1. Describe the 5 key activities in an object-oriented design process.
2. Define the context and modes of use of the system
3. Design the system architecture
4. Identify the principal system objects
5. Develop design models
6. Specify object interfaces
7. Consider software evolution?
8. What are the differences between refactor and reengineering?

Reengineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and reengineer a legacy system to create a new system that is more maintainable.

Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

1. Describe the tree types of software maintenances?
2. Maintenance to repair software faults：

甲、Changing a system to correct deficiencies in the way meets its requirements.

乙、Corrective maintenance

1. Maintenance to adapt software to a different operating environment：

甲、Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.

乙、Adaptive maintenance

1. Maintenance to add to or modify the system’s functionality：

甲、Modifying the system to satisfy new requirements.

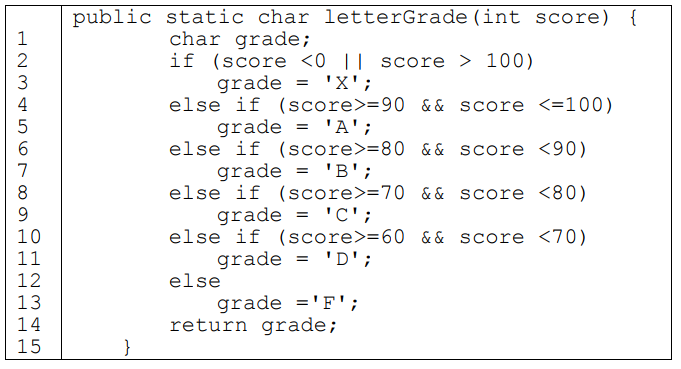
乙、Perfective maintenance

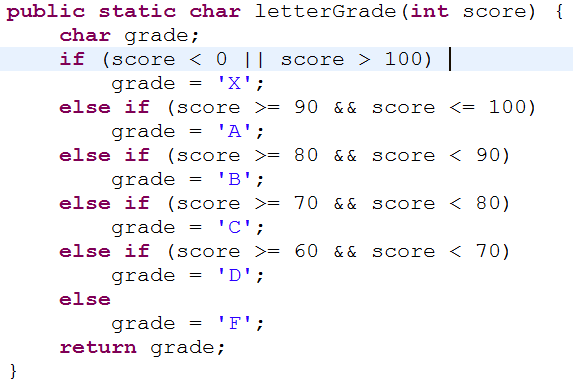
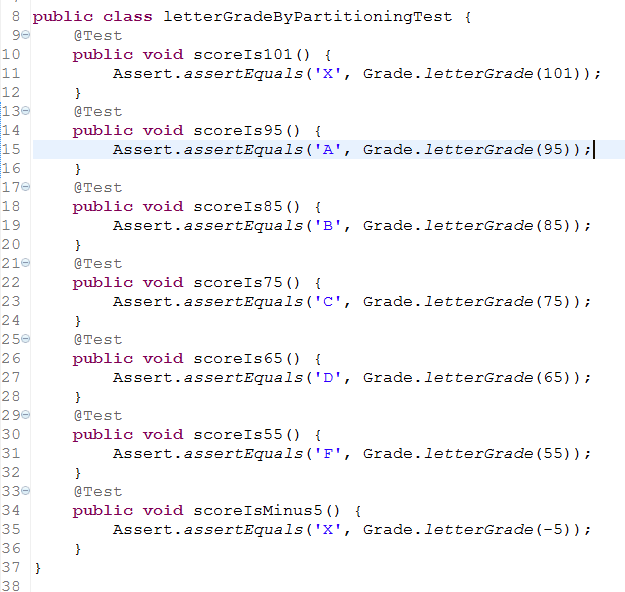
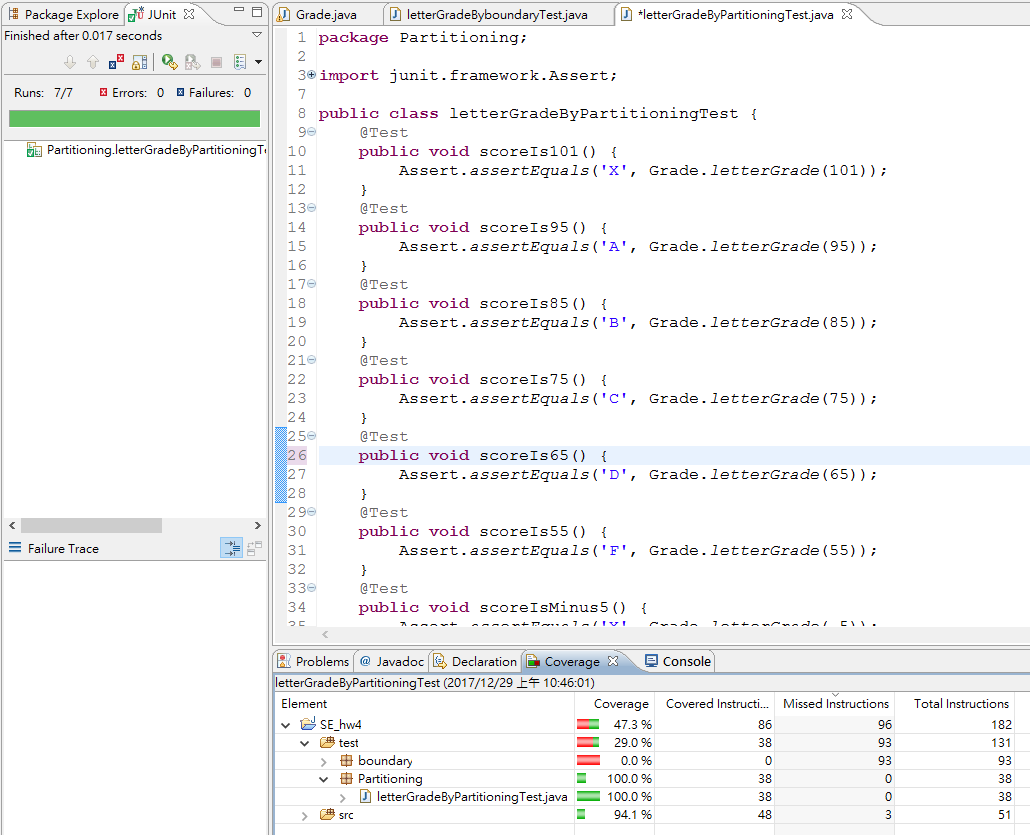
1. What are bad smells (or code smells)? How will you relate bad smells and preventative maintenance?
2. Duplicate code：The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.
3. Long methods：If a method is too long, it should be redesigned as a number of shorter methods.
4. Switch (case) statements：These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.
5. Data clumping：Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.
6. Speculative generality：This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.
7. Consider a program that takes a numerical score (ranged from 0 to 100) and transfers the score to a letter grade A(score>=90), B(80<=score<90), C(70<=score<80), D(60<=score<70), or F(score<60); otherwise X(score<0 or score>100).
8. Apply the equivalence partitioning testing technique to design test cases for testing the program.

|  |  |  |
| --- | --- | --- |
| Score range | grade | Test case |
| score >100 | X | Score = 101 |
| score>=90 | A | Score = 95 |
| 80<=score<90 | B | Score = 85 |
| 70<=score<80 | C | Score = 75 |
| 60<=score<70 | D | Score = 65 |
| score<60 | F | Score = 55 |
| score<0 | X | Score = -5 |

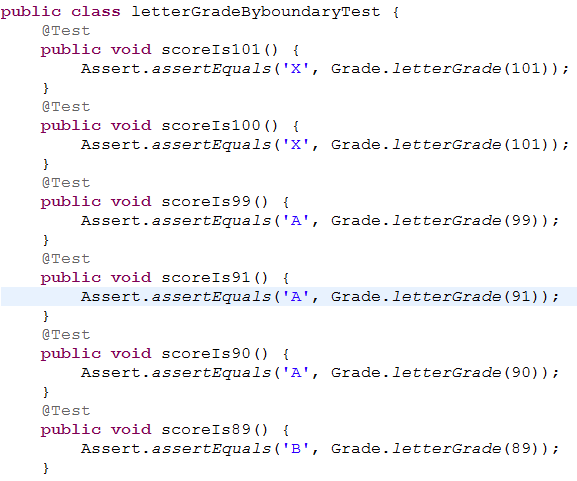
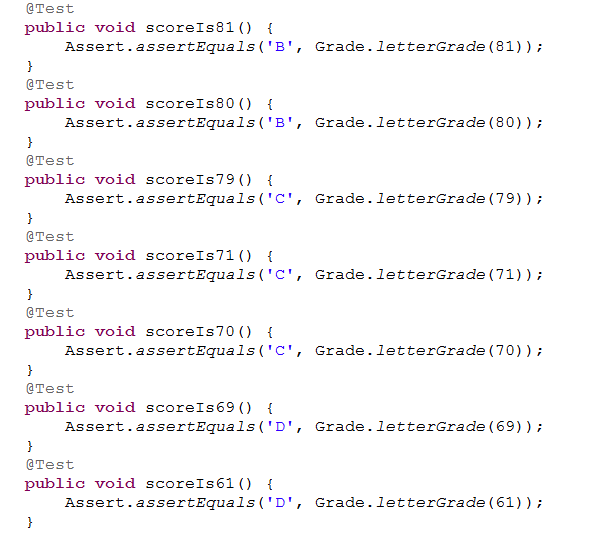
1. Based on your answers in (a), design additional test cases by applying the boundary value analysis testing technique.

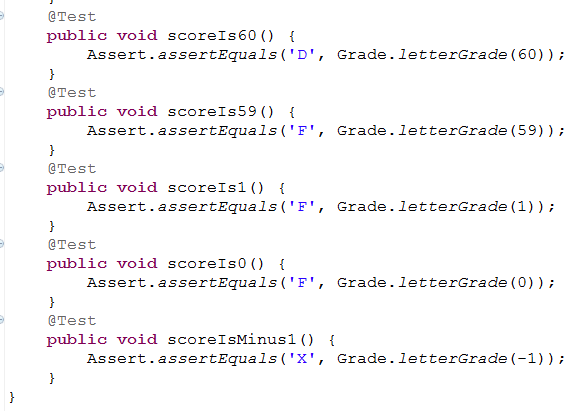
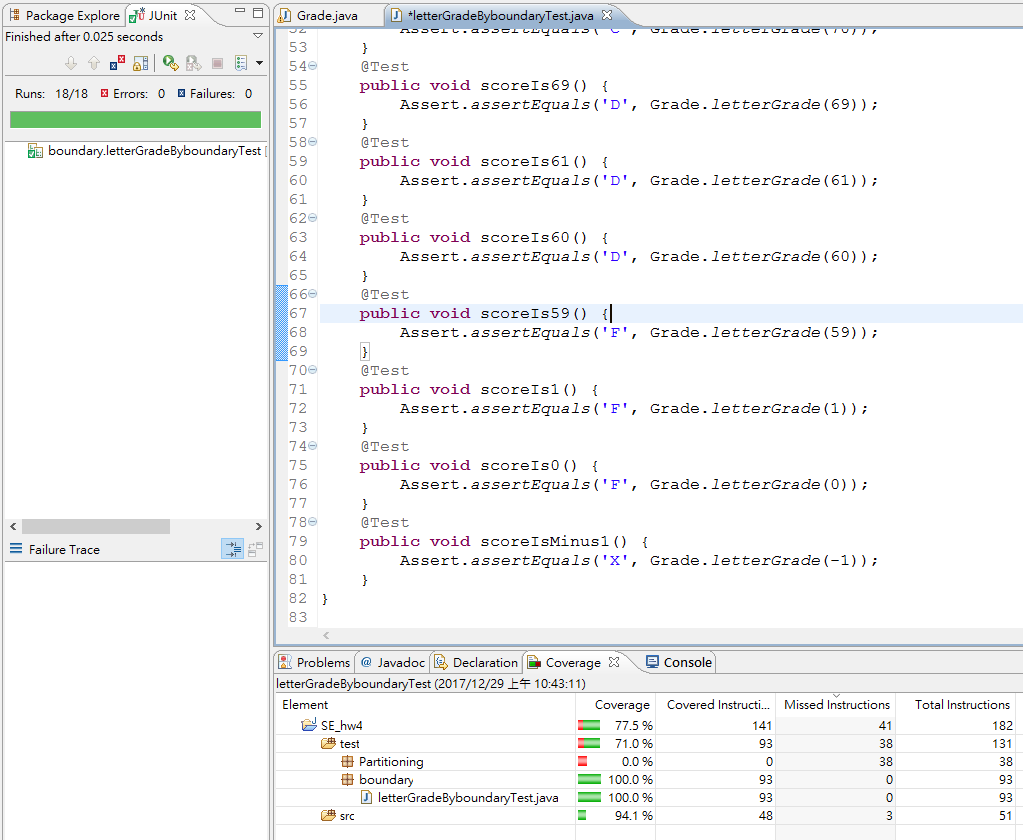
|  |  |  |
| --- | --- | --- |
| Score range | Test case | 期望output |
| score >100 | Score = 101 | X |
| Score = 100 | A |
| Score = 99 | A |
| score >=90 | Score = 91 | A |
| Score = 90 | A |
| Score = 89 | B |
| 80<=score<90 | Score = 91 | A |
| Score = 90 | A |
| Score = 89 | B |
| Score = 81 | B |
| Score = 80 | B |
| Score = 79 | C |
| 70<=score<80 | Score = 81 | B |
| Score = 80 | B |
| Score = 79 | C |
| Score = 71 | C |
| Score = 70 | C |
| Score = 69 | D |
| 60<=score<70 | Score = 71 | C |
| Score = 70 | C |
| Score = 69 | D |
| Score = 61 | D |
| Score = 60 | D |
| Score = 59 | F |
| score<60 | Score = 61 | D |
| Score = 60 | D |
| Score = 59 | F |
| score<0 | Score = 1 | F |
| Score = 0 | F |
| Score = -1 | X |

1. Consider the following program. 
2. Implement your test cases in problem 4(a) using JUnit. Show the JUnit source code of your test cases and the screen snapshots of the execution results of the test cases (including code coverage).

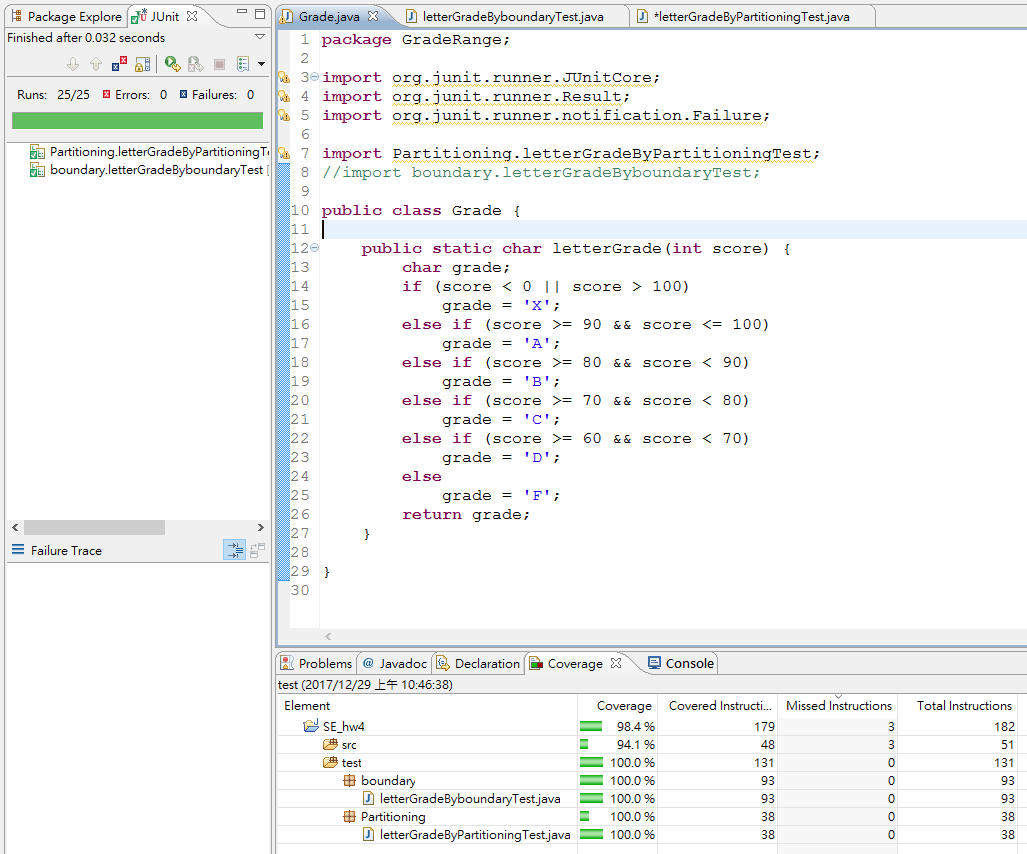
  

1. Implement your test cases in problem 4(b) using JUnit. Show the JUnit source code of your test cases and the screen snapshots of the execution results of the test cases (including code coverage).

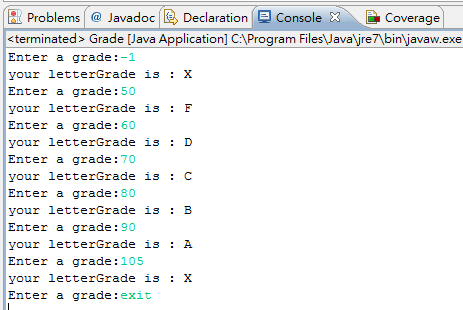
 

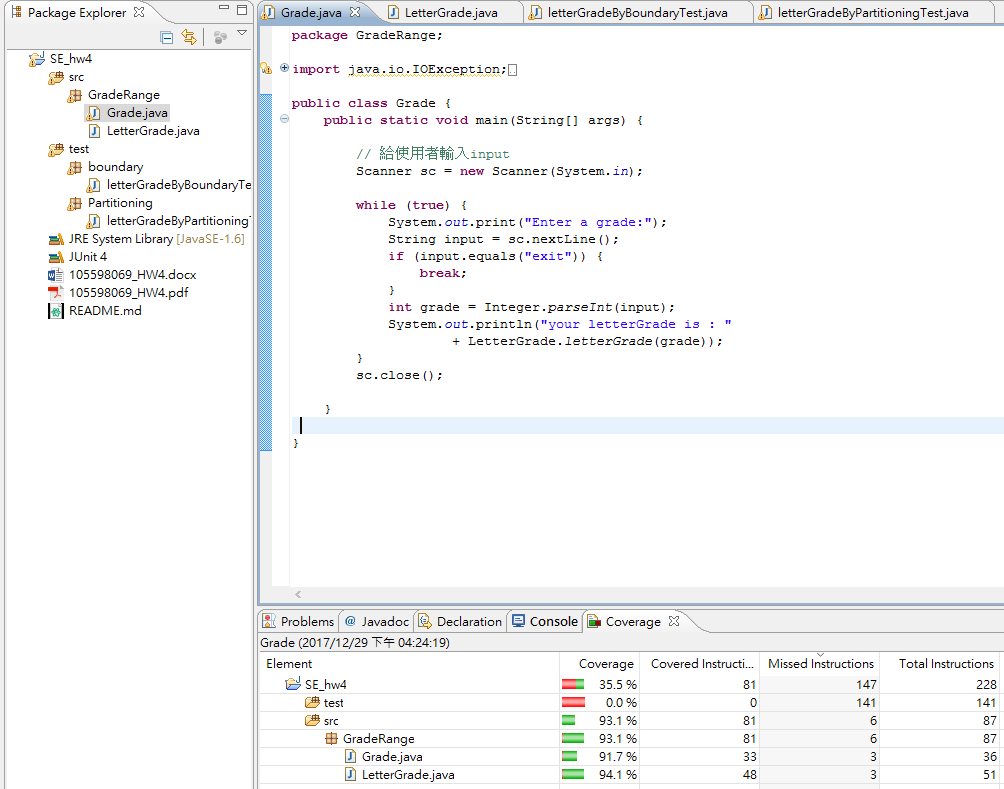
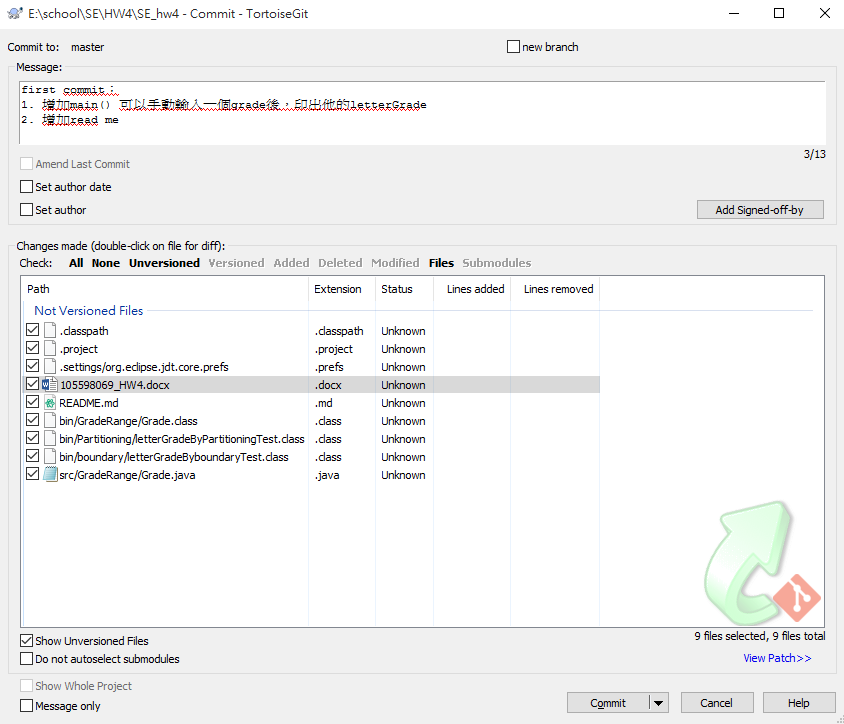
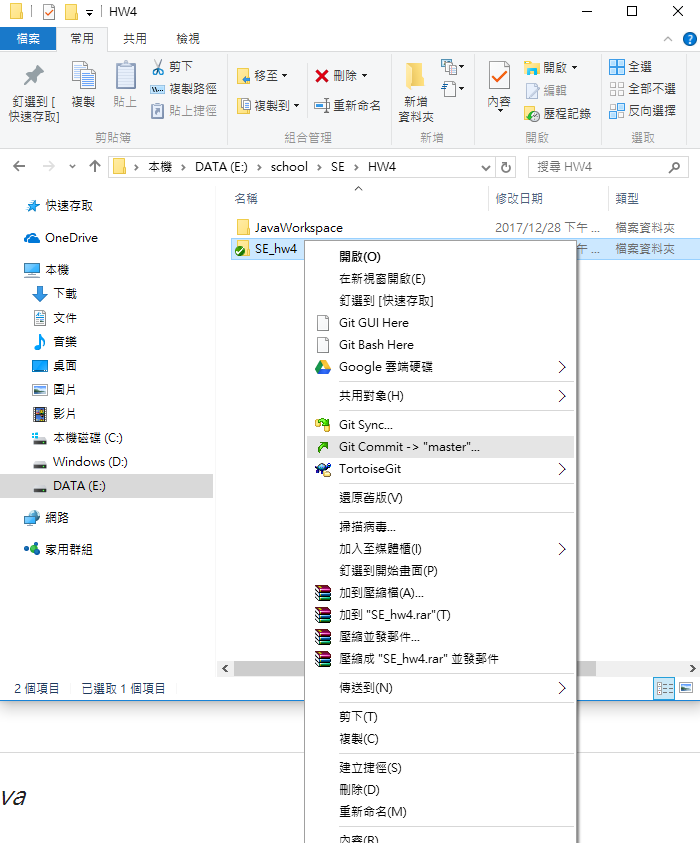
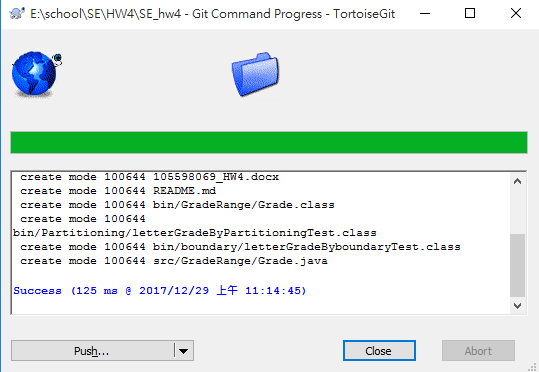
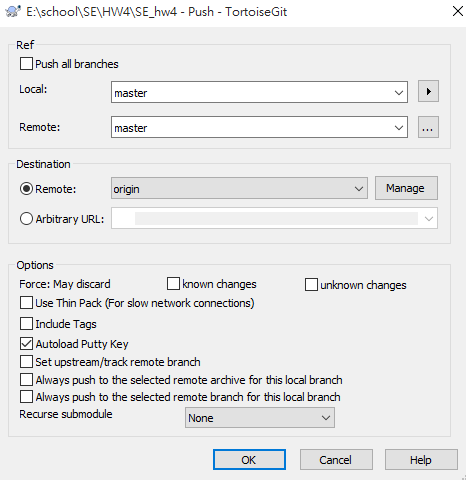
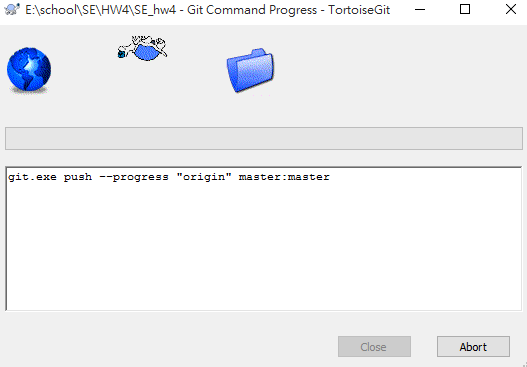
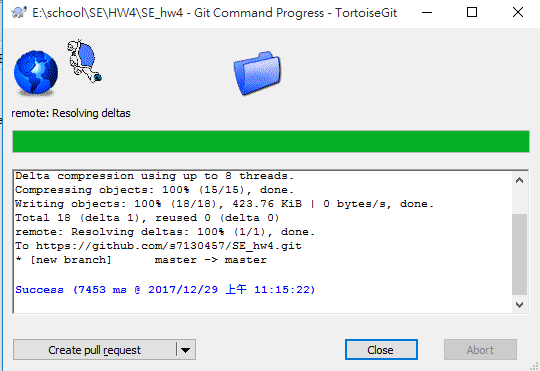
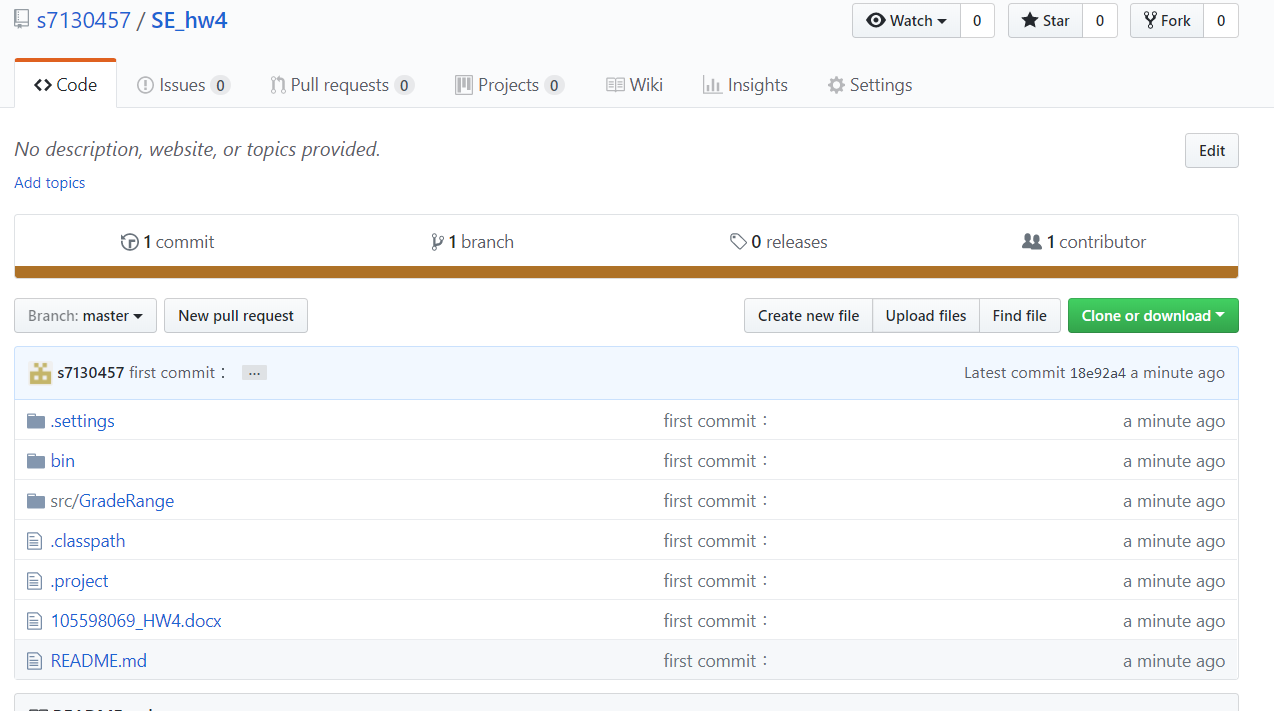
 

下圖為兩種測試方式都測的code coverage



1. Illustrate the application of the JUnit and configuration management (CM) tools, such as subversion or Git (or GitHub), in software development. Noe that you may integrate your IDE tool with your chosen CM tool, and you also need to create your own repository using the chosen CM tool.
2. Show the screen snapshots for using the CM tool to check in the source code of letterGrade.java and then check out the code to add a main()function so that the program can be executed and tested in console manually. After the manual testing is completed and the program is correct, commit the source code to the repository.



1. Show the screen snapshots for using the CM and JUnit tools to check out your source code of letterGrade.java committed in 5(a) and add JUnit test cases to test the program automatically. After all the test cases are pass and the statement coverage is 100%, commit you source code and test cases to the repository.

