03-60-611-01
Advanced Software Eng. Topics Section 1
Lab 4: JUNIT

Jing Tao (104325892)
Pei-En Chang(104498769)
Ying Mi(104322406)
Yunheng Yao (104166626)

November 2, 2015

1 What design patterns would you consider and where in the code (justify why)? [6 pts]

1.1 Information Expert

I applied *information expert* pattern on **CTime**, **CMeeting**, **CSchedule** classes. When displaying object information by invoking *toString()* method, those classes themselves are responsible to fulfill the required printing information as well as the format.

1.2 Creator

When creating **CTime** object, one has to do so by invoking methods in **CMeeting** class. This is because I made its constructor as default level so as to protect the constructor method from being used by external classes. Hence, I assigned *Creator* pattern on **CMeeting** to high detail and reduce coupling.

1.3 Controller

I applied *Controller* pattern on **addMeeting(CMeeting)**. This method as well as the containing class serves as a controller. When adding a new meeting into container by invoking this method, it will only insert the meeting if there is no overlap.
But the time and meeting comparing jobs are actually executed in their own classes.

1.4 High Cohesion

I applied *High Cohesion* pattern on most of the **Setter** methods. Each class is responsible for checking the validity of input parameters by their own. One does not need to care about the validity of start time, stop time etc., For example, by using setters of **CMeeting**, it will ensure that end time is always after the start time and that no meeting lasts more than 1 hour.

1.5 Low coupling

Low coupling is not a method or class level design pattern, it is an objective of software design. For example, by implementing well defined setters, controller does not need to bother with validation of input parameters. Overriding **toString()** when printing object information lowered the coupling of whole application.

1.6 Indirection

When comparing 2 CMeeting objects or CTime object, the actual work is completed by the implementation of **compareTo()** method. By implementing the **Comparable** interface, the comparison could be independent from third part comparator.

2 Implement 4 JUnit Test cases (and obviously the unit you are testing) [2pts each x 4 = 8 pts]

3 Code to pass each test case [1 pt each x 4 = 4 pts]

4 Demonstrate how the automated testing works (use screen-shots to show you runs) [2pts](include your code and the test code)

A Task 1