HW5 – Building and Traversing Collections

Estimated time: 12-20 hours

Objectives

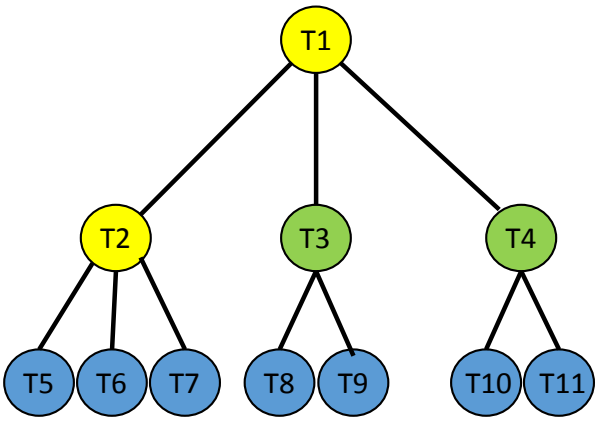
- Become more familiar with UML modeling
- Become skills at applying patterns such as the Composite, Iterator, Visitor, and Builder
- Improve unit testing skills to reduce complexity and improve performance

Overview

In this assignment, you will build a class library for working with Work Breakdown Structures (WBS's), which engineers across many different industries use to plan and manage complex projects. A Work Breakdown Structure is a hierarchy of tasks that represent the work to be completed for a projects. Engineers often descript the tasks in terms of the estimated total hours, percent complete, and estimated remaining hours. With a WBS, an engineer should be able to perform a number of management activities, such as predict the total required resources (hours), a desirable number of engineers who work concurrently, schedule, and more.

For our class library, a WBS will be a tree structure of task nodes, where the leaf nodes are atomic tasks and the non-leaf nodes are aggregate tasks whose subtasks need to be done sequentially or in parallel. Consider the WBS illustrated in Figure 1. The yellow circles represent non-leaf tasks whose subtasks have to be completed sequentially (left to right), the green circles represent non-leaf tasks whose subtasks can be completed in parallel, and the blue circles represent leaf tasks. In this example, tasks T5, T6, T7 would need to be completed in order, followed by T8 and T9 concurrently and then by T10 and T11 concurrently.

Figure 1 – Sample Work Breakdown Structure



Functional Requirements

1. The class library needs to support WBS's, where a WBS is a hierarchy of Tasks.
   1.1. A Task can be Parent Task or Leaf Task
   1.2. A Parent Task can be a Sequential Parent Task or Parallel Parent Task, and can include zero or more Tasks as subtasks.

1.3. A Task must have the following properties
    1.3.1. Id
    1.3.2. Label
    1.3.3. Description
    1.3.4. Assigned Engineers
    1.3.5. Original Estimated Total Hours
    1.3.6. Revised Estimated Total Hours
    1.3.7. Percent Complete
    1.3.8. Estimated Remaining Hours
    1.3.9. Estimated Work Days To Completion
1.4. For a Parent Task, the Estimate Total Hours, Percent Complete, Estimate Remaining Hours, and Estimated Work Days to Completion must be computed from its subtasks.
    1.4.1. For a Sequential Parent Task, the computation of Estimated Work Days to Completion must take into consideration that its subtasks have to be completed one after the other
    1.4.2. For a Parallel Parent Task, the computation of Estimated Work Days to Completion must make assume that its subtasks all start the same time and may proceed concurrently to the degree there are available resources.
1.5. The Estimate Total Hours, Percent Complete, Estimated Remaining Hours, and Estimated Work Work Days to Completion for a project can be obtains from the top-level Task of a project.

2. The class library needs to include an Engineer class whose instance can capture basic information about the available engineers
    2.1. An Engineer must have the following properties
        2.1.1. Id
        2.1.2. Name
        2.1.3. Available hours per day
3. The class library needs to provide tools for constructing and manipulating a WBS, including:
    3.1. Exporting the WBS or any subtree in a WBS as a XML file.
    3.2. Importing an XML file as a new WBS or subtask of a Parent Task in an existing WBS.
4. The class library needs to provide tools for setting up a team of engineers and their availability
5. The class library needs convenient tools for creating estimated schedules for remaining work, by work day and engineer
    5.1. A work day can be identified by a relative number from the beginning of the project. For example, day 0 is the first day when remaining work will commence. Given typical 5-day work weeks, day 10 would be the beginning of the 3rd week.
    5.2. For each day in a schedule, the schedule should contain a list of engineers working on something that day and then for each engineer a list of tasks that the engineer should be working on
6. The class library needs to provide a convenient way to get list of a Tasks to which a given engineer has been assigned.
7. The class library needs to provide a convenient way to print a WBS in an outline form to a text file.

Extra Credit Requirements

8. The class library could allow Tasks to include a daily history of work completed.
   8.1. The days of a project can be numbered using an integer sequence
   8.2. The daily history of a Task would have to contain a list of days on which it was worked
   8.3. For each day, the daily history work need to record the percent complete at the end of the day and list of engineer did work on that task that day and the hours each spent.
   8.4. The daily histories for a Parent Task are an aggregation of its subtask's daily histories
9. With the task daily histories, the class library could provide some convenient tools for measuring performance the team and individual engineers.

Instructions

To build this system, you will need to do the following:

1. Model class library using UML Class Diagrams and Interaction Diagrams.
2. Look for **good** opportunities to apply the patterns discuss to date, particularly the composite, iterator, visitor, and builder. Document your choices and justifications for those choices in a README file.
3. Implement your class library
4. Thoroughly test your class library with meaningful executable unit test cases.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw5_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Max Points
A clear and concise designs consisting of UML class diagrams and interaction diagrams. Also, a README file that describes your application of design patterns and the reasons why you felt those choices were justified.	20
Quality implementation with good encapsulation, abstraction, coupling, cohesion, and modularization	40
Thorough executable unit test cases	40

Optional Extra Credit

Criteria	Max Points
Implementation and test daily histories for Tasks.	10
Implementation and test performance measurement tools using the daily histories	20

Note: if you choose to do the extra credit, please state what you did and your design choices in your README file.