle -- Visitors: 19000 -- Rating: Silver
Name: Arundel Castle -- Visitors: 9000 -- Rating: Bronze
Name: Canterbury Castle -- Visitors: 34000 -- Rating: Gold
Name: Roman Baths -- Visitors: 25000 -- Rating: Silver
Name: Salisbury Cathedral -- Visitors: 22000 -- Rating: Silver
Name: Wells Cathedral -- Visitors: 11000 -- Rating: Silver
  
```

With the right activation date:

```

<terminated> System [Java Application] C:\Program Files (x86)\Java\jre1.8.0_60\bin\javaw.exe (3 Dec 2016, 19:44:12)
Name: Stone Henge -- Visitors: 45000 -- Rating: Gold
Name: Windsor Castle -- Visitors: 10000 -- Rating: Bronze
Name: Tower of London -- Visitors: 30000 -- Rating: Silver
Name: Coventry Cathedral -- Visitors: 3000 -- Rating: Bronze
Name: Calke Abbey -- Visitors: 5000 -- Rating: Bronze
Name: Wollaton Hall -- Visitors: 6000 -- Rating: Bronze
Name: Durham Cathedral -- Visitors: 6000 -- Rating: Bronze
Name: Beamish Museum -- Visitors: 1000 -- Rating: Bronze
Name: Hexham Abbey -- Visitors: 8000 -- Rating: Bronze
Name: Liverpool Cathedral -- Visitors: 31000 -- Rating: Gold
Name: Chester Cathedral -- Visitors: 4000 -- Rating: Bronze
Name: Walker Art Gallery -- Visitors: 16000 -- Rating: Silver
Name: Hever Castle -- Visitors: 19000 -- Rating: Silver
Name: Arundel Castle -- Visitors: 9000 -- Rating: Bronze
Name: Canterbury Castle -- Visitors: 34000 -- Rating: Gold
Name: Roman Baths -- Visitors: 25000 -- Rating: Silver
Name: Salisbury Cathedral -- Visitors: 22000 -- Rating: Silver
Name: Wells Cathedral -- Visitors: 11000 -- Rating: Silver

Date is December 30th // Starting System

Updated Record: Name: Stone Henge -- Visitors: 12435 -- Rating: Silver
Updated Record: Name: Windsor Castle -- Visitors: 42583 -- Rating: Gold
Updated Record: Name: Tower of London -- Visitors: 49745 -- Rating: Gold
Updated Record: Name: Coventry Cathedral -- Visitors: 11905 -- Rating: Silver
Updated Record: Name: Calke Abbey -- Visitors: 38396 -- Rating: Gold
Updated Record: Name: Wollaton Hall -- Visitors: 33096 -- Rating: Gold
Updated Record: Name: Durham Cathedral -- Visitors: 4430 -- Rating: Bronze
Updated Record: Name: Beamish Museum -- Visitors: 21491 -- Rating: Silver
Updated Record: Name: Hexham Abbey -- Visitors: 32635 -- Rating: Gold
Updated Record: Name: Liverpool Cathedral -- Visitors: 41442 -- Rating: Gold
Updated Record: Name: Chester Cathedral -- Visitors: 14147 -- Rating: Silver
Updated Record: Name: Walker Art Gallery -- Visitors: 40807 -- Rating: Gold

Name: Walker Art Gallery -- Visitors: 16000 -- Rating: Silver
Name: Hever Castle -- Visitors: 19000 -- Rating: Silver
Name: Arundel Castle -- Visitors: 9000 -- Rating: Bronze
Name: Canterbury Castle -- Visitors: 34000 -- Rating: Gold
Name: Roman Baths -- Visitors: 25000 -- Rating: Silver
Name: Salisbury Cathedral -- Visitors: 22000 -- Rating: Silver
Name: Wells Cathedral -- Visitors: 11000 -- Rating: Silver

Date is December 30th // Starting System

Updated Record: Name: Stone Henge -- Visitors: 12435 -- Rating: Silver
Updated Record: Name: Windsor Castle -- Visitors: 42583 -- Rating: Gold
Updated Record: Name: Tower of London -- Visitors: 49745 -- Rating: Gold
Updated Record: Name: Coventry Cathedral -- Visitors: 11905 -- Rating: Silver
Updated Record: Name: Calke Abbey -- Visitors: 38396 -- Rating: Gold
Updated Record: Name: Wollaton Hall -- Visitors: 33096 -- Rating: Gold
Updated Record: Name: Durham Cathedral -- Visitors: 4430 -- Rating: Bronze
Updated Record: Name: Beamish Museum -- Visitors: 21491 -- Rating: Silver
Updated Record: Name: Hexham Abbey -- Visitors: 32635 -- Rating: Gold
Updated Record: Name: Liverpool Cathedral -- Visitors: 41442 -- Rating: Gold
Updated Record: Name: Chester Cathedral -- Visitors: 14147 -- Rating: Silver
Updated Record: Name: Walker Art Gallery -- Visitors: 40807 -- Rating: Gold
Updated Record: Name: Hever Castle -- Visitors: 22530 -- Rating: Silver
Updated Record: Name: Arundel Castle -- Visitors: 26159 -- Rating: Silver
Updated Record: Name: Canterbury Castle -- Visitors: 49879 -- Rating: Gold
Updated Record: Name: Roman Baths -- Visitors: 4369 -- Rating: Bronze
Updated Record: Name: Salisbury Cathedral -- Visitors: 13660 -- Rating: Silver
Updated Record: Name: Wells Cathedral -- Visitors: 19579 -- Rating: Silver

Priority Sites
Name: Durham Cathedral -- Visitors: 4430 -- Rating: Bronze
Name: Roman Baths -- Visitors: 4369 -- Rating: Bronze

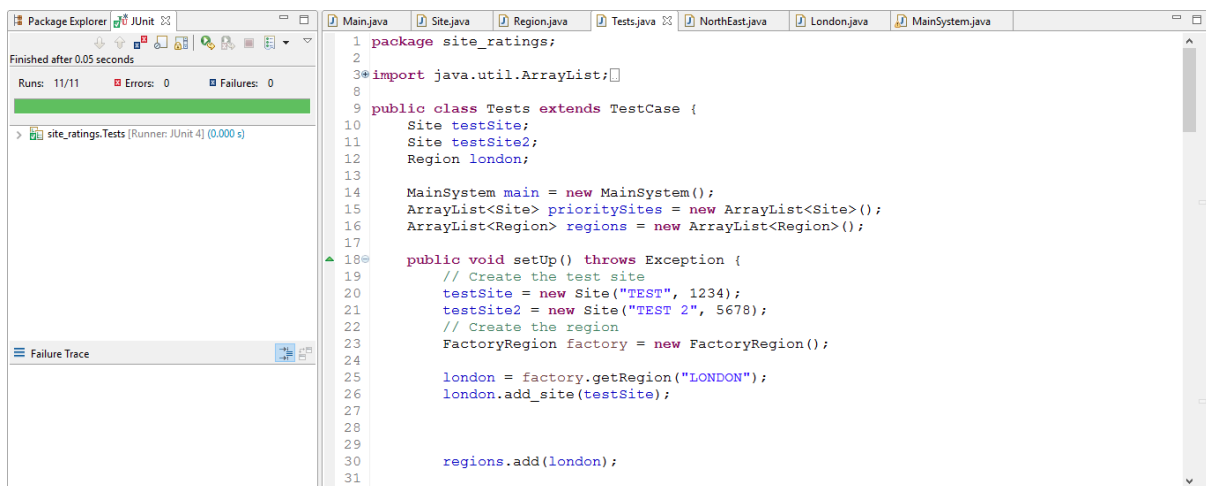
```

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.



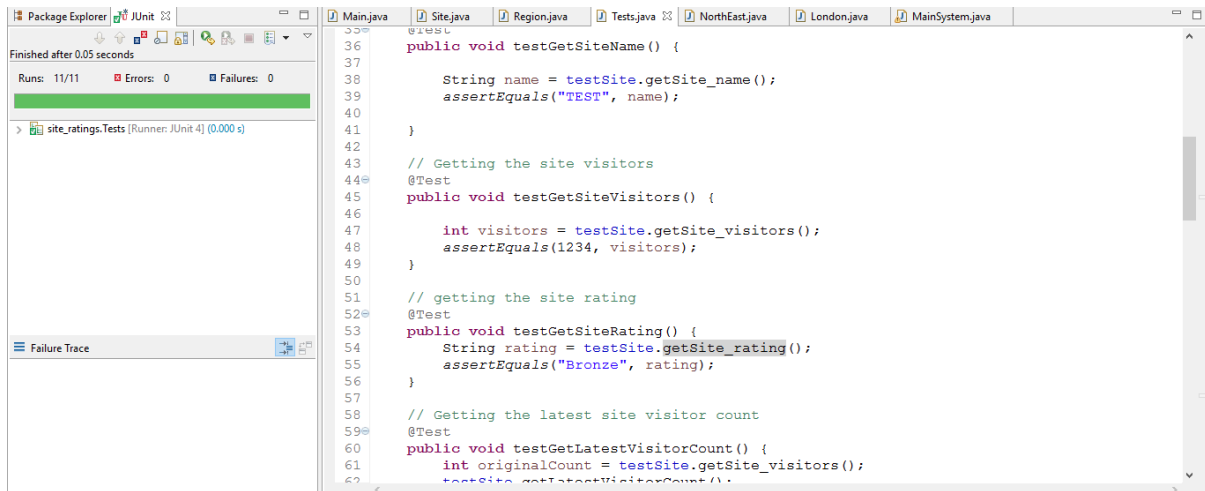
The screenshot shows an IDE with the JUnit test runner on the left and the source code on the right. The test runner shows 'Finished after 0.05 seconds', 'Runs: 11/11', 'Errors: 0', and 'Failures: 0'. The test class is 'site_ratings.Tests [Runner: JUnit 4] (0.000 s)'. The source code is in 'Tests.java' and shows the following:

```

1 package site_ratings;
2
3 import java.util.ArrayList;
4
5
6
7
8
9 public class Tests extends TestCase {
10     Site testSite;
11     Site testSite2;
12     Region london;
13
14     MainSystem main = new MainSystem();
15     ArrayList<Site> prioritySites = new ArrayList<Site>();
16     ArrayList<Region> regions = new ArrayList<Region>();
17
18     public void setUp() throws Exception {
19         // Create the test site
20         testSite = new Site("TEST", 1234);
21         testSite2 = new Site("TEST 2", 5678);
22         // Create the region
23         FactoryRegion factory = new FactoryRegion();
24
25         london = factory.getRegion("LONDON");
26         london.add_site(testSite);
27
28
29
30         regions.add(london);
31

```

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 `assertEquals` 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 visitors 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);  
    }  
}
```