CIS*2430 (Fall 2014) Assignment One

Instructor: F. Song

Due Time: October 15, 2014 by 11:55pm

In this course, you will be working on a project called "Day Planner", which will be built incrementally over the semester through three individual assignments. In addition to the implementation , you are also required to follow proper coding styles and establish good habits for testing and documentation.

General Description

A "Day Planner" program allows us to manage our daily activities: we can enter activities of different kinds, search for activities based on the keywords in their descriptions and/or during a specific time period and/or for a specific activity kind, and then display all activities that match a given search request.

Obviously, there can be many kinds of activities, depending on how we classify them. For this project, we assume that there are only three kinds of activities: "home" activities are anything we do at home; "school" activities are anything we do at school; and "other" activities are anything we do outside of home and school. For each activity, we should have a title (usually a short description that characterizes the activity), a starting time, an ending time, and an optional comment (usually a long decription that can offer more details). For other activities, we also need to know the locations since they can happen in many different places.

To search for relevant activities, the user can specify a search request using any or all of the three components: (1) some keywords that are used in the title of an activity; (2) a time period with a starting time and an ending time; and (3) the kind of an activity (i.e., home, school, or other). If no value is given to a component, it means that the user does not care about it and any value for that component is considered as a match. At an extreme, if no values are given for all three components, then all activities stored in the program will be returned as the search result. If values are given to more than one component, then all the values for those components have to be matched in order to identify all relevant activities for the search result.

Specific Requirements for Assignment One

Assignment One will build a basic system for Day Planner and the remaining two assignments will add further enhancements. More specifically, you are asked to implement the following tasks for this assignment:

(1) A command loop that accepts one of the three commands: *add*, *search*, and *quit*. For the *add* command, the user needs to enter the type of activity, a short title, a starting time, an ending time, and an optional comment. If it is an "other" activity, the user also needs to enter the location.

For the *search* command, the user needs to provide values to three components: title keywords, a time period (with starting and ending times), and an activitiy type (either home, school or other). Note that the user can enter an empty string (i.e., no value) for one and all of the components. For the *add* and *search* commands, you can enter the required information through separate prompts, one for each field, or enter it all together in one line. In the latter case, you need to specify a format so that the individual fields can be extracted properly. Finally, the *quit* command will simply exit the command loop and terminate the program.

(2) All activities will be stored into three arrays: one for home activities, one for school activities, and one for other activities. You can try to store as many activities as you want, but you should always check first if there are rooms available in the corresponding arrays. If there are no room in the related array, you should let the user know and no activities can be added to that array.

(3) Based on the description so far, your implementation should include at least these classes: Time, HomeActivity, SchoolActivity, OtherActivity, and DayPlanner. The Time class should contain these fields: year, month, day, hour, and minute. Our textbook illustrates about five different versions of the Date class. The Java API also contains the related Date, Time, and Calendar classes. You are free to use these examples as references and may re-use some parts of the code as you see fit, but it is probably easier to implement a Time class for your own. If you do re-use some parts of the existing code, you need to document clearly which parts are re-used in the relevant Javadoc sections. Although HomeActivity, SchoolActivity, and OtherActivity are different classes, they can be handled similarly in that they are all used to model activities. For all of HomeActivity, SchoolActivity, and OtherActivity classes, the required fields are title, starting time, ending time, and comment. For the OtherActivity class, it also requires a location field. The DayPlanner class is perhaps the most challenging, since it needs to maintain three arrays for different kinds of activities: adding new activities to the related arrays, searching the arrays sequentially for the matched activities, and displaying the result on the screen.

(4) For all the classes, try to follow the conventions by making all the instance variables private and providing suitable accessor and mutator methods. You should also provide suitable constructors and other commonly used methods for a class, including "equals" and "toString" methods. For some classes such as Time, you also need to provide the "compareTo" method so that we can order one time before another. However, since the DayPlanner class maintains the three arrays for all the activities, it will not be convenient nor relevant to implement accessor and mutator methods, as well as "equals" and "toStrings" for it.

(5) Each search request is made of three components. For the activity type, you can simply choose the corresponding array for further search. For the time period, you need to make sure that the starting and ending times of an activity fall within the specified time period. For the title keywords, you need to make sure that they all appear in the title field of an activity, although they can be in any order. Note that when matching two keywords, they have to be equal at the word level but the cases can be ignored. For example, "Java" and "java" are a match, but "Program" and "Programming" are not. Also, the title keywords "Java Programming" is matched by the title "Programming in Java", since the keywords can be matched in any order.

(6) You should organize all of your classes into one package and use Javadoc to create a set of external documents so that the TA can examine the contents of your package easily for the marking purpose.

Deliverables:

All of the implementations should be done in Java. You can develop your programs in different environments (e.g., your home computers or laptops) but you should always test them on the lab machines in Thornbrough 2420, since this will be the environment we use for marking your assignments.

Your program will be marked for both correctness and style. By correctness, we mean that (1) your programs are free of syntactic errors and can be compiled successfully; (2) your programs are logically correct; and (3) your programs should give appropriate runtime messages (i.e., prompts) and are reasonably roboust in handling user input. To make sure your programs are logically correct, you need to prepare a test plan along with a set of test cases. The test plan describes the major steps involved in testing your program and whether you have considered all possible conditions. For example, to search an element on a list, we need to consider a situation where the element is not on the list, and if the element is on the list, whether it is located at the beginning or end or in the middle of the list. The test cases provided concrete examples that can be used for testing with the corresponding input and output values. To ensure that your programs are reasonably roboust in handling user input, you need to exercise defensive programming. Although you have clearly show a prompt asking for a certain kind of input (e.g., “quit”), the user may enter a shorter input (such as “q” or “Q”) or something quite different (e.g, “bye” or “leave”). Your program should accept most of the reasonable values (such as "quit", "Quit", "QUIT", “q” or “Q”), but reject all the irrelevant values (such as “bye” or “leave”). Furthermore, for the irrelevant values, you should give the user some feedback message and ask the user to re-enter the required values.

A good style in programming allows people to understand and maintain (modify or extend) your programs easily. This often includes (1) meaningful names and well-indented layout for control structures (branches, loops, and blocks); (2) internal documentation (the various comments within your programs); and (3) external documentation (additional description outside your programs, usually the README file). In Java, different naming conventions are used for representing classes, variables and methods, and symbolic constants. There is no limit about the length of an identifier; so try to use meaningful names for the benefit of readability. For internal documentation, Java introduces “Javadoc”, which will automatically turn comments in /** some comments here */ or /* some comments here */ before public classes, public instance/class variables, and public instance/class methods into a set of web documents. In addition to Javadoc comments, the traditional comments can still be used to explain the meanings of local variables and highlight the major steps within a particular method. For external documentation or the README file, you can describe (1) the general problem you are trying to solve; (2) what are the assumptions and limitations of your solution; (3) how can a user build and test your program (also called the user guide); (4) how is the program tested for correctness (i.e., the test plan should be part of the README file); and (5) what possible improvements could be done if you were to do it again or have extra time available.

In summary, a complete submission should include: (i) a README file; (ii) all the Java source files; (iii) JavaDoc files; and (iv) all the testing files if needed.

Organize your program and related files into appropriate directories, and tar and gzip all of your files/directories into one compressed file (name it in the form of <userid>_a<#>.tar.gz, e.g., fsong_a1.tar.gz) and drop it to the course website on moodle.cis.uoguelph.ca by the due time. Please verify afterwards that your submission is indeed uploaded successfully to the moodle server.