Version: 12/09/2023 19:54:01

CSC1016S Assignment 7

UML Modelling and Classes

Assignment Instructions

This assignment concerns the development of UML models, and the construction of classes in Java using object composition i.e. write class declarations that define types of object that contain and manipulate other objects.

This assignment is based on the following scenario:

Scenario

The Fynbos Flower company is flexible on the hours that employees work, and as a consequence, timekeeping is an important administrative practice. The company records the dates and times of shifts (for want of a better word). Employees sign-in when they arrive and sign-out when they leave. The start of a shift and the end of a shift always fall within the same week.

This information supports a range of enquiries

- whether an employee is present,
- whether an employee was present on a given day,
- the details of the shift worked on a given day,
- the total hours worked during a given week.
- the shifts worked during a given week.

Though, currently, none of the employees bear the same name, the company assigns each a unique ID number that may be used to disambiguate.

The concept of a "week" bears elaboration. A business week starts on a Monday and ends on a Sunday. The weeks are numbered. The first week of the year is the one that contains the first Thursday of the year.

In the appendix you will find a set of specifications for an Employee class and classes that it depends on: Shift, Week, CalendarTime, Date, Time, Duration and TimeUnit.

Exercise one concerns constructing a UML class diagram for the classes, showing attributes, constructors, methods, and associations.

On the Amathuba assignment page you will find a ZIP file containing Java code for Week, CalendarTime, Date, Time, Duration and TimeUnit.

Exercise two concerns implementing the Shift class.

Exercise three concerns implementing the Employee class.

Exercise One [20 marks]

Drawing on the described scenario, your task is to study the class specifications in the appendix, and the supplied code, and develop an equivalent UML class diagram.

- Your diagram should depict all the classes.
- For each class you should give public attributes and operations, including those that are static.
- Your diagram should show relationships. Specifically, instances of generalisation, association, and aggregation.
- Associations and aggregations should be annotated with multiplicity.

NB: In the event that the specification of the class given in the appendix is not sufficient, you may refer to the java source code provided on Amathuba for additional information for your UML class diagrams, if it serves to clarify your answer.

Exercise Two [30 marks]

Implement the Shift class as specified in the appendix. To evaluate your work, you should use the interactive features of JGrasp and/or construct test code.

The following code fragment demonstrates class behaviour:

```
//
CalendarTime start = new CalendarTime("1/9/2019%22:00");
CalendarTime finish = new CalendarTime("2/9/2019%06:00");
Shift shift = new Shift(start, finish);
System.out.println(shift);

System.out.println(shift.start());
System.out.println(shift.finish());

System.out.println(shift.includesDate(new Date("31/8/2019")));
System.out.println(shift.includesDate(new Date("1/9/2019")));

System.out.println(shift.inWeek(new Week("35/2019")));
System.out.println(shift.inWeek(new Week("36/2019")));
System.out.println(shift.inWeek(new Week("37/2019")));

System.out.println(Duration.format(shift.length(), "minute"));
//
```

The output will be:

```
1/9/2019%22:00:00 - 2/9/2019%06:00:00
1/9/2019%22:00:00
2/9/2019%06:00:00
false
true
true
true
false
8 hours
```

Exercise Three [40 marks]

Implement the Employee class as specified in the appendix. To evaluate your work, you should use the interactive features of JGrasp and/or construct test code.

The following (extensive) code fragment demonstrates class behaviour:

```
//
Employee employee = new Employee("Sivuyile Ngesi", "01010125");
System.out.println(employee.name());
System.out.println(employee.UID());
System.out.println(employee.present());
//
System.out.println();
employee.signIn(new Date(1, 9, 2019), new Time(6,00));
System.out.println(employee.present());
employee.signOut(new Date(1, 9, 2019), new Time(18,00));
System.out.println(employee.present());
//
System.out.println();
employee.signIn(new Date(2, 9, 2019), new Time(16, 30));
employee.signOut(new Date(3, 9, 2019), new Time(2, 30));
//
System.out.println();
employee.signIn(new Date(3, 9, 2019), new Time(18,00));
employee.signOut(new Date(4, 9, 2019), new Time(4,00));
//
System.out.println();
System.out.println(employee.worked(new Date(31, 8, 2019)));
System.out.println(employee.worked(new Date(1, 8, 2019)));
//
System.out.println();
System.out.println(employee.worked(new Week(34, 2019)));
System.out.println(employee.worked(new Week(35, 2019)));
System.out.println(employee.worked(new Week(36, 2019)));
//
System.out.println();
List<Shift> shifts = employee.get(new Date(1, 9, 2019));
for(Shift shift : shifts) { System.out.println(shift); }
System.out.println();
shifts = employee.get(new Date(2, 9, 2019));
for(Shift shift : shifts) { System.out.println(shift); }
System.out.println();
shifts = employee.get(new Date(3, 9, 2019));
for(Shift shift : shifts) { System.out.println(shift); }
//
System.out.println();
shifts = employee.get(new Week(35, 2019));
for(Shift shift : shifts) { System.out.println(shift); }
//
System.out.println();
System.out.println(Duration.format(employee.hours(new Week(35,
2019)), "minute"));
```

The code produces the following output:

```
Sivuyile Ngesi
01010125
false

true
false

false
false

false
true
true

1/9/2019%06:00:00 - 1/9/2019%18:00:00

2/9/2019%16:30:00 - 3/9/2019%02:30:00

2/9/2019%16:30:00 - 3/9/2019%02:30:00
3/9/2019%18:00:00 - 4/9/2019%04:00:00

1/9/2019%06:00:00 - 1/9/2019%18:00:00

12 hours
```

NOTE: Each employee has a collection of shifts. You will need some type of collection object. An ArrayList is recommended.

Marking and Submission

Submit your UML class diagram and `Shift.java` and `Employee.java` classes all contained within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

yourstudentnumber.zip

Appendices

Class Employee

An object of this class represents an Employee from the perspective of time keeping. It records the employee name and ID, and logs sign-ins and sign-outs.

Constructors

public Employee(String name, String uid)
// Create an Employee representing the employee with given name and UID.

Methods

public String name()
// Obtain this employee's name.

public String UID()
// Obtain this Employee's ID.

public void signIn(Date d, Time t)
// Record that this employee has begun a shift on the given date and at the given time.

public void signOut(Date d, Time t)
// Record that this employee has completed a shift on the given date and at the given time.

public boolean present()
// Determine whether this employee is present i.e. has signed-in and not yet signed-out.

public boolean worked(Date d)
// Determine whether this employee worked a shift that at least partly occurred on the given date.

public boolean worked(Week w)
// Determine whether this employee worked at least one shift during the given week.

public List<Shift> get(Date d)
// Obtain the shift(s) worked by this employee that at least partly occur on the given date.

public List<Shift> get(Week w)
// Obtain a list of the shifts worked by this employee during the given week.

public Duration hours(Week w)
// Returns the total time (hours and minutes) worked during the given week.

Class Shift

An object of this class represents a work shift. A shift begins on a given date at a given time and ends on a given date at a given time.

Constructors

public Shift(CalendarTime start, CalendarTime finish)
// Create a Shift object representing a shift worked between the given date times.

Methods

public CalendarTime start()
// Obtain the start date and time for this shift.

public CalendarTime finish()
// Obtain the end date and time for this shift.

public boolean inWeek(Week w)
// Determine whether this shift occurred within the given week.

public boolean includesDate(Date date)
// Determine whether this shift at least partly occurred on the given date.

public Duration length()
// Obtain the length of this shift.

public String toString()
// Obtain a string representation of this shift of the form "<date>%:<time>-<date>%:<time>".

Class Week

An object of this class represents a business week. A business week starts on a Monday and ends on a Sunday. The business weeks of a year are numbered. The first week of the year is the one that contains the first Thursday of the year.

Constructors

public Week(int n, int y)
// Create a Week object that represents business week number n in year y.

public Week(String string)
// Create a Week object from a string of the form "<number>/<year>" where "<number>" is up to two digits long, and "<year>" is a 4-digit number.

Methods

public boolean includes(Date d)
// Determine whether this business week includes the given date d.

public String toString()
// Obtain a String representation of this Week of the form "<number>/<year>".

Class CalendarTime

An object of this class represents a calendar time i.e. a date and time.

Instance variables

private Date date;
private Time time;

Constructors

public CalendarTime(Date d, Time t)
// Create a CalendarTime object that represents the given date and time.

Methods

public int compareTo(CalendarTime other)
// Returns a value of -1, 0, or 1 depending on whether this calendar time precedes, is equivalent
// to, or follows the given calendar time, other.

public Date date()
// Returns the date component.

public Time time()
// Returns the time component.

public Duration subtract(CalendarTime other)
// Subtract the given CalendarTime from this CalendarTime.

public String toString()
// Obtain a string representation of this CalendarTime in the form "<date>%<time>".

Class Date

A Date object represents Gregorian calendar date.

Constructors

public Date(int d, int m, int y)
// Create a Date representing day d, month m, and year y.

public Date(String string)
// Create a Date from a string of the form "[d]d/[m]m/yyyy".

Methods

public int compareTo(Date other)
// Compare this Date to other date, returning a -ve number if this precedes other, a zero if they
// are coincident, and a positive number if other precedes this.

public Duration subtract(Date other)
// Obtain the time from the other date to this date.

public String toString()
// Obtain a String representation of this date in the form [d]d/[m]m/yyyy.

Class Time

A Time object represents a twenty-four-hour clock reading composed of hours, minutes and seconds.

Constructors

public Time(int h, int m)

// Create a Time representing the h^{th} hour *and m^{th} minute of the day.*

public Time(String reading)

// Create a Time object from a string representation of a twenty-four-hour clock reading
// of the form 'hh:mm[:ss]' e.g. "03:25", "17:55:05".

Methods

public Duration subtract(Time other)

// Set the first, middle and last names of this Student object.

public String toString()

// Obtain a String representation of this Time object in the form "HH:MM:SS".

Class Duration

A Duration object represents a length of time (with millisecond accuracy).

Constructors

public Duration(String timeUnit, long quantity)

// Create a Duration object that represents the given quantity of the given time unit.
// Permissible time units are: "millisecond", "second", "minute", "hour", "day" ,"week".

Methods

public int compareTo(Duration other)

// Returns a negative, zero, or positive value, depending on whether this duration is smaller, equal
// to, or greater than the other duration.

public boolean equals(Object o)

// Determine whether object o is equivalent to this object i.e. if it is a Duration and of the same
// value as this Duration.

public static String format(final Duration duration, final String smallestUnit)

// Obtain a formatted string that expresses the given duration as a series of no-zero quantities of
// the given time units.
// For example: Given Duration d=new Duration("second", 88893);, the expression
// Duration(d, "second") returns the string "1 day 41 minutes 33 seconds", while
// Duration(d, "minute") returns the string "1 day 41 minutes".

Class TimeUnit

A TimeUnit is a (sub) type of duration that has a name, such as ‘hour’, ‘minute’, ‘week.

Fields (Constants)

```
public final static TimeUnit MILLISECOND;
public final static TimeUnit SECOND;
public final static TimeUnit MINUTE;
public final static TimeUnit HOUR;
public final static TimeUnit DAY;
public final static TimeUnit WEEK;
```

Methods

```
public String getName()
        // Obtain this time unit’s name..

public static TimeUnit[] values()
        // Obtain an array of all the defined time units.

public static TimeUnit parse(String string)
        // Obtain the TimeUnit with the given name. Returns null if the string does not match any of
        // the defined units.
```

END