ITM 411 Mini Project 2: Read Me

1. Abstract

Aim: To provide data analytics related to winter solstice, summer solstice, vernal equinox and autumnal equinox using sample data input file.

Objective: The objective of this mini project 2 was to help us understand the fundamental JAVA concepts such as Serialization and De-Serialization, Exception Handling, Reading, Parsing and Writing Files (.csv, .ser, .txt), Implementation of Comparator Interfaces, Date Class Utilization.

Basic Understanding:

- **Serialization and De-Serialization:** Serialization is the process of writing the state of the object to a byte stream. This is useful if we want to save the state of the program to a persistent storage area such as a file. At a later time, we may restore these objects by using the process of de-serialization.

- **Exception Handling:** JAVA exception handling is managed by 5 keywords:

    1. **Try:** To guard against and handle a run-time error, we enclose the code to be monitored inside a try block

    2. **Catch:** Immediately following the try block, we include a catch clause that specifies the exception type of the exception to be caught

    3. **Throw:** This enables to throw an exception explicitly. The flow of execution stops immediately after the throw statement, and the nearest enclosing try and catch block are examined to find the matching type of exception

    4. **Throws:** If a method is capable of causing an exception that it does not handle, it must specify this behavior so that the callers of the method can guard themselves against that exception. This can be done by using throws clause in the method declaration

    5. **Finally:** This block contains the code that will be executed after a try/catch block has completed and before the following try/catch block. This block is executed whether or not an exception is thrown.

All exception types are subclasses of the built-in class Throwable.

2. System Requirement

OS: Windows 7
JDK: 1.7.0
IDE: NetBeans IDE 7.2.1

3. Architecture or System Flow

3.1 Creation of Java Application:
Launch NetBeans and click on new Project to choose a new Java Application, name it as ‘MiniProject2’ and click on Finish

3.2 Creation of packages:
Along with the default package, MiniProject2, create a new package ‘domain’ and ‘domain.util’ at the same level

3.3 Creation of classes/interfaces (Java files):

- Record class- This is the basic abstract class which contains the fields to store the month, date and their respective sunrise and sunset times. The fields month , date, sunrise and sunset are stored as integer value. We can have just 2 fields such as ‘sunrisetime’ and ‘sunsettime’ declared of type Date. While parsing the .csv file we then will have to use DateFormat, SimpleDateFormat and Date classes to send these values to the DaylightRecord class constructor.

- DaylightRecord class- This class extends the Record class. This class also implements the serializable interface. It has few functions to calculate the difference between the sunset/sunrise times for the particular date of the invoking object.

- PersistentObject class- This class encapsulates the Current Timestamp in the form of a date object and the read .csv input in the form of a list object.

- WinterSolstice class- This class implements the Comparator interface in order to return the date for the winter solstice. It implements the compare() method to compare the differences between the sunrise/sunset times of the invoking objects to find out the winter solstice date.

- SummerSolstice class- This class implements the Comparator interface in order to return the date for the summer solstice. It implements the compare() method to compare the differences between the sunrise/sunset times of the invoking objects to find out the summer solstice date.

- Equinox class- This class implements the Comparator interface in order to return the date for the autumnal equinox. It implements the compare() method to compare the differences between the sunrise/sunset times of the invoking objects to find out the autumnal equinox date.

- VernalEquinox class- This class implements the Comparator interface in order to return the date for the vernal equinox. It implements the compare() method to compare the differences between the sunrise/sunset times of the invoking objects to find out the vernal equinox date.

All the above classes are included in the package ‘domain’

- Utilities class- This class contains many methods which are called from the main class. This is defined in order to minimize the main method of the MiniProject2 class.

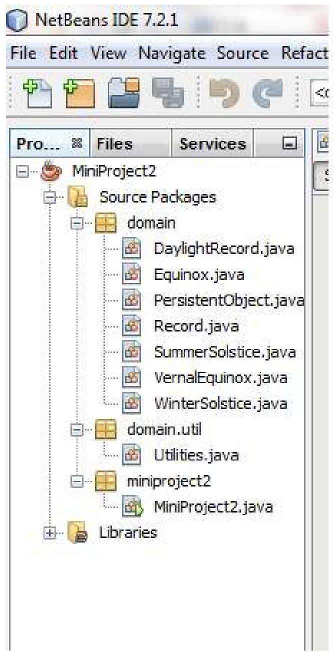
The above classes are included in the package 'domain.util'

- MiniProject2 class- This class represents the Driver class containing the main method. Import the relevant classes Record, DaylightRecord, PersistentObject, WinterSolstice, SummerSolstice Equinox, VernalEquinox and Utilities from package domain and domain.util

3.4 Script file:
Created a 'script.bat' file located in the MiniProject2 folder. This file is responsible to convert the MiniProject2.jar file (This file is created after every successful build of the java code) into mp2out.txt file (located at the root level of the main project).

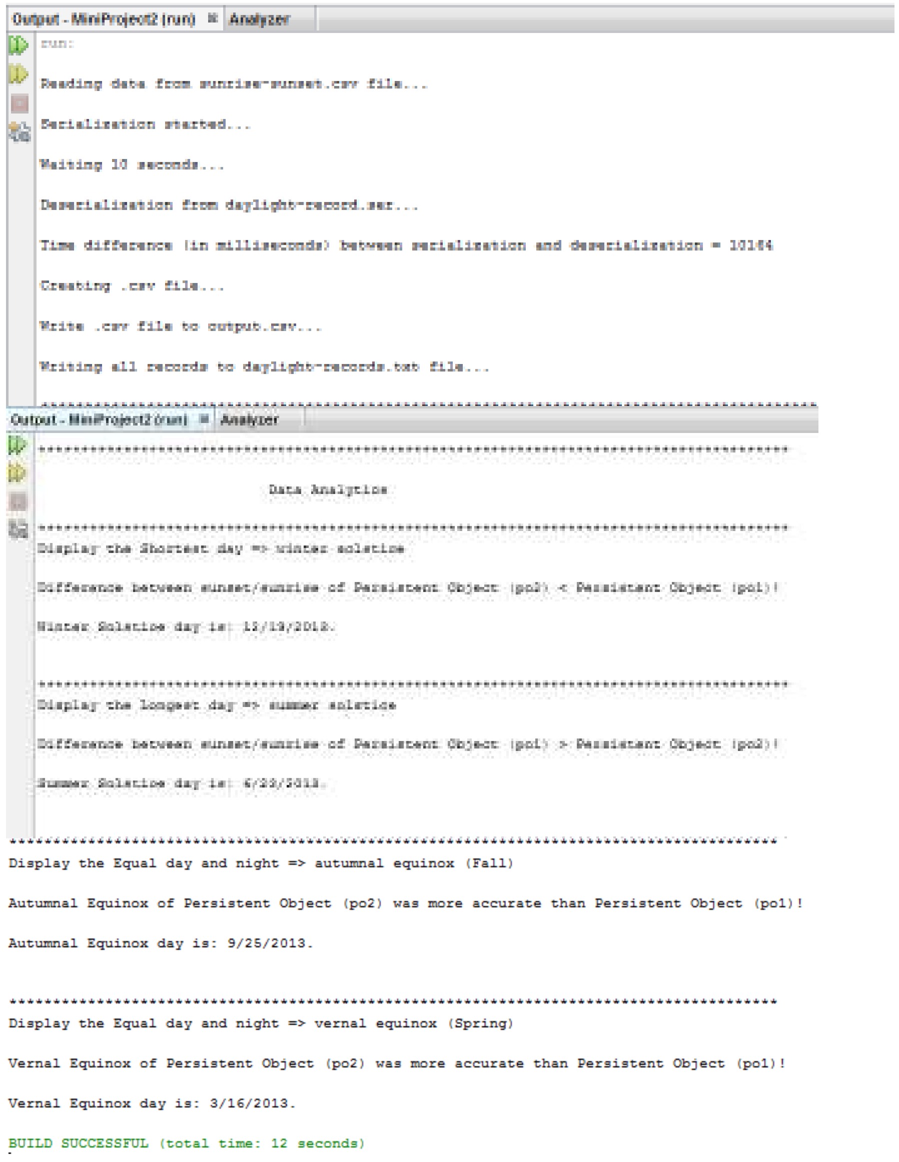
3.5 Javadoc API:
This document depicts in detail in HTML format, all the comments written in the code. It can be created by right clicking on the java project and choosing the 'Generate Javadocs' option. This file folder gets automatically created inside the main project folder's 'dist' folder. It can be moved to a suitable location if needed.

4. Screen Shots



4.1 Structure of the Java Project: The above screen capture shows the basic folder structure for the MiniProject2 JAVA application

4.2 Output files:



Output - MiniProject2 (run) Analyzer

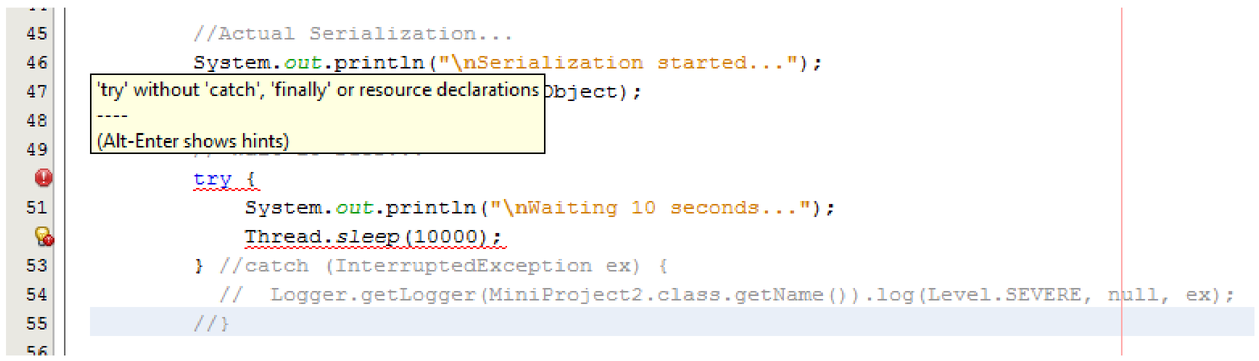
```
run:

Reading data from sunrise-sunset.csv file...

Serialization started...

Waiting 10 seconds...

Deserialization from daylight-record.ser...

Time difference (in milliseconds) between serialization and deserialization = 10164

Creating .csv file...

Write .csv file to output.csv...

Writing all records to daylight-records.txt file...

*************************************************************************************
```

Output - MiniProject2 (run) Analyzer

```
*************************************************************************************

                          Data Analytics

*************************************************************************************
Display the Shortest day => winter solstice

Difference between sunset/sunrise of Persistent Object (po2) < Persistent Object (po1)!

Winter Solstice day is: 12/19/2012.


*************************************************************************************
Display the Longest day => summer solstice

Difference between sunset/sunrise of Persistent Object (po1) > Persistent Object (po2)!

Summer Solstice day is: 6/22/2013.


*************************************************************************************
Display the Equal day and night => autumnal equinox (Fall)

Autumnal Equinox of Persistent Object (po2) was more accurate than Persistent Object (po1)!

Autumnal Equinox day is: 9/25/2013.


*************************************************************************************
Display the Equal day and night => vernal equinox (Spring)

Vernal Equinox of Persistent Object (po2) was more accurate than Persistent Object (po1)!

Vernal Equinox day is: 3/16/2013.

BUILD SUCCESSFUL (total time: 12 seconds)
```

4.3 Test Cases:

- Test Case 1 – Negative



```
        //Actual Serialization...
        System.out.println("\nSerialization started...");
                                               Object);


        try {
            System.out.println("\nWaiting 10 seconds...");
            Thread.sleep(10000);
        } //catch (InterruptedException ex) {
          //  Logger.getLogger(MiniProject2.class.getName()).log(Level.SEVERE, null, ex);
        //}
```

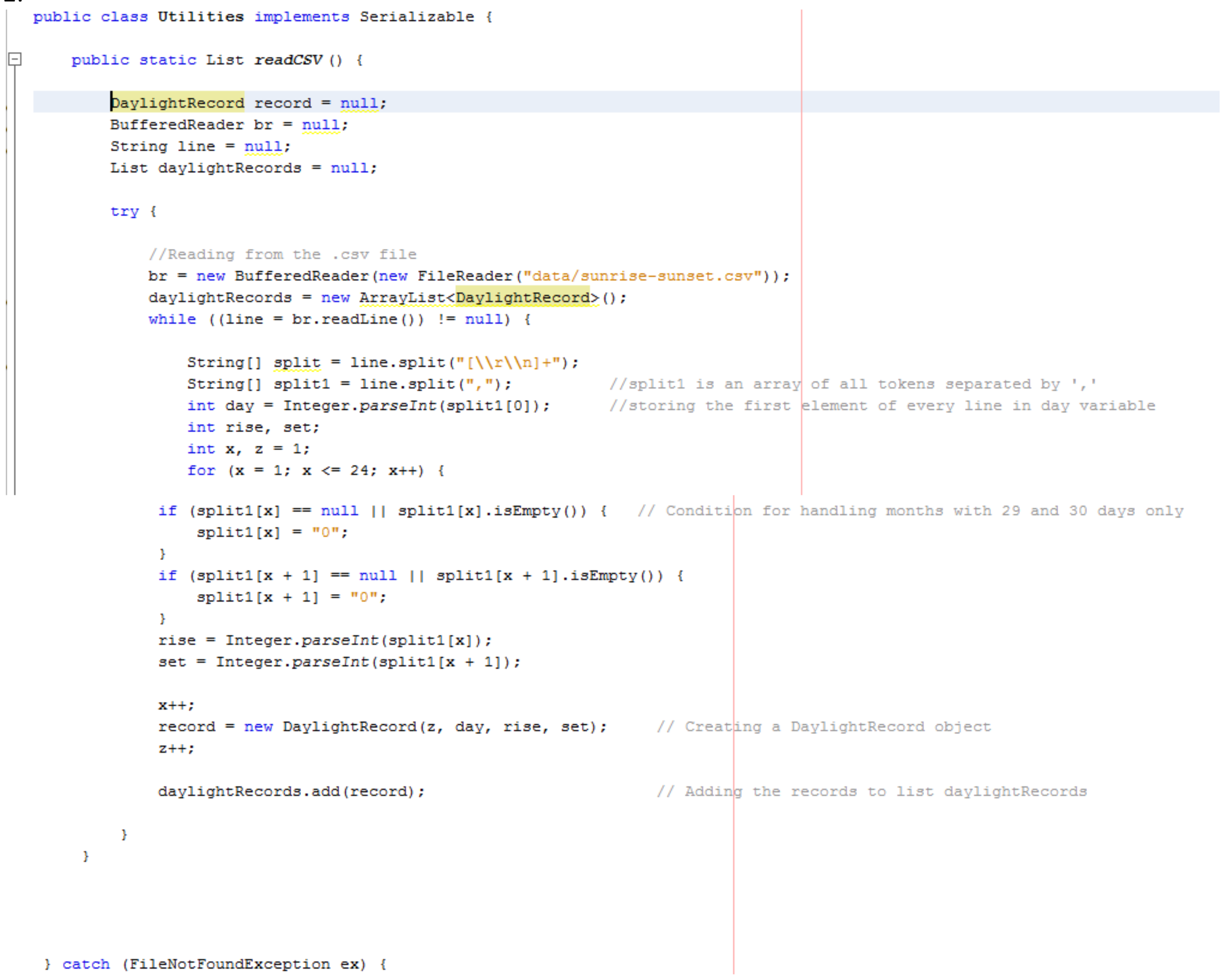
'try' without 'catch', 'finally' or resource declarations

(Alt-Enter shows hints)

It shows that we cannot have a try clause without a following catch clause

- Application Capabilities

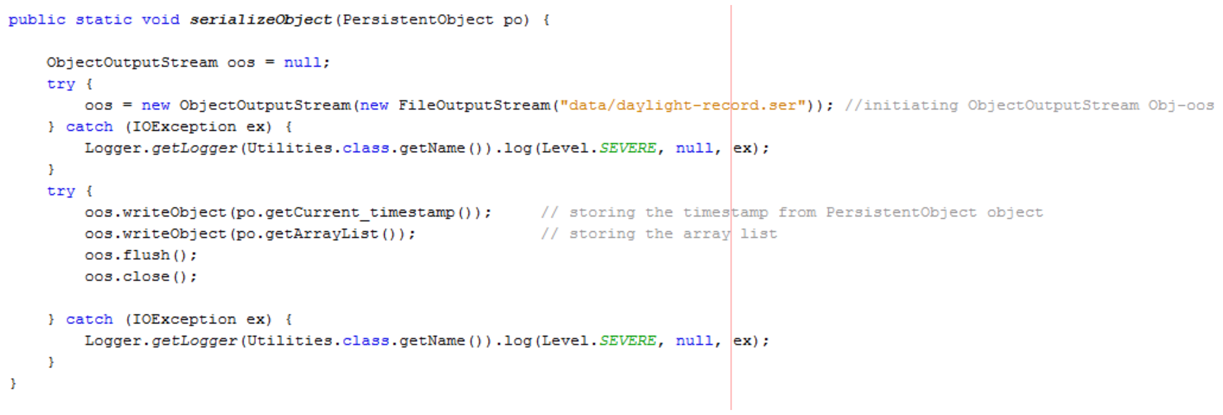
1.



```java
public class Utilities implements Serializable {

    public static List readCSV () {

        DaylightRecord record = null;
        BufferedReader br = null;
        String line = null;
        List daylightRecords = null;

        try {

            //Reading from the .csv file
            br = new BufferedReader(new FileReader("data/sunrise-sunset.csv"));
            daylightRecords = new ArrayList<DaylightRecord>();
            while ((line = br.readLine()) != null) {

                String[] split = line.split("[\\r\\n]+");
                String[] split1 = line.split(",");          //split1 is an array of all tokens separated by ','
                int day = Integer.parseInt(split1[0]);      //storing the first element of every line in day variable
                int rise, set;
                int x, z = 1;
                for (x = 1; x <= 24; x++) {

             if (split1[x] == null || split1[x].isEmpty()) {   // Condition for handling months with 29 and 30 days only
                 split1[x] = "0";
             }
             if (split1[x + 1] == null || split1[x + 1].isEmpty()) {
                 split1[x + 1] = "0";
             }
             rise = Integer.parseInt(split1[x]);
             set = Integer.parseInt(split1[x + 1]);

             x++;
             record = new DaylightRecord(z, day, rise, set);     // Creating a DaylightRecord object
             z++;

             daylightRecords.add(record);                        // Adding the records to list daylightRecords

         }
     }



 } catch (FileNotFoundException ex) {
```

This method parses the given .csv data file and adds the record into daylightRecords list. It returns the daylightRecords list.

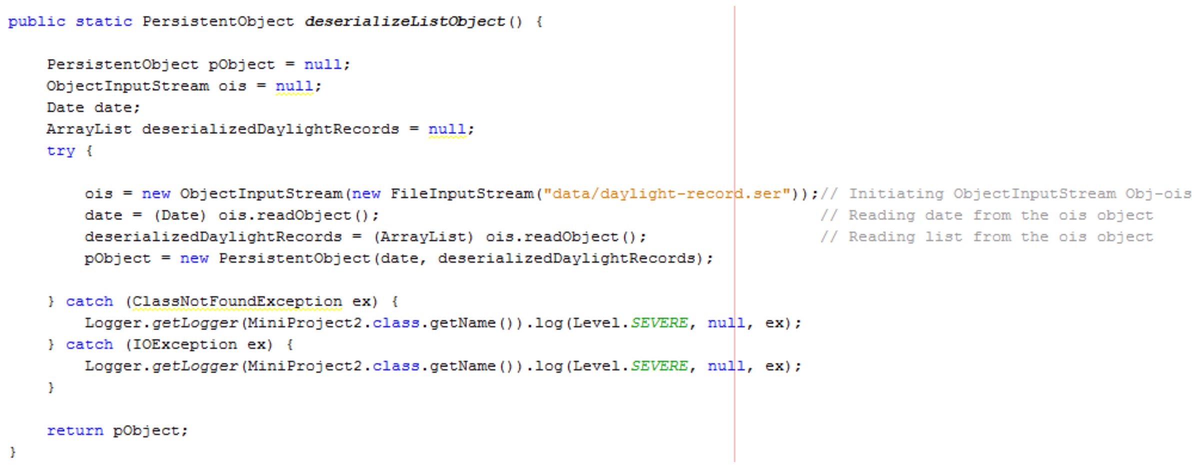
2.



```java
public static void serializeObject(PersistentObject po) {

    ObjectOutputStream oos = null;
    try {
        oos = new ObjectOutputStream(new FileOutputStream("data/daylight-record.ser")); //initiating ObjectOutputStream Obj-oos
    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        oos.writeObject(po.getCurrent_timestamp());     // storing the timestamp from PersistentObject object
        oos.writeObject(po.getArrayList());             // storing the array list
        oos.flush();
        oos.close();

    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

This method serializes the list to create a daylight-record.ser file. It makes use of the writeObject() function of ObjectOutputStream class.

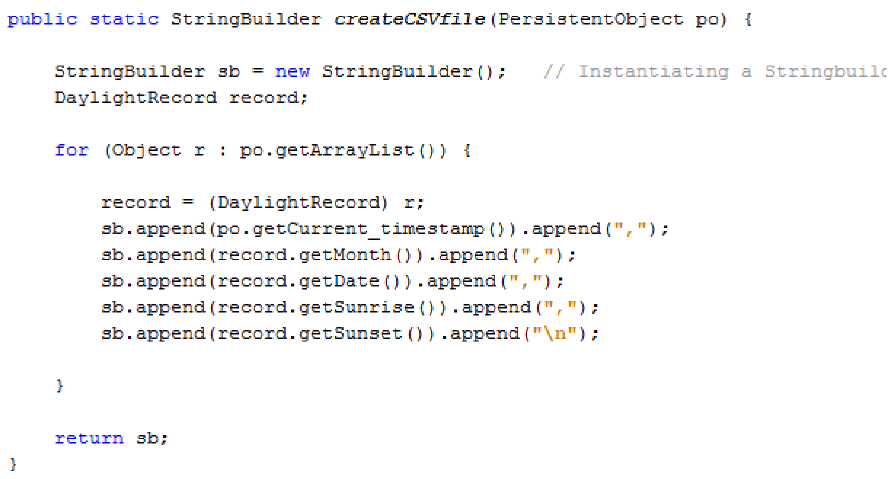
3.



```java
public static PersistentObject deserializeListObject() {

    PersistentObject pObject = null;
    ObjectInputStream ois = null;
    Date date;
    ArrayList deserializedDaylightRecords = null;
    try {

        ois = new ObjectInputStream(new FileInputStream("data/daylight-record.ser"));// Initiating ObjectInputStream Obj-ois
        date = (Date) ois.readObject();                                             // Reading date from the ois object
        deserializedDaylightRecords = (ArrayList) ois.readObject();                 // Reading list from the ois object
        pObject = new PersistentObject(date, deserializedDaylightRecords);

    } catch (ClassNotFoundException ex) {
        Logger.getLogger(MiniProject2.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(MiniProject2.class.getName()).log(Level.SEVERE, null, ex);
    }

    return pObject;
}
```

This method helps in de-serialization of the daylight-record.ser file .It makes use of the readObject() method of ObjectInputStream class. The contents are read into the fields of PersistentObject's object.

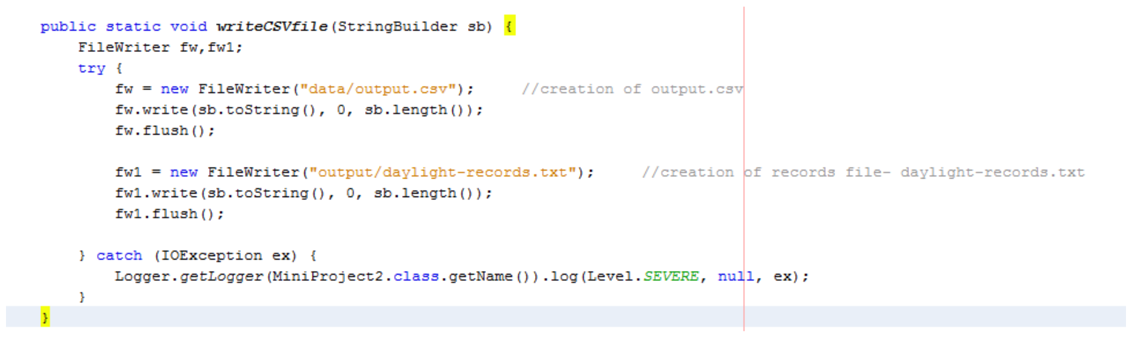
4.



```java
public static StringBuilder createCSVfile(PersistentObject po) {

    StringBuilder sb = new StringBuilder();   // Instantiating a Stringbuilc
    DaylightRecord record;

    for (Object r : po.getArrayList()) {

        record = (DaylightRecord) r;
        sb.append(po.getCurrent_timestamp()).append(",");
        sb.append(record.getMonth()).append(",");
        sb.append(record.getDate()).append(",");
        sb.append(record.getSunrise()).append(",");
        sb.append(record.getSunset()).append("\n");

    }

    return sb;
}
```

This method returns a StringBuilder object which binds the various fields of DaylightRecord class.

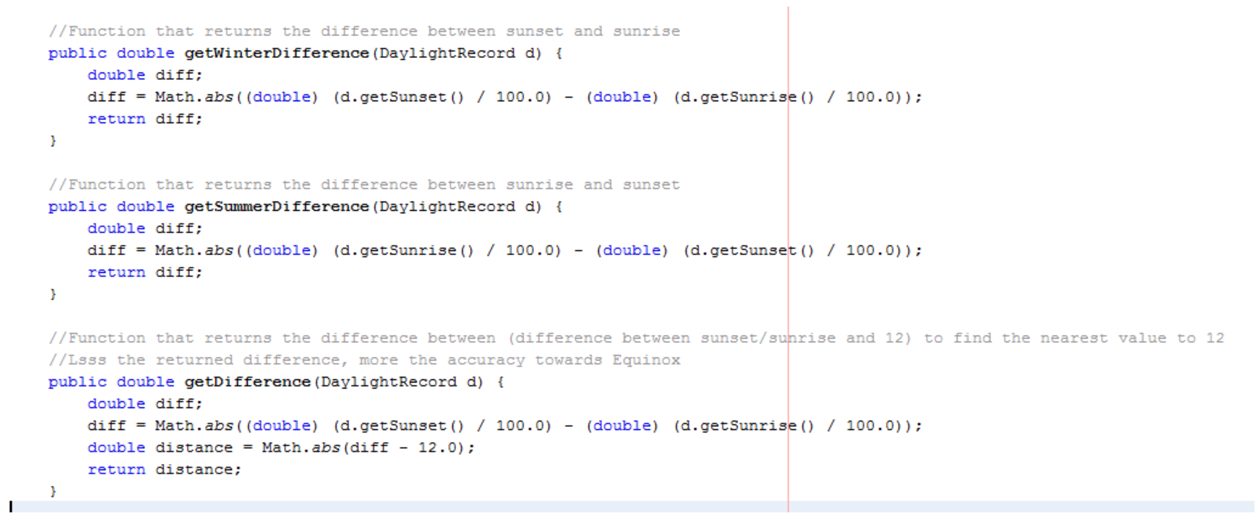
5.



```java
public static void writeCSVfile(StringBuilder sb) {
    FileWriter fw,fw1;
    try {
        fw = new FileWriter("data/output.csv");      //creation of output.csv
        fw.write(sb.toString(), 0, sb.length());
        fw.flush();

        fw1 = new FileWriter("output/daylight-records.txt");     //creation of records file- daylight-records.txt
        fw1.write(sb.toString(), 0, sb.length());
        fw1.flush();

    } catch (IOException ex) {
        Logger.getLogger(MiniProject2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

This method creates two output files that contains the output records one is stored in the .csv format and another one is stored in the .txt file.

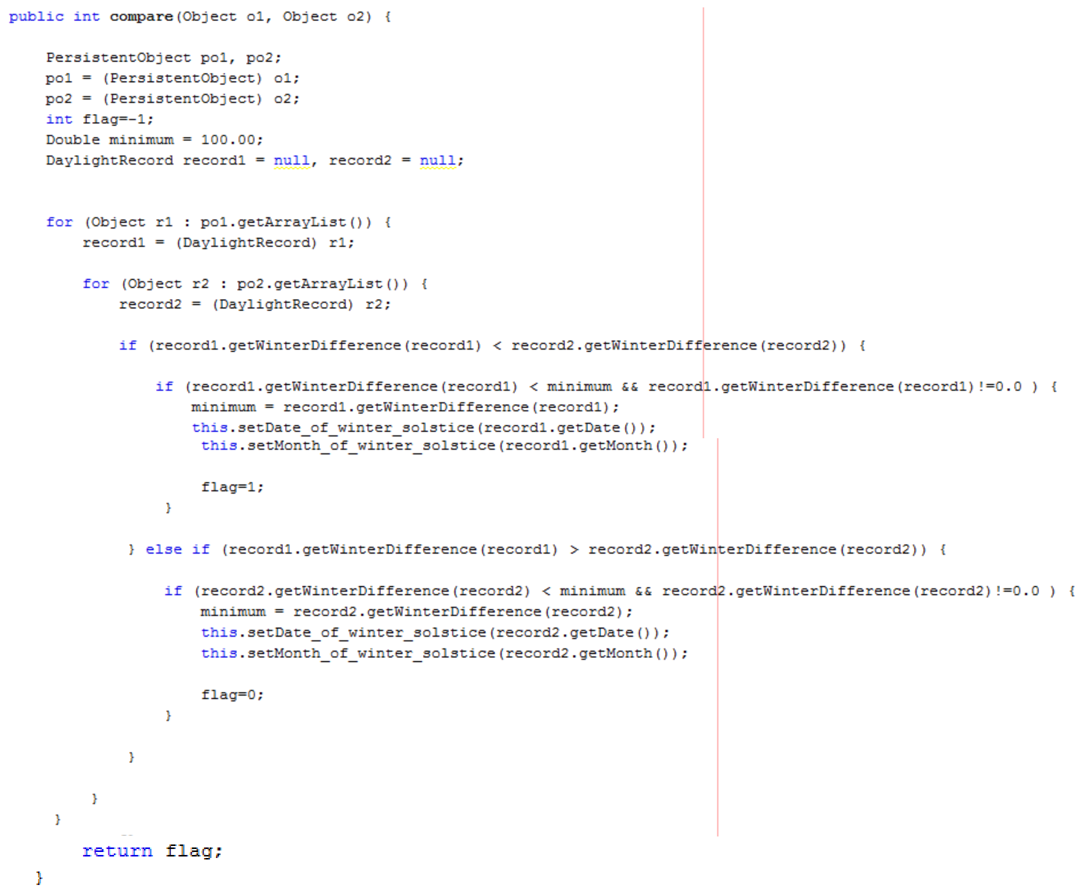
6.



```java
//Function that returns the difference between sunset and sunrise
public double getWinterDifference(DaylightRecord d) {
    double diff;
    diff = Math.abs((double) (d.getSunset() / 100.0) - (double) (d.getSunrise() / 100.0));
    return diff;
}

//Function that returns the difference between sunrise and sunset
public double getSummerDifference(DaylightRecord d) {
    double diff;
    diff = Math.abs((double) (d.getSunrise() / 100.0) - (double) (d.getSunset() / 100.0));
    return diff;
}

//Function that returns the difference between (difference between sunset/sunrise and 12) to find the nearest value to 12
//Lsss the returned difference, more the accuracy towards Equinox
public double getDifference(DaylightRecord d) {
    double diff;
    diff = Math.abs((double) (d.getSunset() / 100.0) - (double) (d.getSunrise() / 100.0));
    double distance = Math.abs(diff - 12.0);
    return distance;
}
```

These three methods are called from the Compare() method by the respective DaylightRecord objects in order to compare the differences between the sunrise and sunset times to calculate the longest day, shortest day and equinoxes.

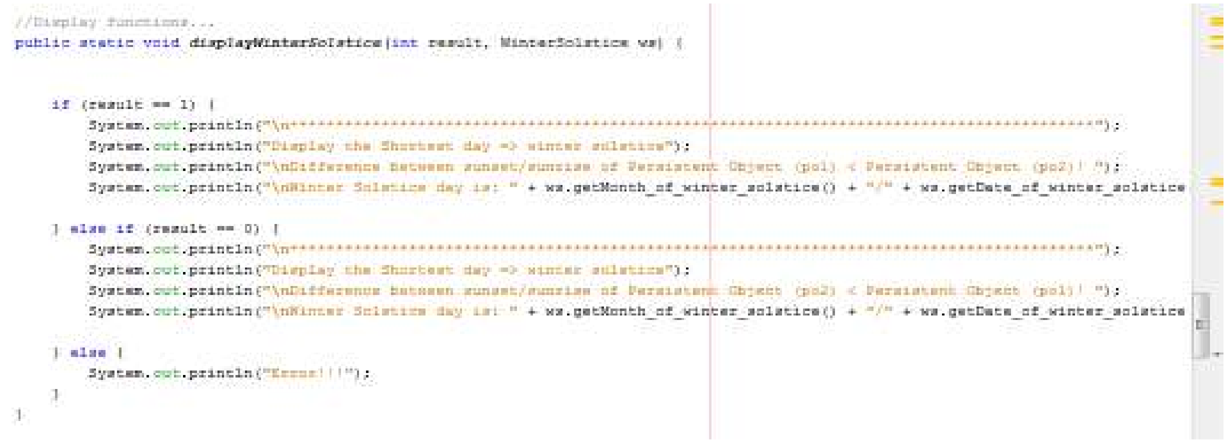
7.



```
public int compare(Object o1, Object o2) {

    PersistentObject po1, po2;
    po1 = (PersistentObject) o1;
    po2 = (PersistentObject) o2;
    int flag=-1;
    Double minimum = 100.00;
    DaylightRecord record1 = null, record2 = null;


    for (Object r1 : po1.getArrayList()) {
        record1 = (DaylightRecord) r1;

        for (Object r2 : po2.getArrayList()) {
            record2 = (DaylightRecord) r2;

            if (record1.getWinterDifference(record1) < record2.getWinterDifference(record2)) {

                if (record1.getWinterDifference(record1) < minimum && record1.getWinterDifference(record1)!=0.0 ) {
                    minimum = record1.getWinterDifference(record1);
                    this.setDate_of_winter_solstice(record1.getDate());
                     this.setMonth_of_winter_solstice(record1.getMonth());

                     flag=1;
                }

             } else if (record1.getWinterDifference(record1) > record2.getWinterDifference(record2)) {

                 if (record2.getWinterDifference(record2) < minimum && record2.getWinterDifference(record2)!=0.0 ) {
                     minimum = record2.getWinterDifference(record2);
                     this.setDate_of_winter_solstice(record2.getDate());
                     this.setMonth_of_winter_solstice(record2.getMonth());

                     flag=0;
                }

             }

         }
     }

        return flag;
    }
```

This compare() method is used across classes winterSolstice, summerSolstice, Equinox and vernalEquinox in order to get the exact date when the days with longer, shorter and equal days would occur. It gives call to the respective difference method from the DaylightRecord class.

8.



```
//Display functions...
public static void displayWinterSolstice(int result, WinterSolstice ws) {


    if (result == 1) {
        System.out.println("\n*****************************************************************************************");
        System.out.println("Display the Shortest day => winter solstice");
        System.out.println("\nDifference between sunset/sunrise of Persistent Object (po1) < Persistent Object (po2)! ");
        System.out.println("\nWinter Solstice day is: " + ws.getMonth_of_winter_solstice() + "/" + ws.getDate_of_winter_solstice

    } else if (result == 0) {
        System.out.println("\n*****************************************************************************************");
        System.out.println("Display the Shortest day => winter solstice");
        System.out.println("\nDifference between sunset/sunrise of Persistent Object (po2) < Persistent Object (po1)! ");
        System.out.println("\nWinter Solstice day is: " + ws.getMonth_of_winter_solstice() + "/" + ws.getDate_of_winter_solstice

    } else {
        System.out.println("Error!!!");
    }
}
```

This method is called in order to display the data analytics information. It is defined in the similar way for all the remaining classes.

5. Conclusion
Thus, the code was implemented successfully. Please go through the mp2out.txt in order to view the output of the code.

6.Submission Package

It includes following folders:

1. MiniProject2:
   This includes the main JAVA application

- It has a script.bat file which is responsible to create the mp2out.txt file at the project top level.
- It has a data folder which contains the input, serialized and deserialized data files.
- It has a output folder that contains the records file- daylight-records.txt

2. MiniProject2Docs:
   This includes the JavaAPI docs and the README file