

Intermediate Software Development – MiniProject 2

Project Description:

The goal of this project is to compute several data analytics by providing sample data file as input feed for the application. An abstract super class called *Record* and subclass called *DaylightRecord* employs instance variables for each of the field in the data file. Default constructor (no-argument) and full-argument overloaded constructors are provided. We should use *java.util.Comparator* or *java.lang.Comparable* for comparing various fields in application's data analytic requirements. *Record*, *DaylightRecord* and *Comparator* files should be in the common package called *domain*. Then encapsulate a *java.util.Date* object and the *ArrayList* object into a serializable class called *PersistentObject*. Ultimately, a driver class should be added through which data files are read and stored in *daylightRecord* arraylist, create instances of persistent object, make application sleep for 10 seconds, serialize the persistentobject to a file named *daylight-record.ser* , deserialize the persistent object to date object and Arraylist object called *deserializedDaylightRecords*, compute the time difference between serialization and deserialization, compute data analytics for shortest day in winter solstice, longest day in summer solstice, equal day and night for vernal and autumnal equinox .

Installation, Compile and Run Time requirements:

1. NetBeans IDE 7.2.1
2. Java 1.7.0_11, Java Hotspot(TM) 64-Bit Server VM 23.6-b04
3. Windows 7 version 6.1 running on amd64

Insights, Expected results, and Challenges:

This project helped me get hands on experience on the concepts of Exceptions, basic I/O streams(File I/O, Serialization, Deserialization), Collections (List), Comparator.

While reading the data from given “.csv” file we used *BufferedReader* and performing serialization (marker interface that tells the JVM to write out the state of the object to some stream – which is usually used to read all the members, and write out their state to a stream. The default mechanism is a binary format) and deserialization (turns a serialized stream of binary format or bytes back into a copy of the original object), we used *ObjectInputStream*, *ObjectOutput Stream* and *File writer* thus implementing Basic I/O operations.

While performing basic I/O operations described above there could be error thrown during the operation and should be handled appropriately by the use of Exceptions.The main exceptions that are caused while performing above operations are *IOException*, *FileNotFoundException*.

Collections are mainly used to group multiple elements into single object. In this project, I have used List to store the entire DaylightRecord object into an ArrayList. I have used the arraylist to compute and store data analytics result.

Comparator are usually used to compare two objects and also used to order the data of some collection objects. In this project, I have used Collection.sort to sort the elements of the dataAnalyticsResult which is resulted by computing various data analytics requirements. WinterSolstice, SummerSolstics and VernalandAutumnal equinox are the comparator classes implemented in this project.

The packages used in this project are listed below:

- 1) domain
- 2) domain.util
- 3) driver

1) **domain** package consists of the following classes:

- i. Record
- ii. DaylightRecord
- iii. PersistentObject
- iv. DataAnalyticsResult
- v. WinterSolstice
- vi. SummerSolstics
- vii. VernalandAutumnalEquinox

i. Record Superclass:

This class is an abstract super class. This is the parent class and it performs the following Operations:

- Import the required packages.
- Declaration of variables.
- Provide No-arguments constructor with default values.
- Provide Full-arguments constructor.
- Provide accessors and mutators for the variables declared.
- Provide toString() method.
- Implement Serializable.

ii. DaylightRecord Subclass:

This class is a subclass which does the following operations:

- Import the required package
- Inherit the parent class ie., the abstract super class 'Record'

iii. DataAnalyticsResult :

This class is used to store data after various data analytics computation and does the following operations:

- Import the required packages.
- Declaration of variables.

iv. WinterSolstice :

This class is used for winter solstice computation as comparator and does the following function:

- Implements comparator.
- Compares the values of datafields in the given objects data.

v. SummerSolstice:

This class is used for the summer solstice computation as comparator and does the following function:

- Implements comparator.
- Compares the values of datafields in the given objects data.

vi. VernalandAutumnalEquinox:

This class is used for the spring and fall equinox computation as comparator and does the following function:

- Implements comparator.
- Compares the values of datafields in the given objects data.

2) **domain.util** package consists of the following class:

i. Utilities :

This class is used to implement methods which will be called from driver class. This class helps in minimizing main() methods and understanding of the main class in a better and easy way. This class has methods to read “.csv” file, serialization, deserialization, create “.csv” file, write “.csv” file back to the system, compute data analytics to retrieve shortest day in winter , longest day summer solstice and equal day and night in equinox. The functionalities of all the methods are as described below:

readCSVfile() :

In this method, I create the instance of DaylightRecord and add the data into an arraylist named daylightRecords.

serializeListObject(PersistentObject pobj):

In this method, persistent object in which date and arraylist daylightRecords encapsulated are serialized to daylight-record.ser.

deserializeListObject():

In this method, the serialized data is deserialized back into app.

createCSVfile():

In this method, deserialized data is read, formatted(ie to insert ‘,’,’\n’) as required and stored accordingly into a stringbuilder..

writeCSVfile(StringBuilder sb):

In this method, the stringbuilder data which is the result of createCSVfile() method, is taken and a file deserializedDaylightRecords.csv is created.

findCorrespondingData(List daylightRecords, String startDate, String endDate):

This method, is used to compute data analytics for winter and summer solstice months. It takes daylightRecords, startDate and endDate as parameters. startDate will be the winter Solstice's or Summer Solstice's starting date and endDate will be the winter solstice or

summer solstice ending dates respectively. The function returns the list of all computed result of all the days in winter and summer.

findRelevantData(List daylightRecords, String startDate, String endDate):

This method, is used to compute data analytics for spring and fall equinox months. It take daylightRecords and startDate and endDate as parameters. startDate will be the spring Equinox or Fall Equinox starting date and endDate will be the Spring Equinox's or Fall Equinox's ending dates respectively. The function returns the list of all computed result for all the days in spring and fall

computeSolsticeDataAnalytics(Object o, String startDayofSolstice, String endDayofSolstice):

This method is mainly used to find the difference between sunrise and sunset for the days in winter and summer solstice. This method is called from findCorrespondingData(List day.....,String ..., String ..), and takes object, startDate and endDate as parameter.

computeEquinoxDataAnalytics(Object o1, Object o2, String equinoxStartDate, String equinoxEndDate):

This method is mainly used to find the difference between sunrise and sunset of a day, and subtract it from 24hrs, for all days coming in fall and spring equinox. This method is called from findRelevantData(List day.....,String ..., String ..), and takes object, startDate and endDate as parameter.

retrieveDataFromComputation(List newList, List dataList):

This method is used to retrieve the final results of all the data analytics computed ie. the shortest day of winter, longest day of summer and equal day and night for spring and fall equinox.

3) **driver** package consist of the following class:

I. MP2

This is the main class from where the project execution starts and performs the following operations:

- Reads data into the DaylightRecordObjects.
- Collects the *DaylightRecord* objects into an *ArrayList* called *daylightRecords*.
- Creates an instance of *PersistentObject* with the current timestamp and the *ArrayList* object.
- Serializes the persistent object to a file called *daylight-record.ser*
- Invokes tread to make application sleep for 10 seconds.
- Deserialize the persisted object into a date object and an *ArrayList* object called *deserializedDaylightRecords*.
- Computes the time difference between serialization and deserialization.
- Invokes methods to compute data analytics in which the results are stored into DataAnalytics objects and then added into an arraylist. Then arraylist is used with collections.sort and comparators to get the desired output from which we can display the result of Shortest day in winter solstice, Equal day and night in vernal equinox, Longest day in summer solstice.

Test Cases:

<i>Expected Results</i>	<i>Test Result</i>
1. Should read input data into DaylightRecord Objects and Collect <i>DaylightRecord</i> objects into an <i>ArrayList</i> called <i>daylightRecords</i> .	Pass
2. Should create an instance of <i>PersistentObject</i> with the current timestamp and the <i>ArrayList</i> object.	Pass
3. Should serialize the persistent object to a file called <i>daylight-record.ser</i>	Pass
4. Make the application sleep for 10 seconds.	Pass
5. Should deserialize the persisted object into a date object and an <i>ArrayList</i> object called <i>deserializedDaylightRecords</i> .	Pass
6. Should compute and display the time difference between serialization and deserialization	Pass
7. Should compute and display the following data analytics: Shortest day in winter solstice Equal day and night in vernal equinox (Spring) Longest day in summer solstice Equal day and night in autumnal equinox (Fall)	Pass
8. Should write data stored in daylightRecords to a text file called <i>daylight-records.txt</i>	Pass
9. Should write all the results of the project to a text file mp2out.txt	Pass

Note: For data analytics computation, this program reads the entire year's data to calculate the shortest and longest day in solstice and for equal day and night, it considers date range from March-June for vernal and September-December for autumnal equinox. Alternatively, you can also specify any date ranges for all the above computations (For ex. You can modify date range from December to March for winter etc)

Challenges Faced:

The major challenge I faced while computing data analytic is for equinox. I was confused to opt in any of the below two methods:

1. I thought of computing it by taking sunrise and sunset of the day and check whether it is 12 so that if there are 24 hours in a day then automatically night length will be 12.
2. The other method I thought is to take sunrise and sunset of the current day subtract it which gives the length of the day, and then take sunset of current day and sunrise of the other day and subtract it which given the length of the night. Compare both of them which should be equal.

Then I decided to use option 1. since the requirement says to find equal day and night of a day. If we use the second option then we might end up considering the next day's value until sunrise.

Screenshots:

1. Record superclass:

Snapshot 1.1:

```
- */
public abstract class Record implements Serializable {

    private String day;
    private Date sunRise;
    private Date sunSet;

    //No argument Constructor
    public Record() {
        day = "noday";
        sunRise = null;
        sunSet = null;
    }

    //full argument Constructor
    public Record(String day, Date sunRise, Date sunSet) {
        this.day = day;
        this.sunRise = sunRise;
        this.sunSet = sunSet;
    }

    //Accessors and Mutators
    public String getDay() {
        return day;
    }
}
```

Record superclass which is made serializable and have fields for storing data read from data files. no-arg and full-arg constructors are implemented for fields

Snapshot 1.2

```

public void setDay(String day) {
    this.day = day;
}

public Date getSunRise() {
    return sunRise;
}

public void setSunRise(Date sunRise) {
    this.sunRise = sunRise;
}

public Date getSunSet() {
    return sunSet;
}

public void setSunSet(Date sunSet) {
    this.sunSet = sunSet;
}

@Override
public String toString() {
    return "Record{" + "day=" + day + ", sunRise=" + df.format(sunRise) + ", sunSet=" + df.format(sunSet) + '}';
}

```

Record superclass getters, setters, and toString method()

2. DaylightRecord subclass

Snapshot 2.1:

```


5 package domain;
6
7 import java.io.Serializable;
8 import java.util.Date;
9
10 /**
11  * @author Meghashree M Ramachandra
12  */
13 public class DaylightRecord extends Record implements Serializable {
14
15     //No argument Constructor
16     public DaylightRecord() {
17         super();
18     }
19
20     //full argument constructor
21     public DaylightRecord(String day, Date sunRise, Date sunSet) {
22         super(day, sunRise, sunSet);
23     }
24
25     @Override
26     public String toString() {
27         return "DayLightRecord{" + super.toString() + '}';
28     }
29 }

```

DaylightRecord subclass with no-arg and full-arg constructors, accessors, mutators and toString() method

3. PersistentObject class

Snapshot 3.1:

<pre> public class PersistentObject implements Serializable { private Date dateObj; private List<DaylightRecord> dayLightRecord; //No argument constructor public PersistentObject() { dateObj = null; dayLightRecord = null; } //Full argument constructor public PersistentObject(Date dateObj, List<DaylightRecord> dayLightRecord) { this.dateObj = dateObj; this.dayLightRecord = dayLightRecord; } public Date getDateObj() { return dateObj; } public void setDateObj(Date dateObj) { this.dateObj = dateObj; } public List<DaylightRecord> getDayLightRecord() { return dayLightRecord; } </pre>		<p>PersistentObject class with Date and arraylist objects encapsulated, which is made serializable. no-arg, full-arg constructors, getter and setter method implemented</p>
---	---	---

Snapshot 3.2:

<pre> public void setDateObj(Date dateObj) { this.dateObj = dateObj; } public List<DaylightRecord> getDayLightRecord() { return dayLightRecord; } public void setDayLightRecord(List<DaylightRecord> dayLightRecord) { this.dayLightRecord = dayLightRecord; } @Override public String toString() { return "PersistentObject{" + "dateObj=" + dateObj + ", dayLightRecord=" + dayLightRecord + '}'; } </pre>		<p>PersistentObject class setters, getters and toString() method.</p>
---	---	---

4. DataAnalyticsResult class:

Snapshot 4.1:

```
public class DataAnalyticsResult {  
  
    private int arrayIndexData;  
    private int computationData;  
  
    //No argument constructor  
    public DataAnalyticsResult() {  
        arrayIndexData = 0;  
        computationData = 0;  
    }  
  
    //full argument constructor  
    public DataAnalyticsResult(int arrayIndexData, int computationData) {  
        this.arrayIndexData = arrayIndexData;  
        this.computationData = computationData;  
    }  
  
    //Accessors and Mutators  
    public int getArrayIndexData() {  
        return arrayIndexData;  
    }  
  
    public void setArrayIndexData(int arrayIndexData) {  
        this.arrayIndexData = arrayIndexData;  
    }  
  
    public int getComputationData() {  
        return computationData;  
    }  
  
    public void setComputationData(int computationData) {  
        this.computationData = computationData;  
    }  
  
    @Override  
    public String toString() {  
        return "DataAnalyticsResult{" + "arrayIndexData=" + arrayIndexData + ", computationData=" + computationData + '}';  
    }  
}
```

Constructors with no-arg and full-arg constructors, Accessors and mutators of DataAnalyticsResult.

Snapshot 4.2:

```
    return arrayIndexData;  
}  
  
public void setArrayIndexData(int arrayIndexData) {  
    this.arrayIndexData = arrayIndexData;  
}  
  
public int getComputationData() {  
    return computationData;  
}  
  
public void setComputationData(int computationData) {  
    this.computationData = computationData;  
}  
  
@Override  
public String toString() {  
    return "DataAnalyticsResult{" + "arrayIndexData=" + arrayIndexData + ", computationData=" + computationData + '}';  
}  
}
```

DataAnalyticsResult setters, getters and toString() method

5. WinterSolstice Comparator

Snapshot 5.1:

```
*/
public class WinterSolstice implements Comparator {

    public int compare(Object o1, Object o2) {
        DataAnalyticsResult ws1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult ws2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then return 1
        //if second is greater then return -1, else return 0
        if (ws1.getComputationData() > ws2.getComputationData()) {
            return 1;
        }
        if (ws1.getComputationData() < ws2.getComputationData()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

Comparator used to find
the shortest day in Winter
Solstice

6. SummerSolstice Comparator

Snapshot 6.1:

```
*/
public class SummerSolstice implements Comparator {

    public int compare(Object o1, Object o2) {
        DataAnalyticsResult ss1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult ss2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then return -1
        //if second is greater then return 1, else return 0
        if (ss1.getComputationData() > ss2.getComputationData()) {
            return -1;
        }
        if (ss1.getComputationData() < ss2.getComputationData()) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Comparator used for
computation of longest
day in Summer Solstice.

7. VernalandAutumnal Comparator

Snapshot 7.1:

```

-  */
  public class VernalandAutumnalEquinox implements Comparator {

      @Override
  ]  public int compare(Object o1, Object o2) {
        DataAnalyticsResult vae1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult vae2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then return 1
        //if second is greater then return -1, else return 0
        if (vae1.getComputationData() > vae2.getComputationData()) {
            return 1;
        }
        if (vae1.getComputationData() < vae2.getComputationData()) {
            return -1;
        } else {
            return 0;
        }
    }
}

```

Comparator used for
the computation of
equal day and night in
Equinox

8. Utilities class:

Snapshot 8.1:

```

*
* @author Meghashree M Ramachandra
*/
public class Utilities {

    //to read data from file
    public static List readCSVfile() {
        BufferedReader br;
        String line;
        DaylightRecord dlrData;
        DateFormat format = new SimpleDateFormat("MM/dd/yyyy HHmm");
        List daylightRecords = new ArrayList<DaylightRecord>();
        int month;

        try {
            // Read data in from CSV file...
            br = new BufferedReader(new FileReader("data/sunrise-sunset.csv"));
            String dateSunrise;
            String dateSunset;
            while ((line = br.readLine()) != null) {
                // Extract tokens from CSV line with comma delimiter...
                month = 1;
                StringTokenizer st = new StringTokenizer(line, ",");
            }
        }
    }
}

```

Function to read data
from files and store in
arraylist

Snapshot 8.2:

```

String sunSetData = st.nextToken();
if (sunRiseData.length() == 3) {
    sunRiseData = "0" + sunRiseData;
}
if (sunSetData.length() == 3) {
    sunSetData = "0" + sunSetData;
}
dateSunrise = Integer.toString(month) + "/" + day + "/2013 " + sunRiseData;
dateSunset = Integer.toString(month) + "/" + day + "/2013 " + sunSetData;
if ((day.equals("29") || day.equals("30")) && month == 2) {
    daylightRecords.add(null);
    dlrData = new DaylightRecord(day, format.parse(dateSunrise), format.parse(dateSunset));
    daylightRecords.add(dlrData);
    month += 2;
} else if ((day.equals("31")) && (month == 2 || month == 4 || month == 6 || month == 9 || month == 11)) {
    daylightRecords.add(null);
    dlrData = new DaylightRecord(day, format.parse(dateSunrise), format.parse(dateSunset));
    daylightRecords.add(dlrData);
    month += 2;
} else {
    dlrData = new DaylightRecord(day, format.parse(dateSunrise), format.parse(dateSunset));
    daylightRecords.add(dlrData);
}

```

code snippet to read data from file and store in an arraylist

Snapshot 8.3:

```

//to serialize
public static void serializeListObject(PersistentObject pObject) {
    ObjectOutputStream oos;
    try {
        oos = new ObjectOutputStream(new FileOutputStream("data/daylight-record.ser"));
        oos.writeObject(pObject);
    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}

//to deserialize
public static PersistentObject deserializeListObject() {
    ObjectInputStream ois = null;
    PersistentObject pObject = new PersistentObject();
    try {
        // Serialize records as a aggregate...
        ois = new ObjectInputStream(new FileInputStream("data/daylight-record.ser"));
        pObject = (PersistentObject) ois.readObject();
    } catch (ClassNotFoundException | IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

code snippet to serialize and deserialize the PersistentObject

Snapshot 8.4:

```
//make app sleep for 10 seconds
public static void makeAppSleep() {
    try {
        System.out.println("Waiting 10 seconds...");
        Thread.sleep(10000);
    } catch (InterruptedException ex) {
        Logger.getLogger(Mp2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Code snippet to make app sleep

⇒ for 10seconds

Snapshot 8.5:

```
public static StringBuilder createCSVfile(PersistentObject pob) {
    DateFormat df = new SimpleDateFormat("HHmm");
    StringBuilder sb = new StringBuilder();
    int day = 1;
    DaylightRecord d;
    for (Object r : pob.getDayLightRecord()) {

        d = (DaylightRecord) r;
        // sb.append(pob.getDateObj() + ",");
        if (d != null) {
            if (Integer.toString(day).equals(d.getDay().toString())) {
                if (day != 1) {
                    sb.append("\n");
                }
                sb.append(d.getDay().toString() + ",");
                sb.append(df.format(d.getSunRise()) + ",");
                sb.append(df.format(d.getSunSet()) + ",");
                day++;
            } else {
                sb.append(df.format(d.getSunRise()) + ",");
                sb.append(df.format(d.getSunSet()) + ",");
            }
        }
    }
}
```

code snippet to create '.csv' file
after deserialization of the
PersistentObject

Snapshot 8.6:

```
//to write csv file
public static void writeCSVfile(StringBuilder sb) {
    try {
        FileWriter fw;
        fw = new FileWriter("data/deserializedDaylightRecords.csv");
        fw.write(sb.toString(), 0, sb.length());
        fw.flush();
    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Write deserialized data into
deserializedDaylightRecords.csv file after
creating it from createcsvfile() method

Snapshot 8.7:

```
//to compute data analytics for winter and summer
public static List findCorrespondingData(List daylightRecords, String startDate, String endDate, boolean checkForSolstice) {
    List computationList = new ArrayList<DataAnalyticsResult>();
    int value;
    for (int index = 0; index < daylightRecords.size(); index++) {
        if (daylightRecords.get(index) == null) {
            index++;
            continue;
        } else {
            value = Utilities.computeSolsticeDataAnalytics(daylightRecords.get(index), startDate, endDate, checkForSolstice);
            if (value != 0) {
                computationList.add(new DataAnalyticsResult(index, value));
            }
        }
    }
    if (checkForSolstice == true) {
        Collections.sort(computationList, new WinterSolstice());
    } else {
        Collections.sort(computationList, new SummerSolstice());
    }
    return computationList;
}
```

Compute and collects the
data of day length into the
arraylist for winter and
summer solstice

Snapshot 8.8:

```
//to compute data analytics for spring and fall
public static List findRelevantData(List daylightRecords, String startDate, String endDate) {
    List computationListEqualDayNight = new ArrayList<DataAnalyticsResult>();
    int nextDaySunriseDataIndex;
    int valueResult;
    for (int index = 0; index < daylightRecords.size(); index++) {

        if (daylightRecords.get(index) == null) {
            index++;
            continue;
        } else {
            nextDaySunriseDataIndex = index + 12;
            if (nextDaySunriseDataIndex >= daylightRecords.size()) {
                nextDaySunriseDataIndex = 0;
            } else if (daylightRecords.get(nextDaySunriseDataIndex) == null) {
                nextDaySunriseDataIndex++;
            } else {
                valueResult = Utilities.computeEquinoxDataAnalytics(daylightRecords.get(index), daylightRecords.get(nextDaySunriseDataIndex), startDate, endDate);
                if (valueResult != -1) {
                    computationListEqualDayNight.add(new DataAnalyticsResult(index, valueResult));
                }
            }
        }
    }
    Collections.sort(computationListEqualDayNight, new VernalandAutumnalEquinox());
    return computationListEqualDayNight;
}
```

computes and collects data of difference between day and night length to an arraylist for data analytics of equal day night in Equinox

Snapshot 8.9:

```
//compute data analytics for the specified date range and aggregate the data(for summer and winter)
public static int computeSolsticeDataAnalytics(Object o, String startDayofSolstice, String endDayofSolstice, boolean checkForSolstice) {
    int finalValue = 0;
    DaylightRecord dr1 = (DaylightRecord) o;

    String winterSolsticeStart = startDayofSolstice;
    String winterSolsticeEnd = endDayofSolstice;
    DateFormat df1 = new SimpleDateFormat("MM/dd/yyyy");
    try {
        if (checkForSolstice == true) {
            if (dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 || dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0) {
                finalValue = Integer.parseInt(findDayLength(dr1));
            }
        } else if (checkForSolstice == false) {
            if ((dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 && dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0)) {
                finalValue = Integer.parseInt(findDayLength(dr1));
            }
        }
    } catch (ParseException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

compute and retrieve the daylength for all the dates for winter and summer solstice

Snapshot 8.10:

```
//compute day length
public static String findDayLength(DaylightRecord dr1) {
    long d;

    d = Math.abs(dr1.getSunRise().getTime() - dr1.getSunSet().getTime());
    int Hours = (int) ((d / (1000 * 60 * 60)) % 24);
    int Minutes = (int) ((d / (1000 * 60)) % 60);
    String s;
    if (Integer.toString(Minutes).length() == 1) {
        s = "0" + Integer.toString(Minutes);
        s = Integer.toString(Hours) + s;
    } else {
        s = Integer.toString(Hours) + Integer.toString(Minutes);
    }
    return s;
}
```

to find the daylength for a particular date.

Snapshot 8.11:

```
//compute data analytics for the specified date range and aggregate the data(for spring and fall)
public static int computeEquinoxDataAnalytics(Object o1, Object o2, String equinoxStartDate, String equinoxEndDate) {

    int finalValue = -1;
    DaylightRecord dr1 = (DaylightRecord) o1;
    DaylightRecord dr2 = (DaylightRecord) o2;
    long d1;
    String winterSolsticeStart = equinoxStartDate;
    String winterSolsticeEnd = equinoxEndDate;
    DateFormat df1 = new SimpleDateFormat("MM/dd/yyyy");
    try {
        if (dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 && dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0 && dr2.getSunRise().compareTo(df1.parse(winterSolsticeStart)) <= 0 && dr2.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) >= 0) {
            d1 = Math.abs(dr1.getSunRise().getTime() - dr1.getSunSet().getTime());
            int dayHours = (int) ((d1 / (1000 * 60 * 60)) % 24);
            int dayMinutes = (int) ((d1 / (1000 * 60)) % 60);

            int timeHoursDifference = 0;
            int timeMinutesDifference = 0;
            String s;
            if (dayHours >= 12 && dayMinutes >= 0) {
                if (dayHours == 12 && dayMinutes == 0) {
                    s = "0" + Integer.toString(timeMinutesDifference);
                    s = Integer.toString(timeHoursDifference) + s;
                } else {
                    s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
                }
                finalValue = Integer.parseInt(s);
            }
        } else {
            timeHoursDifference = 12 - dayHours;
            timeMinutesDifference = 00 - dayMinutes;
            if (timeMinutesDifference < 0) {
                timeHoursDifference = timeHoursDifference - 1;
                timeMinutesDifference = timeMinutesDifference + 60;
            }
            if (Integer.toString(timeMinutesDifference).length() == 1) {
                s = "0" + Integer.toString(timeMinutesDifference);
                s = Integer.toString(timeHoursDifference) + s;
            } else {
                s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
            }
            finalValue = Integer.parseInt(s);
        }
    } catch (Exception e) {
        finalValue = -1;
    }
}
```

Compute and retrieve the difference between day length and night length which can be used to decide the equal day and night in equinox.

Snapshot 8.12:

```
        s = "0" + Integer.toString(timeMinutesDifference);
        s = Integer.toString(timeHoursDifference) + s;
    } else {
        s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
    }
    finalValue = Integer.parseInt(s);
}
} else {
    timeHoursDifference = 12 - dayHours;
    timeMinutesDifference = 00 - dayMinutes;
    if (timeMinutesDifference < 0) {
        timeHoursDifference = timeHoursDifference - 1;
        timeMinutesDifference = timeMinutesDifference + 60;
    }
    if (Integer.toString(timeMinutesDifference).length() == 1) {
        s = "0" + Integer.toString(timeMinutesDifference);
        s = Integer.toString(timeHoursDifference) + s;
    } else {
        s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
    }
    finalValue = Integer.parseInt(s);
}
}
```

continued code snippet of computeEquinoxDataAnalytics().

Snapshot 8.13:

```
//retrieve required data for display
public static ArrayList retrieveDataFromComputation(List newList, List dataList) {
    DateFormat df = new SimpleDateFormat("dd MMMM yyyy");
    ArrayList finalValues = new ArrayList();
    DataAnalyticsResult dar;
    DaylightRecord dlr;
    int firstTime = 0;
    int data = 0;
    int repeat = 1;
    for (Object o : newList) {
        dar = (DataAnalyticsResult) o;
        dlr = (DaylightRecord) dataList.get(dar.getArrayIndexData());

        if (firstTime == 0) {
            data = dar.getComputationData();
            firstTime = 1;
            finalValues.add(df.format(dlr.getSunRise()).toString());
        } else {
            if (data == dar.getComputationData()) {
                repeat++;
                finalValues.add(df.format(dlr.getSunRise()).toString());
            }
        }
    }
    finalValues.add(repeat);
    return finalValues;
}

//display computed data
```

Retrieve required data to display
from all the computation of data
analytics results

Snapshot 8.14:

```
//display computed data
public static void printData(ArrayList computationData) {
    int repeatData = computationData.size();
    for (int m = 0; m < repeatData - 1; m++) {
        System.out.print(computationData.get(m) + ",");
    }
}

//write daylightrecords to file
public static void writeListToFile(List daylightRecords) {
    try {
        BufferedWriter out = null;
        File file = new File("output/daylight-records.txt");
        out = new BufferedWriter(
            new FileWriter(file));
        for (Object s : daylightRecords) {
            // Write line to file.
            if (s != null) {
                DaylightRecord d = (DaylightRecord) s;
                out.write(d.toString());
            } else {
                out.write("null");
            }
            // Write newLine with BufferedReader method.
            out.newLine();
        }
        out.close();
    } catch (IOException ex) {
        Logger.getLogger(Mp2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Code snippet to print the final data for all data
analytics and to write the daylightRecords list
data into daylight-records.txt

9. Driver (MP2) class:

Snapshot 9.1:

```

public static void main(String[] args) {

    // Read CSV file from filesystem...
    System.out.println("Read CSV file from filesystem");
    List daylightRecords = Utilities.readCSVfile();

    //write all daylightRecords to text file daylight-Record.txt
    Utilities.writeListtoFile (daylightRecords);

    //Instance of PersistentObject with the current timestamp and the ArrayList object
    PersistentObject pObj = new PersistentObject(new Date(), daylightRecords);

    //Serialize persistentobject pObj as a aggregate...
    System.out.println("Serialize records as a aggregate");
    Utilities.serializeListObject(pObj);

    //Wait 10 secs...
    Utilities.makeAppSleep();

    // Deserialize records back into app...
    System.out.println("Deserialize records back into app");
    PersistentObject pObject = Utilities.deserializeListObject();

    // Print time difference
    System.out.print("Time between serialization and deserialization is: " + Math.abs((pObj.getDateObj().getTime() - System

```

Driver class which initiates methods to read data from csv file, create instance of PersistentObject, serialize, deserialize persistentObject, to make app sleep for 10 seconds, and to write data of daylightRecords arraylist to file name daylight-Records.txt

Snapshot 9.2:

```

System.out.print("Time between serialization and deserialization is: " + Math.abs((pObj.getDateObj().getTime() - System.currentTimeMillis()) / 1000) + " secs");

// Create CSV file from records...
System.out.println("Create CSV file from records");
StringBuilder sb = Utilities.createCSVFile(pObject);

// Write CSV file to filesystem...
System.out.println("Write CSV file to filesystem");
Utilities.writeCSVFile(sb);

System.out.println("\nBy processing the given data :");
//Compute Shortest day => winter solstice
List computationListForShortestDay = Utilities.findCorrespondingData(daylightRecords, "01/01/2013", "12/31/2013", true);
ArrayList computationData = Utilities.retrieveDataFromComputation(computationListForShortestDay, daylightRecords);
System.out.print("Shortest Day -> Winter Solstice\nFound ");
Utilities.printData(computationData);
System.out.println(" with same day length as the shortest days of Winter.\n" + computationData.get(0) + " is the first shortest day of Winter.\n");

//Equal day and night => vernal equinox (Spring)
List computationListEqualDayNightForSpring = Utilities.findRelevantData(daylightRecords, "03/01/2013", "06/30/2013");
computationData = Utilities.retrieveDataFromComputation(computationListEqualDayNightForSpring, daylightRecords);
System.out.print("Equal Day and night -> Vernal Equinox(Spring)\nNo days have equal day and night.\nFound");

```

Driver class which initiates function 1. to create and write the '.csv' file, 2. to compute data analytics and display the shortest day for Winter Solstice

Snapshot 9.3:

```
//Equal day and night => vernal equinox (Spring)
List computationListEqualDayNightForSpring = Utilities.findRelevantData(daylightRecords, "03/01/2013", "06/30/2013");
computationData = Utilities.retrieveDataFromComputation(computationListEqualDayNightForSpring, daylightRecords);
System.out.print("Equal Day and night -> Vernal Equinox(Spring)\nNo days have equal day and night.\nFound");
Utilities.printData(computationData);
System.out.println(" as the days closest to be an Equinox with difference of 1 min.\n" + computationData.get(0) + " is the first Vernal Equinox.\n");

//Longest day => summer solstice
List computationListForLongestDay = Utilities.findCorrespondingData(daylightRecords, "01/01/2013", "12/31/2013", false);
computationData = Utilities.retrieveDataFromComputation(computationListForLongestDay, daylightRecords);
System.out.print("Longest Day -> Summer Solstice\nFound ");
Utilities.printData(computationData);
System.out.println(" with same day length as the longest days of Summer.\n" + computationData.get(0) + " is the first longest day of Summer.\n");

//compute Equal day and night => autumnal equinox(Fall)
List computationListEqualDayNightForFall = Utilities.findRelevantData(daylightRecords, "09/01/2013", "12/30/2013");
System.out.println("Equal Day and night -> Autumnal Equinox(Fall)");
computationData = Utilities.retrieveDataFromComputation(computationListEqualDayNightForFall, daylightRecords);
System.out.println(computationData.get(0) + " is the day which have Equal day and night for Autumnal Equinox\n");
```

Driver class which initiates the method to compute the longest day in Summer, equal day and night in Vernal and Autumnal Equinox data analytics.

10. Output:

Snapshot 10.1:

```
run:
Read CSV file from filesystem
Mar 10, 2013 4:19:35 PM domain.util.Utilities readCSVfile
SEVERE: null
Creating the Instance of PersistentObject with the current timestamp and the ArrayList object.
Serialize records as a aggregate
java.io.FileNotFoundException: data\sunrise-sunset.csv (The system cannot find the file specified)
    at java.io.FileInputStream.open(Native Method)
Waiting 10 seconds...
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:97)
    at java.io.FileReader.<init>(FileReader.java:58)
    at domain.util.Utilities.readCSVfile(Utilities.java:52)
    at driver.Mp2.main(Mp2.java:26)
```

Example of use of logger used in the code so if error is thrown, one can easily find out the line where error occurred.

Snapshot 10.2:

<pre> Read CSV file from filesystem Creating the Instance of PersistentObject with the current timestamp and the ArrayList object. Serialize records as a aggregate Waiting 10 seconds... Deserialize records back into app Time between serialization and deserialization is: 11 secs. Create CSV file from records Write CSV file to filesystem writing arraylist daylightRecords data into daylight-records.txt </pre>	<p>Result of reading csv file from filesystem, serialization of instance of persistentobject, make app sleep 10 seconds, deserialize of persistentobject, time difference between serialization and deserialization, creating and writing data into csv file after deserialization, writing data of daylightRecords into daylight-records.txt.</p>
---	--

Note: Below data analytics results shown in Snapshot 10.3, are purely based on the data given to us in sunrise-sunset.csv file. But according to many official websites:

Winter Solstice is - December 21, 2013

Summer Solstice – June 21, 2013

Equal Day and Night (Vernal Equinox) – March 20, 2013

Equal Day and Night (Autumnal Equinox) – September 22, 2013

Snapshot 10.3:

<pre> By processing the given data : Shortest Day -> Winter Solstice Found 17 December 2013,19 December 2013,20 December 2013,21 December 2013,23 December 2013,25 December 2013, with same day length as the shortest day 17 December 2013 is the first shortest day of Winter. Equal Day and night -> Vernal Equinox(Spring) No days have equal day and night. Found 16 March 2013,17 March 2013, as the days closest to be an Equinox with difference of 1 min. 16 March 2013 is the first Vernal Equinox. Longest Day -> Summer Solstice Found 18 June 2013,19 June 2013,23 June 2013, with same day length as the longest days of Summer. 18 June 2013 is the first longest day of Summer. Equal Day and night -> Autumnal Equinox(Fall) 25 September 2013 is the day which have Equal day and night for Autumnal Equinox </pre>	<p>Result of data analytics:</p> <ol style="list-style-type: none"> 1. shortest day in winter. 2. equal day and night in vernal equinox. 3. longest day in summer. 4. equal day and night in autumnal equinox
--	---