CS1112 Fall 2014 Project 6 Part B **due Thursday 12/4 at 11pm**

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

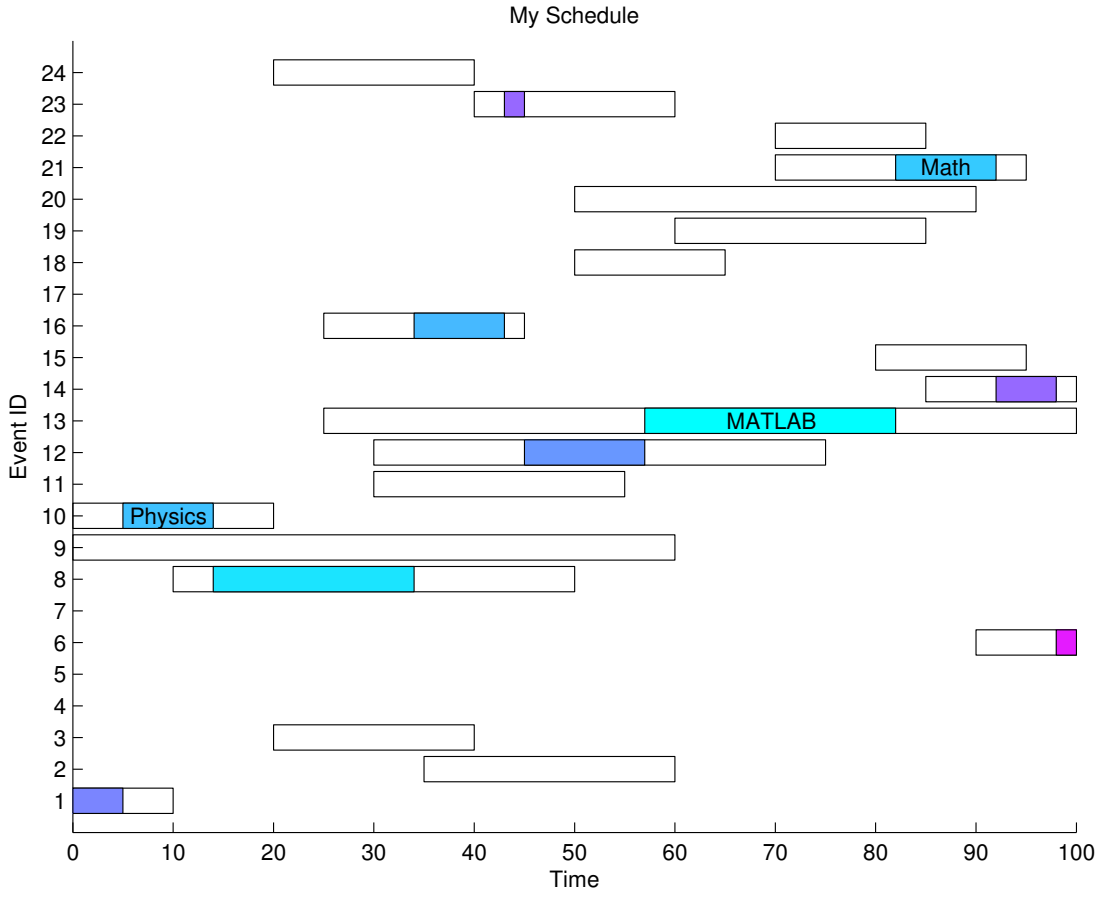
Objectives

Completing this project will solidify your understanding of structs and struct arrays (Part A), object-oriented programming (Part B), and recursion (Part C). In Part B you will also get practice on developing and testing code *incrementally*—one class (or even one method) at a time.

Part A (question 1) appears in a separate document.

2 Event Scheduling

We busy students and professors need to pack many events into a day. A convention venue wants to schedule events in order to maximize profits. You will write code for event scheduling, given the needs of the events and constraints on the scheduling time window. Below is a graphic of an example schedule. Every event has an integer ID while some additionally have names (e.g., course names). Time is represented using non-negative integer values. The colored boxes indicate the actual scheduled time of the events while the white boxes indicate, according to the event data, the time intervals during which the events could have been scheduled. The importance of the events is color coded from cyan—most important—to magenta—least important.

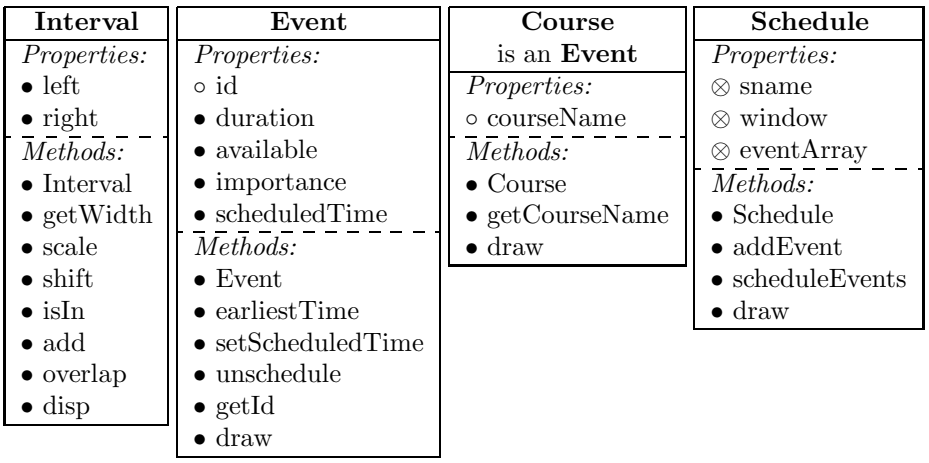


2.0 Object-Oriented Design

We provide the *design* of the classes that you will *implement* in this project. Read carefully about the reasoning behind the design below.

In the above description of the project, these *nouns* keep coming up: event, schedule, and time interval. These are the objects that we will use. An *Interval* has a left and right end point. An *Event* has an ID, a duration, and a time interval during which it can take place. There're special events that are named; we'll call such a named event a `Course` in this project. A *Schedule* contains a set of events that are set at specific times within a time window.

Our design involves four classes: `Interval`, `Event`, `Schedule`, and `Course` (subclass of `Event`). We then write a function `createSchedule` to read event data from a file and then instantiate objects of the different classes in order to create a feasible schedule. Don't be intimidated by the number of classes! Class `Interval` is given, and partial code is given in all the other classes. Below is a summary of the classes, showing what properties and methods are available in each class. Properties and methods with `private` access are shown using an open circle, ○; those with `private` *set* access are shown using a circle with an X, ⊗; those with `public` access are shown using a filled circle, ●. The skeleton code posted on the course website gives further details on each item. The rest of this document helps you develop the classes incrementally, one at a time.



The above summary diagrams will be handy when you later write code that involves interactions among the objects of the different classes.

So which class should you work on first? Answer: the most independent class, the one that doesn't depend on other classes. We will start with class `Interval`. As you complete the classes, do not change the names of the properties, methods and parameters. Do not change the property and method attributes (e.g., `private`). You should not need to add any extra methods in the classes.

2.1 Class `Interval`

Download the `Interval.m` file from the *Projects* page of the course website. (We used multiple versions of the class definition during lecture; please be sure to use the version released for Project 6.)

Read the class definition and then experiment with it! Assuming that all the skeleton files are in your Current Directory, type the following statements in the Command Window:

```
i1= Interval(3,9);  % Instantiate an Interval with endpoints 3 and 9
a= i1.left          % Should be 3.  The properties have the attribute "public" so
                    %  it is possible to access the property left directly.
o= i1.overlap( Interval(5,15) )
                    % o references an Interval with endpoints 5 and 9.
```

We started a file `testScript.m` to help you test your classes as you develop them. Read and run `testScript` now; you can see that it contains the same kind of code as suggested above for "exercising" the `Interval`

class. You will later add code to `testScript.m` in order to test the methods that you implement. You will submit `testScript.m` as a record of your development process.

If there is anything that you don't understand in this class, ask and figure it out before moving on! You want to make sure that you understand everything here before working on another class that depends on class `Interval`.

2.2 Class `Event`

Read the partially completed file `Event.m`. Notice that the property that has a non-numeric type is given an initial value so that its type is made clear. Read the given constructor and the associated comments carefully; note how `nargin` is used to check for the number of arguments passed.

Chaining up references Consider the following fragment:

```
e1= Event(3, 20, 10, .5, 4);
                  % An Event with id 4, importance .5, and duration 10.
                  % It's available for scheduling in the interval [3,20].
disp(e1.available.right)       % Should be 20
disp(e1.available.getWidth())  % Should be 17
disp(e1.id)       % Error: id is private
e1.setScheduledTime(5)
figure;  hold on
e1.draw()         % Should see colored box with left edge at x=5
hold off
```

Notice how two references are "chained together" using the dot notation:
`e1` references an `Event`.
An `Event` object has the field `available`, which references an `Interval`. Therefore
`e1.available` references an `Interval`.
An `Interval` has the property `right` and an instance method `getWidth`, therefore
`e1.available.right` is a scalar numeric value (the right end point) and
`e1.available.getWidth()` returns a scalar numeric value (the width).

Add the above statements to `testScript.m`, which exercises the constructor and several instance methods and confirms that property `id` is private. As you work on the individual instance methods, after implementing each one be sure to add code to `testScript.m` to call that method as a test. It is tempting to rush through coding without stopping to test, but that will burn more time in the end because you will have to deal with many confounding errors in your buggy atop buggy code when you finally get around to running your program. Make sure you understand everything in this class and solve any problems before moving on.

Comment out the test statements that cause errors, e.g., `disp(e1.id)` above, but do not remove such statements from `testScript`. `testScript` is a documentation of all the tests that you do.

2.3 Class `Schedule`

Read the partially completed file `Schedule.m`. The properties block is given; read it carefully. You need to implement the constructor and all the instance methods.

Constructor Be sure to check the number of arguments passed using `nargin`. Assign to the field `window` only if the number of arguments is at least two. Assign to the field `name` only if the number of arguments is three.

addEvent This is a short method; don't be alarmed.

scheduleEvents We use a heuristic[1] to schedule events. Implement this instance method *according to the heuristic given* in the comments. Depending on the actual event data and the scheduling window, not all events in `eventArray` may get scheduled. (In the figure on page 1 you see both the scheduled and unscheduled events.)

[1] A heuristic is a rule-of-thumb, or an experience-based method to solve a problem. A heuristic is often used instead of exhaustive search to avoid the computational cost associated with exhaustive search. Heuristics do not guarantee that the optimal solution will be found but generally the result is "good enough."

draw This method draws the schedule, an example of which is shown on page 1. Start a figure window using the following command to make it full-screen:

```
figure('units','normalized','outerposition',[0 0 1 1], 'name', 'Schedule')
hold on
```

At the end of the method, set the y-tick marks at integer (ID) values only and set the axis limits using the following commands:

```
set(gca, 'ytick', minId:maxId)
axis([xmin xmax minId-1 maxId+1])
```

where `minId` and `maxId` stores the lowest and highest ID values of the events in the event array, respectively, and `xmin` and `xmax` store the limits of the scheduling window. The schedule should be drawn only if the event array is not empty. Also use the command `hold off` at the end of the method.

In `testScript`, create several `Events` to add to a `Schedule` for testing:

```
e2= Event(0, 30, 8, .3, 1)
e3= Event(8, 25, 6, 0, 5)
s = Schedule(0, 40, 'Test Schedule')  % Instantiate a Schedule object.  s.eventArray is empty.
s.sname= 'New name'                   % ERROR: property sname has private set access
disp(s.sname)                         % Should see 'Test Schedule' since get access is public
s.addEvent(e2)                        % Add Event e2 to s.eventArray
s.addEvent(e3)
s.addEvent( Event( 10, 38, 5, 1, 2) )
disp(s)                               % s.eventArray should be a length 3 cell array of Events
s.eventArray{1}.getId()               % Should see 1
s.eventArray{1}.setScheduledTime(21)
figure;  hold on
s.eventArray{1}.draw()                % Should see colored box with left edge at x=21
hold off
```

Add more code to `testScript` to exercise every instance method defined in class `Schedule`! Do you understand every method call above? If not, ask for help! Make sure that all the methods work before moving on.

2.4 Function `createSchedule`

We take a break from class definitions now and write a function to read event data from a file and then schedule the events. Read the partially implemented function `createSchedule`. (Note: this is an independent function in the file `createSchedule.m`, not an instance method inside some class definition.)

The given code opens and closes the data file and instantiates a `Schedule` object. You need to insert code to read the data file, instantiate `Event` objects and add them to the event array in `Schedule s`, schedule the events (by calling an appropriate method), and draw the schedule graphic (by calling an appropriate method). For now, *assume that there are* `Event` *objects only*, not `Course` objects.

Syntax note: The comment block showed the expression `L(30:end)` where `L` is a vector. The `end` keyword, when used as an index, is the last index of the vector. So `L(30:end)` is the same as `L(30:length(L))`.

Test your function using the data file `eventdata1.txt`, which contains data for `Events` only (no `Course`). For example, try to schedule the events in `eventdata1.txt` in a time window of 0 to 100, like this:
`[a, ex] = createSchedule('eventdata1.txt', 0, 100)`

Is everything working? If not, debug and if necessary get help from the course staff before moving on.

2.5 Class `Course`

Class `Course` is a child class of `Event`. Read the given constructor and then implement the instance methods as specified.

Consider these and more examples for testing your code using the file `testScript` that you've been building up:

```
c1= Course(8, 25, 6, 0.5, 6, 'CS1000')
figure;  hold on
c1.draw()                % Should see white box with x range of 8 to 25
c1.setScheduledTime(9)
hold off
figure;  hold on
c1.draw()                % Should see colored box with left edge at x=9 and
                         %  the course name in the middle
hold off
s.addEvent(c1)
disp(s.eventArray)  % Should see that the last cell references a Course,
                    %  not an Event
```

Again, add code to `testScript` to test every instance method.

2.6 Function `createSchedule` (again)

You're at the final step! Now modify your code so that the function can handle `Course`s as well as `Event`s. Then use the data file `eventdata2.txt` to create a schedule! (You can modify the data file or create another file for more testing.)

Submit your files `testScript.m`, `Event.m`, `Schedule.m`, `Course.m`, and `createSchedule.m` on CMS.