

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
6.170 LABORATORY IN SOFTWARE ENGINEERING
SPRING 2000

Quiz 2 part B
April 5, 2000

This is an essay test.

There are four questions: one question worth 5 points, and three worth 10 points each.

Write all answers in the blue books provided to you.

Clarity of expression and use of relevant course concepts in your argument is much more important than quantity of writing.

You may write your answer in bullet point form. For example

- Trains are a better way to travel than planes:
 - Plenty of room to move around
 - No annoying security delays at the stations
 - Less risky
- Plenty of room to move around
 - bigger compartments
 - wider aisles and seat spacing

You should read the entire test before beginning to write, to allocate your time properly.

Background

You are on a two person team assigned to build a software system, called **Remind**.

The purpose of Remind is to help the user remember tasks and appointments. The user prepares a file of appointments, starts Remind and leaves it running. Remind prints a message to the screen when it is time to do a task or leave for an appointment.

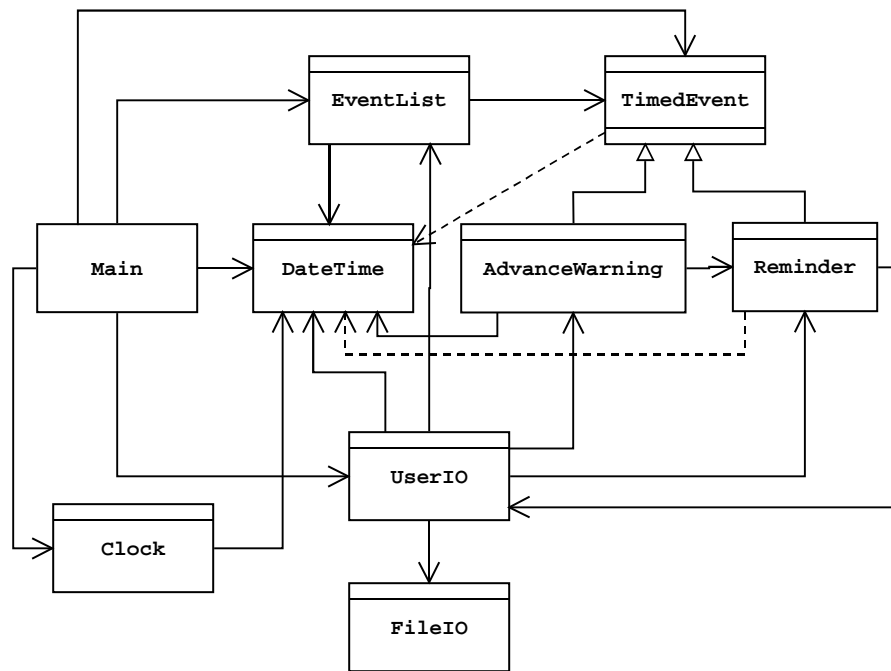
While Remind is running, it shows a dialog box that the user can use to add new appointments. When the user presses quit in the dialog box, Remind writes out the current list of appointments to the file (so it will be available when Remind runs again) and exits.

Each appointment has a description string (like **Pick up Fred at the airport**), a weekday (like **Sunday**), a date (like **4/23/2000**) and a time (like **10:00**). Additionally, each appointment has a list of advance warning times (like **30,15**). An appointment whose time is **10:00** with advance warning list **30,15** means that the user wants to be warned 30 minutes in advance (at 9:30) and 15 minutes in advance (at 9:45) in addition to seeing the final reminder at 10:00.

For regular appointments, Remind allows the user (or the file) to specify *any*. An appointment `6.170 team meeting Tuesday 4/any/2000 12:00` requests a reminder at noon on every Tuesday in April, 2000. The user or file may specify *any* in any of the parts of the weekday, date, or time.

Part A

Your partner does an initial design and draws the following MDD.



In this design, appointments are represented by **AdvanceWarning** and **Reminder** objects. These objects are stored in an **EventList** which keeps a list of **TimedEvent** objects in order, earliest first.

```

interface TimedEvent {
    DateTime when();           // answer the date & time of this event
    void display();           // display the reminder on the screen
}
  
```

Main alternates between calling **UserIO** and **Clock**. Each time **UserIO** is called, it checks for and processes user input. If requested by the user, **UserIO** will create an **AdvanceWarning** or **Reminder** object and insert it in the **EventList**. The first time **UserIO** is called, it calls **FileIO** multiple times and treats each appointment retrieved from the file as if it had been entered by the user. The last time **UserIO** is called (ie. when it detects that quit has been pressed) it reads all the appointments out of the event list and uses **FileIO** to write them back to the file.

When **Main** calls **Clock**, **Clock** returns the current time as a **DateTime** object. **Main** then looks at the first event in the **EventList**, and checks whether `event.when()` is earlier than the current time. If yes, **Main** calls a method `display()` of the **TimedEvent**, causing it to display itself using the **UserIO** object. **Main** then removes the **TimedEvent** from the **EventList** and loops.

AdvanceWarning stores a pointer to the **Reminder** object it is associated with, and a number which represents a time interval (in minutes) in advance that the user has requested an advance warning.

There are no static methods in the design other than **Main**.

Questions for Part A: (Each numbered question is independent)

1. (5 points)

This design has a flaw which can cause it to potentially use a large amount of space when it runs.

- Describe the flaw precisely.

(5 point answer) The design requires many reminder objects to be inserted in the eventlist when the user specifies a repeating reminder using *any*. A user request such as 11/1/*any*, which is allowed by the specification, would require an infinite number of objects to be created.

(3 point answer) The design requires the entire reminder file to be read into memory when the program starts. This could potentially be a large number of reminders. *(A quick calculation of the amount of memory likely to be used shows that this is not a significant problem on any modern PC, given the number of reminders one could expect in a reminder file).*

(1 or 2 point answer) The design requires the creation of multiple AdvanceWarning objects per reminder. The information about advance warnings could be represented more efficiently as fields in the Reminder object. *(The actual difference in space is in the noise given that there won't be millions of AdvanceWarning objects).*

2. (10 points)

A potential change to the specification is that the user might want to delete appointments. Implementing delete in this design would be somewhat inefficient.

- Describe a change to the design that improves delete.
- Explain why delete is faster in your modified design than in the design shown above.
- Show any changes to the MDD that go along with your modifications.

2 points for identifying the problem: there is no reference from a Reminder object to its AdvanceWarning objects, so having chosen some Reminder to delete, it is inefficient to locate all the AdvanceWarning objects that need to be deleted.

4 points for an efficient solution: add a reference from Reminder to AdvanceWarning. *(Other valid solutions were proposed; points were deducted if your solution reduced the efficiency of the main loop, which after all runs much more frequently than delete).*

2 points for drawing the MDD *(You could earn 1 or 2 points if your MDD matched your description even if the design modification itself was judged incorrect).*

2 points for overall clarity of explanation.

3. (10 points)

Your partner suggests a revised MDD in which **AdvanceWarning** has no dependence on **Reminder**. Instead, **AdvanceWarning** has a dependency on **UserIO**.

- Is this change desirable or undesirable?
- Explain why, using concepts from the lecture or textbook where relevant.

(You will need to figure out what else changes in order for the program to function correctly with the revised MDD.)

The change is undesirable. There are several reasons for this.

Major reasons: *(Must answer two of the major reasons to get 10 points)*

(8 points) The current design allows AdvanceWarning to reuse code in Reminder for storing the time and message, displaying the message, etc. With the change, this code will have to be duplicated.

(7 points) The change results in more dependencies on UserIO. Although there are the same number of strong dependencies in the two designs, the AdvanceWarning and Reminder modules serve closely related purposes so the strong dependency between them is better than an additional strong dependency on UserIO. *(You must explain the reason that the dependency on UserIO is worse than the dependency on Reminder in order to receive full points).*

(6 points) In the event that the functionality is extended to allow the user to modify an existing reminder, modifications will be more difficult in the changed design.

Minor reasons: *(You earned these points only if you didn't state any of the major reasons)*

(4 points) You could argue that the change is actually desirable, because in the changed design an AdvanceWarning could display a message that is different from its associated Reminder. In the existing design the same message is displayed at the advance warning times and at the final reminder.

(3 points) The change is undesirable because more memory will be consumed by storing multiple copies of the message and other information that is stored only in the Reminder in the current design.

Part B

There are 21 days available from when you are assigned the task until the program must be delivered. Your partner suggests the following development plan.

Day	Person A	Person B
1-3	Develop detailed interfaces for all modules	
4-7	Implement FileIO	Implement DateTime, Reminder, & EventList
8-9	Integrate, write initial documentation	
10	Demo to user	
11-12	Revise module interfaces	
13-15	Implement Clock & Main	Implement AdvanceWarning & UserIO
16-17	Write & perform unit tests	
18-21	Integrate, perform integration tests, final documentation	

Question for part B

4. (10 points)

There are major flaws in this plan.

- Identify two of the major flaws in the plan.
You should identify flaws that create a significant risk of failure to complete the project on time (or to complete it in an acceptable manner). There are more than two; it does not matter which two you choose.
- For each flaw you have chosen:
 - In what way does that aspect of the plan create a significant risk that the project will fail?
 - How would you change the plan to fix the flaw?
 - Why will this change significantly reduce the risk of failure?

If you have a question about the plan that is not answered by the description given, make an assumption, state it clearly, and continue with your analysis.

5 points for the first flaw you identified, 5 points for the second.

Within each flaw, 2 points for identifying a major flaw (as opposed to a minor one) and explaining it clearly, 3 points for explaining the risk it creates of failure and the benefits of the change you propose.

Some of the major flaws: UserIO not implemented before the demo, testing done too late, documentation done too late, demo too late, no days scheduled to recover from schedule slips.