Software Engineering Group Projects – Design Specification

Author: C. J. Price and B.P.Tiddeman
Config Ref: SE_01_DS_001
Date: 29th September 2013
Version: 1.02
Status: Draft

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION 2
   1.1 Purpose of this Document 2
   1.2 Scope 2
   1.3 Objectives 2
2. ARCHITECTURAL DESCRIPTION 3
   2.1 Programs in system 3
      2.1.1 The DIARY program 3
      2.1.2 The HEATING program 3
      2.1.3 The LOG READING Program 3
   2.2 Significant classes 3
      2.2.1 Significant classes in the DIARY program 3
      2.2.2 Significant classes in the HEATING program 3
      2.2.3 Significant classes in the LOG READING Program 4
      2.2.4 Significant support classes (used by more than one of the programs) 4
3. DEPENDENCY DESCRIPTION 5
   3.1 Component Diagrams 5
      3.1.1 Component Diagram for DIARY program 5
      3.1.2 Component Diagram for HEATING program 5
      3.1.3 Component Diagram for LOG program 6
   3.2 Compilation / inheritance dependencies 6
4. CLASS INTERFACE DESCRIPTION 7
   4.1 Class Interface for Diary 7
   4.2 Class Interface for DiaryAccess 9
   4.3 Class Interface for DiaryItem (and subclasses for meeting types) 10
   4.4 Class interface for Heating (also DayBooking/HeatingEvent) 14
   4.5 Class interface for LogReading 17
   4.6 Class interface for LogAccess 18
   4.7 Class interface for LogItem 19
5. DETAILED DESIGN 20
      5.1.1 The DIARY program 20
      5.1.2 The HEATING program 20
      5.1.3 The LOG READING Program 20
6. REFERENCES 20

1. INTRODUCTION

1.1 Purpose of this Document

This document describes the outline design for the Software Engineering Group Project 2001. It should be read in the context of the Group Project, taking into account the details of the group project assignment and the group project quality assurance (QA) plan [1].

1.2 Scope

The design specification breaks the project into separately implementable components and describes the interfaces to and interaction between those components. It is cross referenced to the Requirements Specification for the group project. References to specific requirements are given in round brackets, e.g. (EIR1) refers to External interface requirement EIR1. This document should be read by all members of the implementation team.

1.3 Objectives

The objectives of this document are:

- To describe the main components of the Automated Church Heating Environment (ACHE)
- To depict the dependencies between the components, both for compilation, and in execution
- To provide interface details for each of the main classes in the ACHE

2. ARCHITECTURAL DESCRIPTION

2.1 Programs in system

The ACHE is composed of three programs:

- The DIARY program
- The HEATING program
- The LOG READING program

2.1.1 The DIARY program

The Diary program provides the interface to enable users to make, examine, and amend diary bookings. It implements requirements (FR1), (FR2), (FR3) and must make sure it conforms with requirements (FR5), (FR9), (EIR5), (DC1), (DC2), (PR2).

The program will have a form-based method of entry as described in (EIR5). That interface is not described in detail in this document, but will be determined during implementation. It will be implemented in Swing. The Diary program will call the support class DiaryAccess to read and write the diary text file mentioned in (FR1), and use the support class DiaryItem to represent a single diary entry.

2.1.2 The HEATING program

The Heating program reads the diary file and turns the physical heating on and off. It writes a log of significant events for later examination. It implements requirements (FR4), (FR6), (FR7), (FR8), (FR10), (FR11), (EIR1), (EIR4), and must make sure it conforms with requirements (FR5), (FR9), (EIR2), (EIR3), (DC1), (DC2), (PR1), (PR2), (DC1), (DC2).

The Heating program must be able to deal with each of the different types of diary entry describe in (FR2), and given in greater detail in Appendix A. It will call the support class DiaryAccess to obtain the details of the diary entries, and use the support class DiaryItem to represent a single diary entry. It will call the support class LogAccess to write the details of the log entries, and use the support class LogItem to represent a single log entry.

2.1.3 The LOG READING Program

The Log Reading program allows the user to choose a log file for a specific day, and displays the contents, letting the user scroll the display up and down if that is necessary in order to see all of the log contents. It implements requirement (FR12). It will call the support class LogAccess to obtain the details of the log entries, and use the support class LogItem to represent a single log entry.

2.2 Significant classes

2.2.1 Significant classes in the DIARY program

Diary. This will be the main class of the Diary program.

2.2.2 Significant classes in the HEATING program

Heating. This will be the main class of the Heating program.

DayBooking. This will hold a list of HeatingEvents detailing all of the changes to the Heating which need to be made today.

HeatingEvent. This is a single event in the list of events in DayBooking. It tells you which rooms need heating turned on or off and why.

2.2.3 Significant classes in the LOG READING Program

LogReading. This will be the main class of the Log Reading program.

2.2.4 Significant support classes (used by more than one of the programs)

DiaryAccess. This class handles the details of a meeting diary file. It provides the ability to read or write a meeting diary file. It holds a list of DiaryItem objects and can turn them into a textual file, or alternatively, read a meeting diary file and obtain a list of DiaryItem objects.

DiaryItem. This class contains the information representing a single booking diary entry. It cannot be directly instantiated, but has subclasses for each type of booking - SingleBooking, WeeklyBooking, OddWeeklyBooking, MonthlyBooking. These subclasses have different routines for reading/writing details, and for calculating whether a booking occurs on a given date.

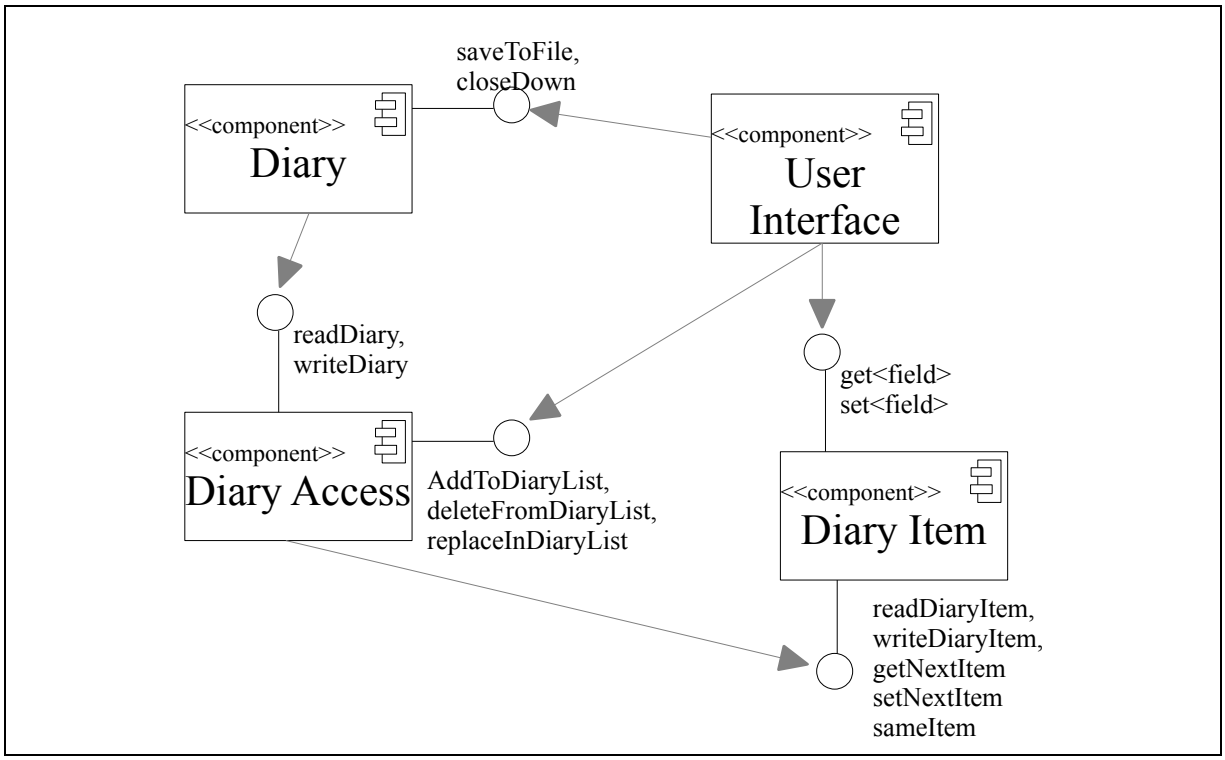
LogAccess. This class handles the details of a log file. It provides the ability to read or write a log file. It holds a list of LogItem objects, and can turn them into a file of LogItem objects. It can read a log file and obtain a list of LogItem objects.

LogItem. This class contains the information representing a single log file entry. It is able to read/write its details to an open file, and to return text strings of its contents.

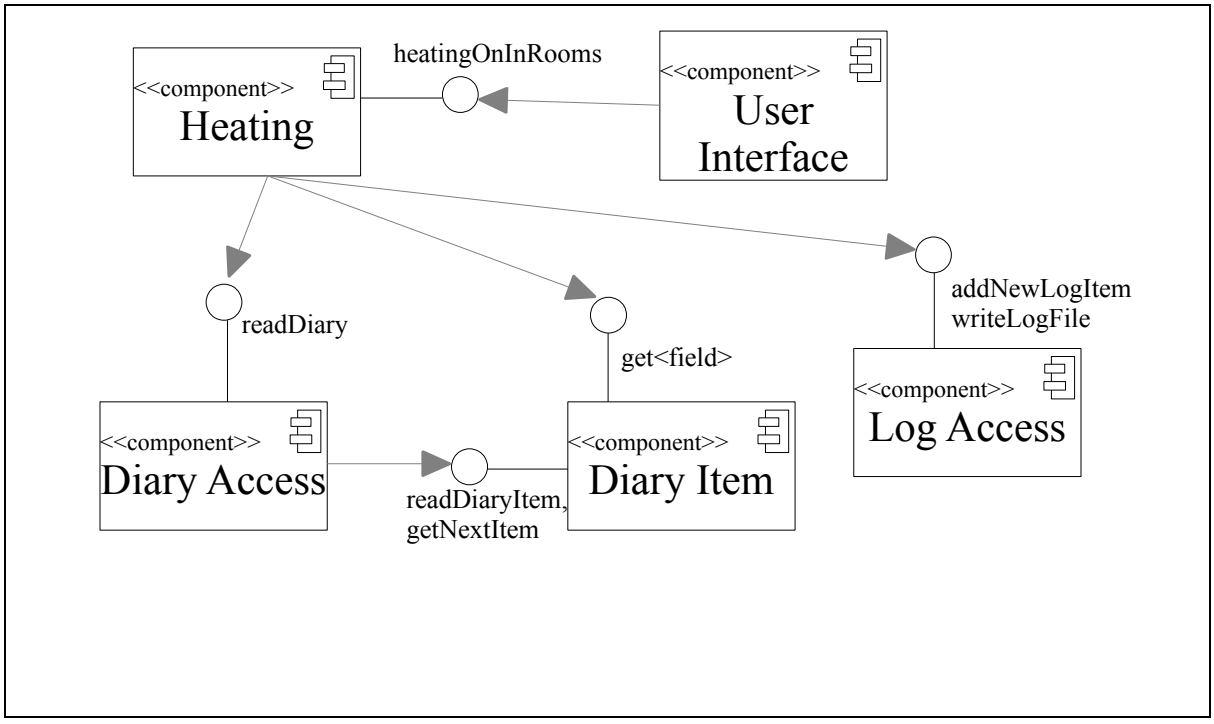
3. DEPENDENCY DESCRIPTION

3.1 Component Diagrams

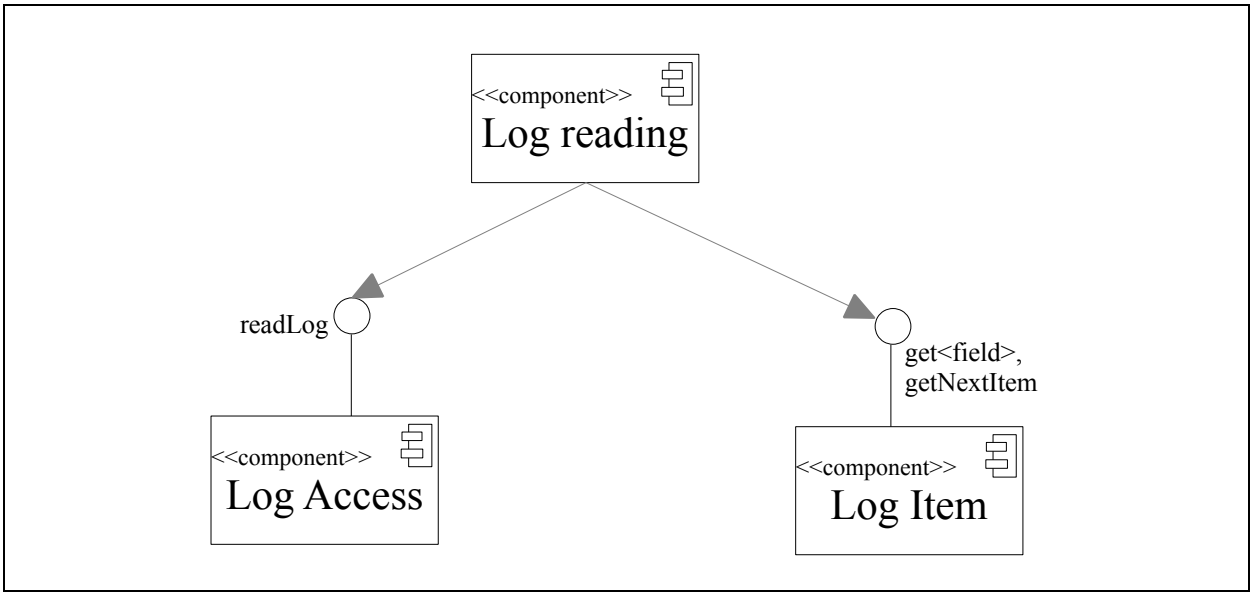
3.1.1 Component Diagram for DIARY program



3.1.2 Component Diagram for HEATING program



3.1.3 Component Diagram for LOG program



3.2 Compilation / inheritance dependencies

There are very few compilation dependencies:

- Diary depends on DiaryAccess and DiaryItem.
- Heating depends on DiaryAccess, DiaryItem, LogAccess and LogItem.
- LogReading depends on LogAccess and LogItem.

Classes are independent, except that SingleBooking, WeeklyBooking, OddWeeklyBooking, MonthlyBooking are subclasses of DiaryItem.

4. CLASS INTERFACE DESCRIPTION

4.1 Class Interface for Diary

```
import uk.ac.aber.cs221.group01.global.utils.DiaryItem
import uk.ac.aber.cs221.group01.global.utils.DiaryAccess

/** Outline for Diary booking class

 * This design does not prescribe the user interface
 * for the diary booking program
 * It assumes that there will be a user interface
 * to collect and verify the information
 * for each type of booking.

 * The application starts the user interface and reads a diary file
 * It provides facilities for the user interface to close down the program
 */
public class Diary {
   protected myDiary DiaryAccess;
   private String givenFileName

   /* If there is a file named as args[0],
    * then use the contructor with that file name as a parameter, else give
    * a default file name
    * @param args the command line argument array
    */
   public static void main(String args[]) { }

   /** Create myDiary
    *
    * Remember fileName in givenFilename, for use on Closedown
    * Create the user interface - pass it 'this' as parameter so it can use myDiary
    * @param fileName If fileName exists, use it to read in the contents of myDiary
    */
    public static void Diary( String fileName ) { }

    /** Create file fileName
      * Write  contents of myDiary to the file
      * Close file
      * @param fileName name of file to write to
      */
    public static void saveToFile ( String fileName ) { }

 /** Call saveToFile with givenName as parameter */
    public static void closeDown ( ) { }


   /* Diary might contain a bunch of checking routines for fields, but they
    * might more appropriately be placed in the user interface class(es)
    *
    * When the user interface wants to change DiaryItem values,
    * it should NOT change them directly.
    * It should take a copy and change the copy.
    *
    * When it wants to update the Diary,
    * it should call methods like replaceInDiaryList
    * with the appropriate parameters
    */

}
```

4.2 Class Interface for DiaryAccess

```
package uk.ac.aber.cs221.group01.global.utils;

import uk.ac.aber.cs221.group01.global.utils.DiaryItem;

/** Outline for DiaryAccess class
 *
 * This class handles the details of a meeting diary file.
 * It provides the ability to read or write a meeting diary file.
 * It holds a list of DiaryItem objects and can turn them into a textual file,
 * or alternatively, read a meeting diary file and obtain a list of DiaryItem
 * objects.
 *
 *It provides routines to add, delete, amend the list of entries.
 */
public class DiaryAccess {
   private DiaryItem diaryList;

/** Constructor - should initialise DiaryList to null */
   public DiaryAccess() {}

   /** This should open fileName and read contents into a set of
     * DiaryItems in DiaryList
     * For each item it decides what type it is,
     * creates right type of object and passes
     * it the complete relevant line from the file.
     * @param fileName name of file to read from
     */
    public static void readDiary( String fileName){}

  /** This should create fileName and write the
    * DairyItems in DiaryList to the file.
    * @param fileName name of file to write to
    */
   public static void writeDiary( String fileName){}

  /** Once a new diaryItem has been created,
    * this can be used to add it to diaryList.
    * @param thisItem item to add to diary
    */
   public static void addToDiaryList(DiaryItem  thisItem){}

   /** thisItem is compared to the list,
     * and the first matching booking is deleted
     * @param thisItem the item to delete
     */
   public static void deleteFromDiaryList(DiaryItem  thisItem){}

  /** oldItem is compared to list, and first matching booking
    * is replaced with newItem
    * This is used to implement the Amend feature
    * @param oldItem old item to replace
    * @param newItem new item to insert
    */
   public static void replaceInDiaryList(DiaryItem oldItem, DiaryItem newItem){}

}
```

4.3 Class Interface for DiaryItem (and subclasses for meeting types)

```
package uk.ac.aber.cs221.group01.global.utils;

import uk.ac.aber.cs221.group01.global.utils.CentralHeating;

/** Outline for DiaryItem class */
 * This class handles the details of a single meeting diary entry.
 */
public abstract class DiaryItem {
   private DiaryItem nextItem = NULL;  // used to make a list of these things
   private String itemName;  // The string describing what the diary entry is for
   private Time startTime;  // The time that the booking starts at
   private Time endTime;  // The time that the booking ends at
   private String [] usedRooms; // Names of the rooms that are to be heated
   private Date startDate; // The first date for the booking

   /** This reads the info for this DiaryItem from a given string
     * and fills it in
     * @param detailString a string containing info about the booking
   public abstract void readDiaryItem(String detailString){}

   /** This should return the SingleMeeting as a string for the diary file.
     * @return returns a single meeting as a string
     */
   public abstract String writeDiaryItem(){}

  /** Compares this diaryItem's details with otherItem.
    * Would be sensible to compare that it is of same class first!
    * @param otherItem item to compare this with
    * @return true if it is the same item, false otherwise
    */
   public abstract boolean sameItem(DiaryItem otherItem){}

  /** Check if the item occurs on a specific date
    * @param theDate the date to check
    * @return returns true if this DiaryItem occurs on theDate, else false*/
   public abstract boolean itemOccursOnDate(Date theDate){}

   /** Get the next item
     * @return Returns item pointed to by nextItem
     */
   public DiaryItem getNextItem(){}

   /** Sets nextItem to theNext
    *  @param theNext the item to use next
    */
   public void setNextItem(DiaryItem theNext) {
   }

  /** Get the item's name
    * @return Returns the string describing what the diary entry is for
    */
   public String getItemName() {}

  /** Sets the string describing what the diary entry is for
    * @param description the description of the diary item to use */
   public void setItemName(String description) { }

   /** Returns the time that the booking starts at */
   public Time getStartTime() { }

   /** sets the time that the booking starts at
     * @param theTime the time that the booking starts
```

```
     */
    public void setStartTime(Time theTime) {}

/** Get the end time
  * @return Returns the time that the booking ends at
  */
   public Time getEndTime() {}

 /** Sets the time that the booking ends at
   * @param  theTime the time that the booking should end
   */
   public void setEndTime(Time theTime) {}

 /** Get the names of the rooms used
   * @return Returns names of the rooms that are to be heated
   */
  public String[] getUsedRooms() {
   }

 /** Sets names of the rooms that are to be heated
   * @param theRooms the array of room names to be heated
   */
   public setUsedRooms(String[] theRooms) {}

  /** Returns the first date for the booking
    * @return returns the start date
    */
   public Date getStartDate() {}

   /** Sets the first date for the booking
     * @param theDate the value to use as the startDate
     */
   public void setStartDate(Date theDate) { }

}

/** A class to represent a single meeting
  * [Had to decide whether to make DiaryItem with lots of abstracts,
  * or to have SingleMeeting methods in DiaryItem, and then
  * use super to call them for the more complex classes, decided on separate class]
  */
public class SingleMeeting extends DiaryItem {
   /** This reads the info for a SingleMeeting
    * from a given string and fills it in
    * @param detailString the detailed information string
    */
   public void readDiaryItem(String detailString){}

  /** Write this diary item
    * @return return the SingleMeeting as a string for the diary file.
    */
   public String writeDiaryItem(){}

   /** Compares this SingleMeeting's details with OtherItem.
     * Would be sensible to compare that it is of same class first!
     * @param diaryItem
     * @return true if they are the same item
   */
   public boolean sameItem(DiaryItem otherItem){}

   /** Check if the item occurs on a specific date
     * @param theDate the date to check
     * @return returns true if this DiaryItem occurs on theDate, else false
     */
   public boolean itemOccursOnDate(Date theDate){}
```

```
}

/** Class for a repeated meeting
  */
public class RepeatedMeeting extends DiaryItem {
   private date endDate; /* The last date for the booking */
   private String[] daysOfWeek; /* Which days of the week the booking will be */

   /** Returns the last date for the booking
    * @return returns the last date for the booking
   */
   public Date getEndDate() {}

   /** Sets the last date for the booking
    * @param theDate the desired end date
    */
   public void setEndDate(Date theDate) {}

   /** Get the days of week for this booking
    * @return returns strings with days of week that booking will happen
    */
   public String[] getDaysOfWeek() {}

   /** Sets strings with days of week that booking will happen
    * @param theDays the desired days of the week
    */
    public void getDaysOfWeek(String[] theDays) {}
}

/** Weekly meeting class
public class WeeklyMeeting extends RepeatedMeeting {

   /** Reads the info for a WeeklyMeeting from a given string and fills it in
    * @param detailString the string to get the info from
    */
   public void readDiaryItem(String detailString){}

   /** Writes this diary item
    * @return should return the WeeklyMeeting as a string for the diary file.
    */
   public String writeDiaryItem(){}

   /** Compares this WeeklyMeeting's details with OtherItem.
    * Would be sensible to compare that it is of same class first!
    * @param otherItem item to compare with
    * @return returns true if they are the same, false otherwise
    */
   public boolean sameItem(DiaryItem otherItem){}
}

/** Class for meeting that occur at some interval of weeks
 */
public class OddWeeklyMeeting extends RepeatedMeeting {
   public int weeksGap; /* Length of gap in weeks between meetings */

   /** This reads the info for an OddWeeklyMeeting
     * from a given string and fills it in
     * @param detailString the string to read the info from
     */
   public void readDiaryItem(String detailString){}

   /** Write the diary item
     * @return should return the OddWeeklyMeeting as a string for the diary file.
     */
   public String writeDiaryItem(){}
```

```
   /** Compares this OddWeeklyMeeting's details with OtherItem.
    * Would be sensible to compare that it is of same class first!
    * @param otherItem the item to compare with
    */
   public boolean sameItem(DiaryItem otherItem){}

   /** Check if this item occurs on a date
   * @param theDate the date to check
   * @return returns true if this DiaryItem occurs on theDate, else false
   */
   public boolean itemOccursOnDate(Date theDate){}

   /** Get the gap between meetings in weeks
     * @return returns the length of gap in weeks between meetings
     */
   public int getWeeksGap() {}

   /** sets the length of gap in weeks between meetings
    * @param weeksGap the desired gap in weeks
   */
   public void setWeeksGap(int weeksGap) {}
}

/** Monthly meeting class
 */
public class MonthlyMeeting extends RepeatedMeeting {
   public int [] whichWeeks; /* Array of ints saying which weeks of month have
meetings */

   /** This reads the info for a MonthlyMeeting from a given string and fills it in
    * @param detailString the detail string to read from
    */
   public void readDiaryItem(String detailString){}

   /** Write this diary item
    * @return should return the MonthlyMeeting as a string for the diary file.
    */
   public String writeDiaryItem(){}

   /** Compares this MonthlyMeeting's details with OtherItem.
    * Would be sensible to compare that it is of same class first!
    * @param otherItem the item to compare with
    * @return returns true if they are the same item, otherwise false
   */
   public boolean sameItem(DiaryItem otherItem){}

  /* Check if item occurs on date specified
   * @param theDate the date to compare with
   * @return returns true if this DiaryItem occurs on theDate, else false
   */
   public boolean itemOccursOnDate(Date theDate){}

   /** Get the weeks that have meetings
     * @return returns which weeks of the month have meetings
     */
   public int[] getWhichWeeks() {}

   /** sets which weeks of the month have meetings
    * @param weeks the weeks that have meetings
   */
   public void setWhichWeeks(int[] weeks) {}

}
```

4.4 Class interface for Heating (also DayBooking/HeatingEvent)

```
import uk.ac.aber.cs221.group01.global.utils.DiaryItem
import uk.ac.aber.cs221.group01.global.utils.DiaryAccess
import uk.ac.aber.cs221.group01.global.utils.CentralHeating;
import uk.ac.aber.cs221.group01.global.utils.DateAndTime;

/** Outline for Heating program
 * This design does not prescribe the user interface for the heating program
 * It assumes that there will be a user interface to invoke the different features,
 * and to show what is happening to the heating system.
 *
 * The application starts the user interface and reads a diary file
 * It provides facilities for the user interface to use
*/
public class Heating{
   protected DiaryAccess myDiary;
   protected DateAndTime timeNow;
   protected DateAndTime diaryCreated;
   protected DayBooking todaysBookings;
   protected Boolean diaryChanged;
   protected Boolean [] heatingOnInRoom;

   /** If there is a file named as args[0],
     * then use the constructor with that file name as a parameter, else give
     * a default file name
     * If there is a 14 character string as args[1] (dd/mm/yy hh:mm),
     * turn it into a time and date to start the simulation (timeNow), else default
     * @param args the command line arguments
    */
   public static void main(String args[]) {}

   /** Create myDiary
    *
    * Record creation date for diary in diaryCreated
    * If fileName does not exist, exit on error
    * Create the user interface - pass it heatingOnInRooms as a parameter
    * so it can see which rooms are lit
    * Call startUp(timeNow)
    * Call operateHeating in a thread
    * Call detectChange in a thread with diaryCreated as a parameter
    * @param fileName if not null, use it to read in the contents of myDiary
    * @param timeNow the time now
    */
    public static void Heating( String fileName, DateAndTime timeNow ) { }

    /** Set system time to now
      * Put system into coherent state - turn off all heating and boiler
      * Create todaysBookings, passing myDiary and now as the parameter
      * @param now the time now
      */
    public static void startUp ( DateAndTime now ) {}

    /** Check whether heat diary has changed
      * if it has, then re-read diary file into my Diary, and reset diaryCreated
      * if heat diary changed or it is a new day (within 5 minutes after midnight)
      * then recreat todaysBookings passing myDiary and Now as the parameter, and
      * turn all heating off
      * if it is past time for the first of the events in todaysBooking,
      * then do the event, log it and delete it from the list.
      * Repeat until only future events left
```

```
      * Wait five minutes and loop to start
      */
      public static void operateHeating ( ) {}

   /** Make sure all heating is turned off (and boiler), close log and exit
    */
    public static void closeDown ( ) {}
}


/** Outline for DayBooking class
  * This class handles the details of one day's booking.
  *
  * It creates an ordered list of all turn on and turn off events
  * for the heating for today, annotated
  * with the reason why the event is happening
  */
public class DayBooking {
   private HeatingEvent heatingList;

  /** For each booking in heatingList, check whether it applies to today */
   *     if it does, and the heating off event is not today before the time now */
   *           then create a heating on and heating off event for it */
   * @param heatingList the
   * @param now the date and time now
   */
   public static void DayBooking( DiaryItem heatingList, DateAndTime now){ }
}



/** Outline for HeatingEvent class */
 *
 * This class represents a single on or off event.
 * It says whether the heating is to be turned on or off, and when within the day
*/
public class HeatingEvent {
   private boolean turnOn;
   private String ReasonForChange;
   private HeatingEvent nextEvent;
   private boolean [] heatingOnInRoom;

   /** Set turnOn, ReasonForChange, heatingOnInRoom from parameters */
    * Set nextEvent to null
    * @param turningOn if  true turn on, if false if turn off
    * @param reasonWhy string describing reason for change
    * @param whichRooms array defining which rooms to change
   */
    public static void HeatingEvent( boolean turningOn, String reasonWhy,
                                     boolean[] whichRooms){}

   /** Check if the heating is on or off
    * @return returns true if Turn On, false if Turn Off
    */
   public boolean getTurnOn(){}

   /** Find out why the heating has been changed
    * @return returns the string saying why rooms being turned on or off
    */
   public String getReasonForChange(){}

  /** Returns an array of booleans matching the roomlist.
   * true means change that room's heating
   * @return returns the array of rooms to change
  */
   public boolean[] getHeatingOnInRoom(){}
```

```
    /** Get the next heating event
      * @return returns next HeatingEvent
     */
    public HeatingEvent getNextEvent() {}

    /** Sets nextEvent to myNextEvent
     * @param  myNextEvent the next heating event to use
     */
    public void setNextEvent(HeatingEvent myNextEvent) {}

}
```

4.5 Class interface for LogReading

```
import uk.ac.aber.cs221.group01.global.utils.LogItem
import uk.ac.aber.cs221.group01.global.utils.LogAccess

/** Outline for LogReading class
* This design does not prescribe the user interface for the LogReading program
* It assumes that there will be a user interface to show the logs to the user.
* The application starts the user interface and reads a log file
*/
public class LogReading {
   protected LogItem TheLog;

  /** If there is a file named as args[0],
    * then use the constructor with that file name as a parameter, else give
    * a default file name - could be adapted to read file selected from interface.
    * @param args the command line arguments
    */
   public static void main(String args[]) { }

   /** Create a LogAccess( fileName ), and put LogList in TheLog
     * Call DisplayLog(1)
     * @param fileName the name of the log file
     */
    public static void LogReading( String fileName ) { }

    /** Contents will depend on what interface is provided. This routine should
      * display the contents of the log file starting at the recordNum record.
      * You might provide an interface which only has a next record button and only
      * shows one record at a time. In that case you would update the count in the
      * interface and call displaylog with the new value.
      * You might have a scroll bar,
      * in which case you would recalculate the number when the scroll bar moved
      * @param recordNum the record number
      */
    public static void DisplayLog(int recordNum) {}

}
```

4.6 Class interface for LogAccess

```
package uk.ac.aber.cs221.group01.global.utils;

import uk.ac.aber.cs221.group01.global.utils.LogItem;
/** Outline for LogAccess class
  * This class handles the details of a log file.
  * It provides the ability to read or write a log file.
  * It holds a list of LogItem objects and can turn them into a file of objects,
  * or alternatively, read a Log file and obtain a list of LogItem objects.
  *
  * It provides routines to add, read, write the list of entries.
*/
public class LogAccess {
   private LogItem logList;
   private LogItem lastLogItem;

   /** Constructor - should initialise logList to Null */
   public static void LogAccess() { }

   /** This should open the file fileName and read the
     * contents into a set of LogItems in logList
     * @param fileName the name of the file to open
     */
   public static void readLog( String fileName){}

   /** This should open the file fileName and write the
     * LogItems in logList to the file.
     * @param fileName the of the file to write to
     */
    public static void writeLog(String fileName){ }

   /** This routine can be used to create a new LogItem and add it to LogList.
     * @param  changeReason the reason for the change
     * @param usedRooms the rooms used
     * @param turnedOn true if the system was turne on
     * @param changeTime the time the change took place
     */
   public static void addNewLogItem(String changeReason,
                                    String[] usedRooms,
                                    boolean turnedOn,
                                    Time changeTime){ }

}
```

4.7 Class interface for LogItem

```
/** Outline for LogItem class
  * This class handles the details of a single log file entry.
  */
public abstract class LogItem {
   private LogItem nextItem = NULL;  /* used to make a list of these things */

   private String ChangeReason;  /* Why the heating settings were changed */
   private String [] usedRooms; /* Names of the rooms that Changed */
   private Boolean turnedOn;  /* TRUE if the heating was turned on, else FALSE */
   private Time changeTime; /* When in the day the change happened */

   /* Get the next log item
    * @return returns item pointed to by nextItem
    */
   public LogItem getNextItem() { }

   /* Sets nextItem to theNext
    * @param theNext the next item to use
    */
   public void setNextItem(LogItem theNext) { }

   /* Get the change reason
     @return returns the string describing what the log entry is for
    */
   public String getChangeReason() {}

   /* Sets the string describing what the log entry is for */
    * @param theReason the reason for the change
    */
   public void setChangeReason(String theReason) {}

   /** Get the rooms used
    * @return returns the list of rooms changed
    */
   public String[] getUsedRooms() {}

   /** Sets the list of rooms changed
    * @param roomsUsed the new values for rooms used
    */
   public void setUsedRooms(String [] roomsUsed) {}

   /**
    * @return returns whether event was to turn on or off
    */
   public boolean getTurnedOn() {}

  /** Sets whether event was to turn on or off
    * @param ifOn the on/off value to use
    */
   public void setTurnedOn(boolean ifOn) {}

   /** Get the change time
     * @return returns the time that the change happened
     */
   public Time getChangeTime() {}

   /** Sets the time that the change happened at
    * @param theTime the time to use for the change
    */
   public setChangeTime(Time theTime) {}

}
```

5. DETAILED DESIGN

Content required.

5.1.1 The DIARY program

?????

5.1.2 The HEATING program

??????

5.1.3 The LOG READING Program

??????

6. REFERENCES

[1] SE.QA.01 Software engineering group projects - quality assurance plan.

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	17/2/01	N/A - original version	CJP
1.01	N/A	05/11/01	Right header, better scope	CJP
1.02	N/A	01/10/13	Changed comments to javadoc, fixed a few syntax errors.	BPT