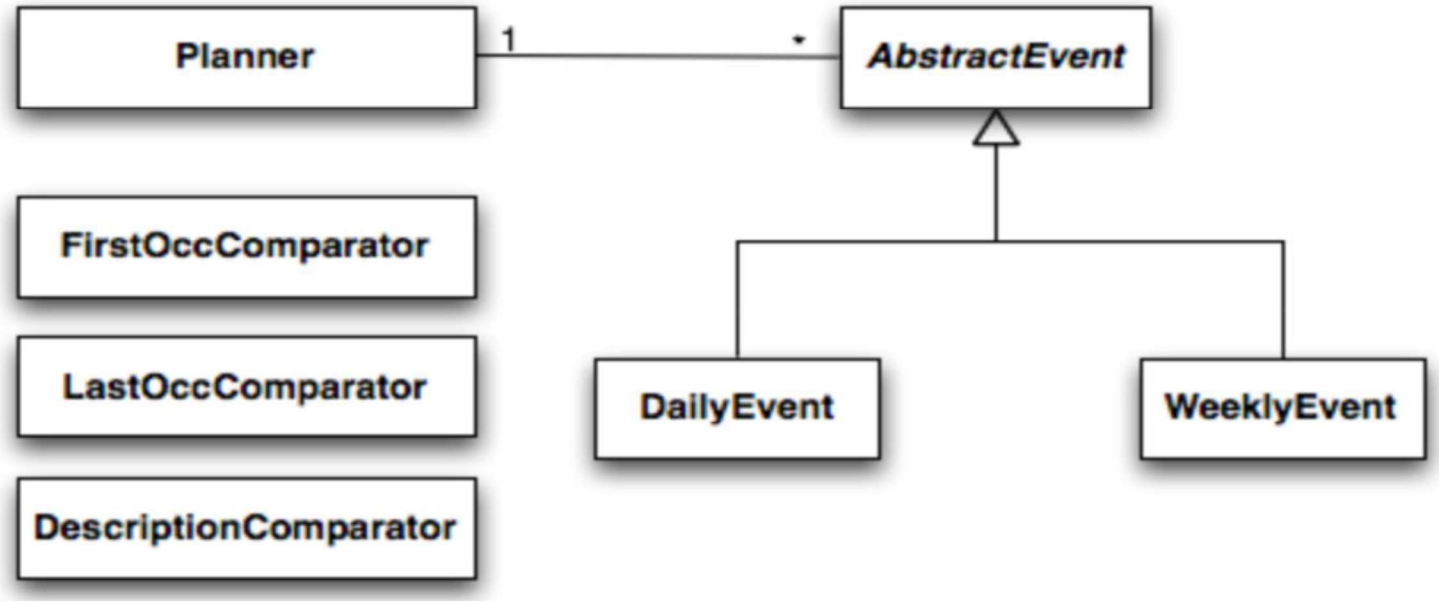
The Registry

Objectives

- Develop a good understanding of inheritance and polymorphism.
- Further understanding of the concept of abstract class and pure virtual function.
- Creating a simple data structure.

Introduction

As a member of the software development team for the company **Research In Progress**, you have been asked to implement classes to represent calendar events for their **Planner** mobile application. Specifically, there is a hierarchy of classes to represent events. At the top of the hierarchy there is a class called **AbstractEvent** to represent the characteristics that are common to all events. There are two specific kinds of events: **DailyEvent** and **WeeklyEvent**. All the events have a method **nextOccurrence** that returns the date of the next recurrence. However, the specific implementation of the method **nextOccurrence** depends on the kind of events. Finally, you must implement a simple data structure, here called **Planner**, to store any kind of events.



(1) Inheritance

1.1 AbstractEvent

The abstract class **AbstractEvent** implements the characteristics that are common to all the events.

- All the events have a description (of type **string**);
- All the events have a start and end time (both of type **time_t** included in the header file **<ctime>**) for the first recurrence of the event;
- All the events have setter and getter methods for their attributes;
- All the events have a method **bool hasMoreOccurrences( )**, however the implementation depends on the kind of event;
- All the events have a method **time_t nextOccurrence( )**, however the implementation depends on the kind of event;
- All the events have a method **void init( )**, however the implementation depends on the kind of event.
- All the events have a method **time_t lastOccurrence( )**, however the implementation depends on the kind of event. This method returns the date (start time) of the last occurrence of the event;
- All the events have a method **bool hasRecurrenceOn(const time_t &)**, however the implementation depends on the kind of event. This method returns **true** if the event has a recurrence on the specified date and **false** otherwise;
- All the events have a method **string toString( )**, however the implementation depends on the kind of event. This method returns a string representation of the event. An example of the expected format is given below.
- The methods do not change the state of the object; must be defined as **const**.

1.2 DailyEvent

DailyEvent is a concrete subclass of **AbstractEvent**.

- Daily events have a number of recurrences, which represents the number of consecutive days the event is repeated. There are also getter and setter methods for this attribute;
- Each call to the method **nextOccurrence( )** returns the date (start time) of the next recurrence of the event. If this event has 3 recurrences, the first call to the method **nextOccurrence( )** will return the start date of the event, the second call returns the start date plus one day, and finally, the third call will return the start date plus two days. The behavior of the method is not defined if the number of calls exceeds the number of recurrences;
- The method **hasMoreOccurrences( )** returns **true** if a call to the method **nextOccurrence( )** would return a valid answer, and **false** otherwise. The method does not change the state of the object;
- The method **init( )** re-initializes the state of the object so that a call to the method **nextOccurrence()** returns the date of the first occurrence of this event;
- This class must provide a valid implementation for the methods **lastOccurrence**, **hasRecurrenceOn**, and **toString**.

1.3 WeeklyEvent

WeeklyEvent is a concrete subclass of **AbstractEvent**.

- Weekly events have recurrences every week until a limit date is reached;
- The limit date (of type **time_t**) is an attribute of weekly events. There are setter and getter methods for this attribute;
- Each call to the method **nextOccurrence( )** returns the date of the next occurrence of the event. The first call to the method **nextOccurrence( )** will return the start date of the event, the second call returns the start date plus 7 days, the third call will return the start date plus 14 days, etc. The behavior of the method is not defined if the last recurrence exceeded the limit;
- The method **hasMoreOccurrences()** returns **true** if a call to the method **nextOccurrence()** would return a date before the limit and **false** otherwise;
- The method **init()** re-initializes the state of the object so that a call to the method **nextOccurrence()** returns the date of the first occurrence of this event;
- This class must provide a valid implementation for the methods **lastOccurrence**, **hasRecurrenceOn**, and **toString**.

---

(2) Comparator

The method for comparing two objects is written outside of the class of the objects to be sorted. Several methods can be written for comparing the objects according to different criteria. Specifically, write three classes, **DescriptionComparator**, **FirstOccComparator**, and **LastOccComparator** that are subclasses of the class **Comparator**.

- **Comparator** is an abstract class that contains only the pure virtual function **bool compare(AbstractEvent& e1, AbstractEvent& e2).** The implementation depends on the kind of event. The function should return **true** if the first argument is considered "**less than**" the second argument, and **false** otherwise. This class will act as the interface Comparator in the java language.
- **FirstOccComparator** is a subclass of **Comparator**. It must provide a valid implementation of **compare** method which compares the start time of two events.
- **LastOccComparator** is a subclass of **Comparator**. It must provide a valid implementation of compare method which compares the date of the last occurrence of two events.
- **DescriptionComparator** is a subclass of **Comparator**. It must provide a valid implementation of **compare** method which compares the description of two events.

---

(3) Planner

Create a class called **Planner**. A Planner is a data structure for storing events. The maximum capacity of the planner (physical size) is specified by the formal parameter of the constructor of the class. You can use the build-in **vector** class.

Instance methods

- **int size( )**; returns the number of events that are currently stored in this **Planner** (logical size);
- **bool addEvent(AbstractEvent * event)**; adds an event to the last position of this **Planner**. It returns **true** if the insertion was a success, and **false** if this **Planner** was already full;

- **AbstractEvent* eventAt(int pos)**; returns the event at the specified position in this **Planner**. This operation must not change this state of this **Planner**. The first event of the **Planner** is at position 0;
- **AbstractEvent* remove(int pos)**; removes the event at the specified position of this **Planner**. Shifts any subsequent items to the left (start of the array). Returns the event that was removed from this **Planner**;
- **void display(time_t date)**; prints all the events that have a recurrence on the specified date;
- **void sort(const Comparator & comp);** passes the list of events and the comparator object function to the method **sort** from the header file **<algorithm>.**
- **string toString( );** returns a string representation for the list of events. An example of the expected format is given below.
- **~Planner( );** to deallocate the dynamic allocated memory if there.

---

(4) Utilis

Create a class called **Utilis**. A Utilis is a utility class that provides two public static functions:

- **static string timeToString(const time_t& time)**
  This function converts **time_t** object into a string of the expected format as given below.
  e.g. `Tue May 23 21:10:00 2023`
- **static time_t createTime(int year, int month, int day, int hour = 0, int minute = 0)**
  This function creates **time_t** object holding specific data and time.

Part of points are awarded for a professional evaluation of your submissions.

Here is a small test program. Complete the missing parts in the main to get the corresponding output as given below. Your Test program and output must follow the sample given below.

Other suggestions:

- ***You can use templates where appropriate.***
- ***You can use the class array or suitable container where appropriate.***

```
int main() {


    Planner p(20);
    time_t input, start, end, limit;

    limit = Utilis::createTime(2023, 5, 25);

    start = Utilis::createTime(2023, 1, 3, 21, 10);
    end = Utilis::createTime(2023, 1, 3, 22, 30);
    p.addEvent(new WeeklyEvent("Shopping Time", start, end, limit));

    start = Utilis::createTime(2023, 3, 23, 18, 00);
    end = Utilis::createTime(2023, 3, 23, 19, 0);
    p.addEvent(new DailyEvent("Breakfast time", start, end, 10));

    start = Utilis::createTime(2023, 2, 5, 15, 30);
    end = Utilis::createTime(2023, 2, 5, 16, 45);
    p.addEvent(new WeeklyEvent("OOD SUN Lecture", start, end, limit));

    start = Utilis::createTime(2023, 1, 11, 10, 0);
```

```
    end = Utilis::createTime(2023, 1, 11, 11, 30);
    p.addEvent(new WeeklyEvent("CSCE 1521 Section B", start, end, limit));

    start = Utilis::createTime(2023, 2, 1, 15, 30);
    end = Utilis::createTime(2023, 2, 1, 16, 45);
    p.addEvent(new WeeklyEvent("OOD WED Lecture", start, end, limit));

    start = Utilis::createTime(2023, 1, 10, 11, 30);
    end = Utilis::createTime(2023, 1, 10, 13, 0);
    p.addEvent(new WeeklyEvent("CSCE 1521 Section A", start, end, limit));

    start = Utilis::createTime(2023, 1, 11, 14, 30);
    end = Utilis::createTime(2023, 1, 11, 17, 30);
    p.addEvent(new WeeklyEvent("CSCE 4321 Section A", start, end, limit));

    start = Utilis::createTime(2023, 2, 14, 14, 30);
    end = Utilis::createTime(2023, 2, 14, 17, 0);
    p.addEvent(new WeeklyEvent("TA Office Hours", start, end, limit));

    start = Utilis::createTime(2023, 4, 11, 9, 30);
    end = Utilis::createTime(2023, 4, 11, 17, 30);
    p.addEvent(new DailyEvent("Examination", start, end, 18));
    cout << "################################################################" << endl;
    //TODO: Display here the Recurrences of: OOD SUN Lecture
    /* Write the required code here
        To get the corresponding output as given below */
    cout << "################################################################" << endl;
    cout << "################################################################" << endl;
    //TODO: Display here the Recurrences of: Breakfast time
    /* Write the required code here
       To get the corresponding output as given below */
    cout << "################################################################" << endl;
    cout << "################################################################" << endl;
    cout << "The start time of last occurrence of all the events:" << endl;
    /* Write the required code here
       To get the corresponding output as given below */
    cout << "################################################################" << endl;
    cout << "################################################################" << endl;
    cout << "Events that have a recurrence on March 28, 2023:" << endl;
    p.display(Utilis::createTime(2023, 3, 28));
    cout << "################################################################" << endl;
    p.addEvent(p.remove(0));
    cout << "################################################################" << endl;
    cout << "Content of the Planner:" << endl;
    cout << p.toString() << endl;
    cout << "################################################################" << endl;
    cout << "Sorting the content of the Planner by first occurrence..." << endl;
    p.sort(FirstOccComparator());
    cout << "Content of the Planner:" << endl;
    cout << p.toString() << endl;
    cout << "################################################################" << endl;
    cout << "Sorting the content of the Planner by last occurrence..." << endl;
    p.sort(LastOccComparator());
    cout << "Content of the Planner:" << endl;
    cout << p.toString() << endl;
    cout << "################################################################" << endl;
    cout << "Sorting the content of the Planner by description..." << endl;
    p.sort(DescriptionComparator());
    cout << "Content of the Planner:" << endl;
    cout << p.toString() << endl;

    cout << "################################################################" << endl;
    cout << "################################################################" << endl;
}
```

Here is an example to illustrate the expected output.

```
############################################################
Recurrences of: OOD SUN Lecture
Sun Feb 12 15:30:00 2023
Sun Feb 19 15:30:00 2023
Sun Feb 26 15:30:00 2023
Sun Mar  5 15:30:00 2023
Sun Mar 12 15:30:00 2023
Sun Mar 19 15:30:00 2023
Sun Mar 26 15:30:00 2023
Sun Apr  2 15:30:00 2023
Sun Apr  9 15:30:00 2023
Sun Apr 16 15:30:00 2023
Sun Apr 23 15:30:00 2023
Sun Apr 30 15:30:00 2023
Sun May  7 15:30:00 2023
Sun May 14 15:30:00 2023
Sun May 21 15:30:00 2023
############################################################
############################################################
Recurrences of: Breakfast time
Thu Mar 23 18:00:00 2023
Fri Mar 24 18:00:00 2023
Sat Mar 25 18:00:00 2023
Sun Mar 26 18:00:00 2023
Mon Mar 27 18:00:00 2023
Tue Mar 28 18:00:00 2023
Wed Mar 29 18:00:00 2023
Thu Mar 30 18:00:00 2023
Fri Mar 31 18:00:00 2023
Sat Apr  1 18:00:00 2023
############################################################
############################################################
The start time of last occurrence of all the events:
Event: Shopping Time            Last occurrence: Tue May 23 21:10:00 2023
Event: Breakfast time           Last occurrence: Sat Apr  1 18:00:00 2023
Event: OOD SUN Lecture          Last occurrence: Sun May 21 15:30:00 2023
Event: CSCE 1521 Section B      Last occurrence: Wed May 24 10:00:00 2023
Event: OOD WED Lecture          Last occurrence: Wed May 24 15:30:00 2023
Event: CSCE 1521 Section A      Last occurrence: Tue May 23 11:30:00 2023
Event: CSCE 4321 Section A      Last occurrence: Wed May 24 14:30:00 2023
Event: TA Office Hours          Last occurrence: Tue May 23 14:30:00 2023
Event: Examination              Last occurrence: Fri Apr 28 09:30:00 2023
```

```
############################################################

############################################################

Events that have a recurrence on March 28, 2023:

WeeklyEvent: { description= "Shopping Time", startTime= "Tue Jan  3 21:10:00 2023", endTime= "Tue Jan  3 22:30:00 2023", limit= Thu
May 25 00:00:00 2023 }

DailyEvent: { description= "Breakfast time", startTime= "Thu Mar 23 18:00:00 2023", endTime= "Thu Mar 23 19:00:00 2023",
numberOfOccurrences= 10 }

WeeklyEvent: { description= "CSCE 1521 Section A", startTime= "Tue Jan 10 11:30:00 2023", endTime= "Tue Jan 10 13:00:00 2023",
limit= Thu May 25 00:00:00 2023 }

WeeklyEvent: { description= "TA Office Hours", startTime= "Tue Feb 14 14:30:00 2023", endTime= "Tue Feb 14 17:00:00 2023", limit=
Thu May 25 00:00:00 2023 }

############################################################

############################################################

Content of the Planner:

Planner: {

DailyEvent: { description= "Breakfast time", startTime= "Thu Mar 23 18:00:00 2023", endTime= "Thu Mar 23 19:00:00 2023",
numberOfOccurrences= 10 },

WeeklyEvent: { description= "OOD SUN Lecture", startTime= "Sun Feb  5 15:30:00 2023", endTime= "Sun Feb  5 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section B", startTime= "Wed Jan 11 10:00:00 2023", endTime= "Wed Jan 11 11:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "OOD WED Lecture", startTime= "Wed Feb  1 15:30:00 2023", endTime= "Wed Feb  1 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section A", startTime= "Tue Jan 10 11:30:00 2023", endTime= "Tue Jan 10 13:00:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 4321 Section A", startTime= "Wed Jan 11 14:30:00 2023", endTime= "Wed Jan 11 17:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "TA Office Hours", startTime= "Tue Feb 14 14:30:00 2023", endTime= "Tue Feb 14 17:00:00 2023", limit=
Thu May 25 00:00:00 2023 },

DailyEvent: { description= "Examination", startTime= "Tue Apr 11 09:30:00 2023", endTime= "Tue Apr 11 17:30:00 2023",
numberOfOccurrences= 18 },

WeeklyEvent: { description= "Shopping Time", startTime= "Tue Jan  3 21:10:00 2023", endTime= "Tue Jan  3 22:30:00 2023", limit= Thu
May 25 00:00:00 2023 }

}

############################################################

Sorting the content of the Planner by first occurrence...

Content of the Planner:

Planner: {

WeeklyEvent: { description= "Shopping Time", startTime= "Tue Jan  3 21:10:00 2023", endTime= "Tue Jan  3 22:30:00 2023", limit= Thu
May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section A", startTime= "Tue Jan 10 11:30:00 2023", endTime= "Tue Jan 10 13:00:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section B", startTime= "Wed Jan 11 10:00:00 2023", endTime= "Wed Jan 11 11:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 4321 Section A", startTime= "Wed Jan 11 14:30:00 2023", endTime= "Wed Jan 11 17:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "OOD WED Lecture", startTime= "Wed Feb  1 15:30:00 2023", endTime= "Wed Feb  1 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "OOD SUN Lecture", startTime= "Sun Feb  5 15:30:00 2023", endTime= "Sun Feb  5 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "TA Office Hours", startTime= "Tue Feb 14 14:30:00 2023", endTime= "Tue Feb 14 17:00:00 2023", limit=
Thu May 25 00:00:00 2023 },
```

```
DailyEvent: { description= "Breakfast time", startTime= "Thu Mar 23 18:00:00 2023", endTime= "Thu Mar 23 19:00:00 2023",
numberOfOccurrences= 10 },

DailyEvent: { description= "Examination", startTime= "Tue Apr 11 09:30:00 2023", endTime= "Tue Apr 11 17:30:00 2023",
numberOfOccurrences= 18 }

}

############################################################

Sorting the content of the Planner by last occurrence...

Content of the Planner:

Planner: {

DailyEvent: { description= "Breakfast time", startTime= "Thu Mar 23 18:00:00 2023", endTime= "Thu Mar 23 19:00:00 2023",
numberOfOccurrences= 10 },

DailyEvent: { description= "Examination", startTime= "Tue Apr 11 09:30:00 2023", endTime= "Tue Apr 11 17:30:00 2023",
numberOfOccurrences= 18 },

WeeklyEvent: { description= "OOD SUN Lecture", startTime= "Sun Feb  5 15:30:00 2023", endTime= "Sun Feb  5 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section A", startTime= "Tue Jan 10 11:30:00 2023", endTime= "Tue Jan 10 13:00:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "TA Office Hours", startTime= "Tue Feb 14 14:30:00 2023", endTime= "Tue Feb 14 17:00:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "Shopping Time", startTime= "Tue Jan  3 21:10:00 2023", endTime= "Tue Jan  3 22:30:00 2023", limit= Thu
May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section B", startTime= "Wed Jan 11 10:00:00 2023", endTime= "Wed Jan 11 11:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 4321 Section A", startTime= "Wed Jan 11 14:30:00 2023", endTime= "Wed Jan 11 17:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "OOD WED Lecture", startTime= "Wed Feb  1 15:30:00 2023", endTime= "Wed Feb  1 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 }

}

############################################################

Sorting the content of the Planner by description...

Content of the Planner:

Planner: {

DailyEvent: { description= "Breakfast time", startTime= "Thu Mar 23 18:00:00 2023", endTime= "Thu Mar 23 19:00:00 2023",
numberOfOccurrences= 10 },

WeeklyEvent: { description= "CSCE 1521 Section A", startTime= "Tue Jan 10 11:30:00 2023", endTime= "Tue Jan 10 13:00:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 1521 Section B", startTime= "Wed Jan 11 10:00:00 2023", endTime= "Wed Jan 11 11:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "CSCE 4321 Section A", startTime= "Wed Jan 11 14:30:00 2023", endTime= "Wed Jan 11 17:30:00 2023",
limit= Thu May 25 00:00:00 2023 },

DailyEvent: { description= "Examination", startTime= "Tue Apr 11 09:30:00 2023", endTime= "Tue Apr 11 17:30:00 2023",
numberOfOccurrences= 18 },

WeeklyEvent: { description= "OOD SUN Lecture", startTime= "Sun Feb  5 15:30:00 2023", endTime= "Sun Feb  5 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "OOD WED Lecture", startTime= "Wed Feb  1 15:30:00 2023", endTime= "Wed Feb  1 16:45:00 2023", limit=
Thu May 25 00:00:00 2023 },

WeeklyEvent: { description= "Shopping Time", startTime= "Tue Jan  3 21:10:00 2023", endTime= "Tue Jan  3 22:30:00 2023", limit= Thu
May 25 00:00:00 2023 },

WeeklyEvent: { description= "TA Office Hours", startTime= "Tue Feb 14 14:30:00 2023", endTime= "Tue Feb 14 17:00:00 2023", limit=
Thu May 25 00:00:00 2023 }

}

############################################################
```