

# CSIE5142/CSIE4302 Software Engineering

## Homework # 4

Due on 01/03/2023

學號:111598031 姓名: 楊佑鴻

1. (10%) Describe the 5 key activities in an **object-oriented design process**.

物件導向設計程序有以下五個關鍵活動:

- **Define the context and models of use of the system**  
定義系統使用的環境與模型，表示必須要去了解設計的軟體及其外部環境所要提供的基本功能，及如何建構該軟體、如何與該環境溝通兩者之間的關係。充分理解context可以協助建立系統的邊界，同時幫助理解要在系統內實作與其關聯系統相關的功能。
- **Design the system architecture**  
設計系統架構是區分出能建構系統及其互動的主要元件，並將這些元件使用架構的模式組織起來，抽象化地去敘述整個系統的設計，保留結構化的系統架構，省略其細節。
- **Identify the principal system objects**  
識別主要系統物件仰賴於系統設計者的技術、經驗及領域知識，是一個迭代式的過程。利用需求文件與使用者討論，及對現有系統分析來進行資訊蒐集，最後產出對應的class diagram。
- **Develop design models**  
Design model 展示了系統中物件之間的關係，可作為需求到實作前的轉換，會用許多類型的模型來表達其關係。包含使用Structural (static) models來描述靜態結構，以及使用Dynamic models來描述物件之間的動態關係。
- **Specify object interfaces**  
物件介面用於定義物件的抽象行為，透過設計界面來使一個物件可以與其他物件互相協作，並此其遵守該介面的行為。

2. (15%) Consider software design.

- (a) (6%) What are the concepts of **coupling** and **cohesion**?
- (b) (4%) What is **Inversion of Control (IoC)**? Name a technique to implement IoC.
- (c) (5%) What is **design pattern**? Name the design pattern that can be used to separate the display of object state from the object itself.

(a)

**Coupling** 定義程序模組之間相互依賴程度的度量方式，它告訴模組在什麼層級上會相互依賴及互相關連。當兩個模組間的相依性越高，那它們的耦合性越高，稱之為高耦合。反之則稱為低耦合。

**Cohesion** 定義模組元素內部依賴程度的度量方式，意即一個模組內的不同功能的相關程度。例如當處理某個任務的程式跟資料都在同一個模組內，

表示這個模組有很高的內聚力，反之則稱為低內聚。

(b)

Inversion of Control (IoC) 是一種程式原則或設計原則，用來減低電腦代碼之間的耦合度。將類別依賴關係的選擇具體實作類型的功能委託給外部元件或來源。

Dependency injection 是一種軟體設計模式，也是實現 IoC 的其中一種技術。這種模式能讓一個物件接收它所依賴的其他物件。「依賴」是指接收方所需的物件。「注入」是指將「依賴」傳遞給接收方的過程。在「注入」之後，接收方才會呼叫該「依賴」。此模式確保了任何想要使用給定服務的物件不需要知道如何建立這些服務。取而代之的是，連接收方物件（像是 client）也不知道它存在的外部代碼（注入器）提供接收方所需的服務。

(c)

Design pattern 是對軟體設計中普遍存在（反覆出現）的各種問題，所提出的解決方案。並不是直接用來完成程式碼的編寫，而是描述在各種不同情況下，要怎麼解決問題的一種方案。

Observer pattern 便是一種 Design pattern，它將物件狀態的顯示與物件本身分開，定義物件之間的一對多的依賴關係，以便當一個物件變更狀態實，它的所有依賴物件都會收到通知並自動更新。

### 3. (10%) Describe the major activities of **configuration management (CM)**.

Configuration management 是管理不斷變化的軟體系統過程的名稱，它有以下四項主要的活動：

- Version management

版本管理，其中提供支持持續追蹤軟體元件的不同版本，版本管理系統包括協調多個程式設計師開發的工具。

- System integration

系統整合，提供支持已幫助開發人員定義用於創建系統的每個版本的元件版本。然後此描述用於通過編譯和連結所需元件來自動建構該系統。

- Problem tracking

問題追蹤，提供支持允許用戶報告錯誤及其他問題，允許所有開發人員查看誰在處理這些問題，以及何時會修復這些問題。

- Release management

發布管理，向客戶發布新版本的軟體系統。發布管理與規劃新版本的功能和組織軟體發布有關。

4. (5%) What are **bad smells** (or **code smells**)?

Code smell 為代碼中的任何可能導致深層次問題的症狀。

常見的 Code smell 包含：

- Duplicate code

相同或非常相似的代碼可能出現在程式的不同位置。這可以被刪除並實現為一個獨立的方法或函式，根據需要時在再呼叫。

- Long methods

如果一個方法的程式碼過長，應該被重新設計為一些更短的方法。

- Switch (case) statements

Switch 通常涉及重複，其中的 case 取決於值的類型。Switch 語句可能散佈在程序中。在物件導向的語言中，經常可以使用多型來實現同樣的事情。

5. (20%) Consider a program that takes an integer *score* as input and returns a char *letterGrade* according to the following rules:

Score	letterGrade
$90 \leq \text{score} \leq 100$	A
$80 \leq \text{score} < 90$	B
$70 \leq \text{score} < 80$	C
$60 \leq \text{score} < 70$	D
$0 \leq \text{score} < 60$	F
$\text{score} < 0 \text{ or } \text{score} > 100$	X

(a) (10%) Apply the **equivalence partitioning** testing technique to design test cases for testing the program. List all possible partitions for the program input *score* and your test case corresponding to each partition in term of {partition of *score*, <input, expect output> }.

{  $\text{score} < 0 \text{ or } \text{score} > 100$ ), < -1, X > }  
{  $\text{score} < 0 \text{ or } \text{score} > 100$ ), < 101, X > }  
{  $0 \leq \text{score} < 60$ ), < 0, F > }  
{  $0 \leq \text{score} < 60$ ), < 50, F > }  
{  $0 \leq \text{score} < 60$ ), < 55, F > }  
{  $60 \leq \text{score} < 70$ ), < 60, D > }  
{  $60 \leq \text{score} < 70$ ), < 65, D > }  
{  $70 \leq \text{score} < 80$ ), < 70, C > }  
{  $70 \leq \text{score} < 80$ ), < 75, C > }  
{  $80 \leq \text{score} < 90$ ), < 80, B > }  
{  $80 \leq \text{score} < 90$ ), < 85, B > }  
{  $90 \leq \text{score} \leq 100$ ), < 90, A > }  
{  $90 \leq \text{score} \leq 100$ ), < 95, A > }  
{  $90 \leq \text{score} \leq 100$ ), < 100, A > }

- (b) (10%) Based on your answers in (a), design additional test cases by applying the **boundary value analysis** testing technique. List possible boundaries of each partition and your test case corresponding to the boundaries in term of {boundary, <input, expect output> }.

```
{ score < 0 or score > 100), < -1, X > }
{ score < 0 or score > 100), < 101, X > }
{ 0 <= score < 60), < 0, F > }
{ 0 <= score < 60), < 1, F > }
{ 0 <= score < 60), < 59, F > }
{ 60 <= score < 70), < 60, D > }
{ 60 <= score < 70), < 61, D > }
{ 60 <= score < 70), < 69, D > }
{ 70 <= score < 80), < 70, C > }
{ 70 <= score < 80), < 71, C > }
{ 70 <= score < 80), < 79, C > }
{ 80 <= score < 90), < 80, B > }
{ 80 <= score < 90), < 81, B > }
{ 80 <= score < 90), < 89, B > }
{ 90 <= score <= 100), < 90, A > }
{ 90 <= score <= 100), < 91, A > }
{ 90 <= score <= 100), < 99, A > }
{ 90 <= score <= 100), < 100, A > }
```

6. (20%) Consider the following program `letterGrade.java`.
- (a) (10%) Implement your test cases in problem 5(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).

- (b) (10%) Implement your test cases in problem 5(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**).

```

1      public static char letterGrade(int score)
2          {char grade;
3            if (score < 0 || score > 100)
4              grade = 'X';
5            else if (score >= 90 && score <= 100)
6              grade = 'A';
7            else if (score >= 80 && score < 90)
8              grade = 'B';
9            else if (score >= 70 && score < 80)
10             grade = 'C';
11           else if (score >= 60 && score < 70)
12             grade = 'D';
13           else
14             grade = 'F';
15           return grade;
16         }

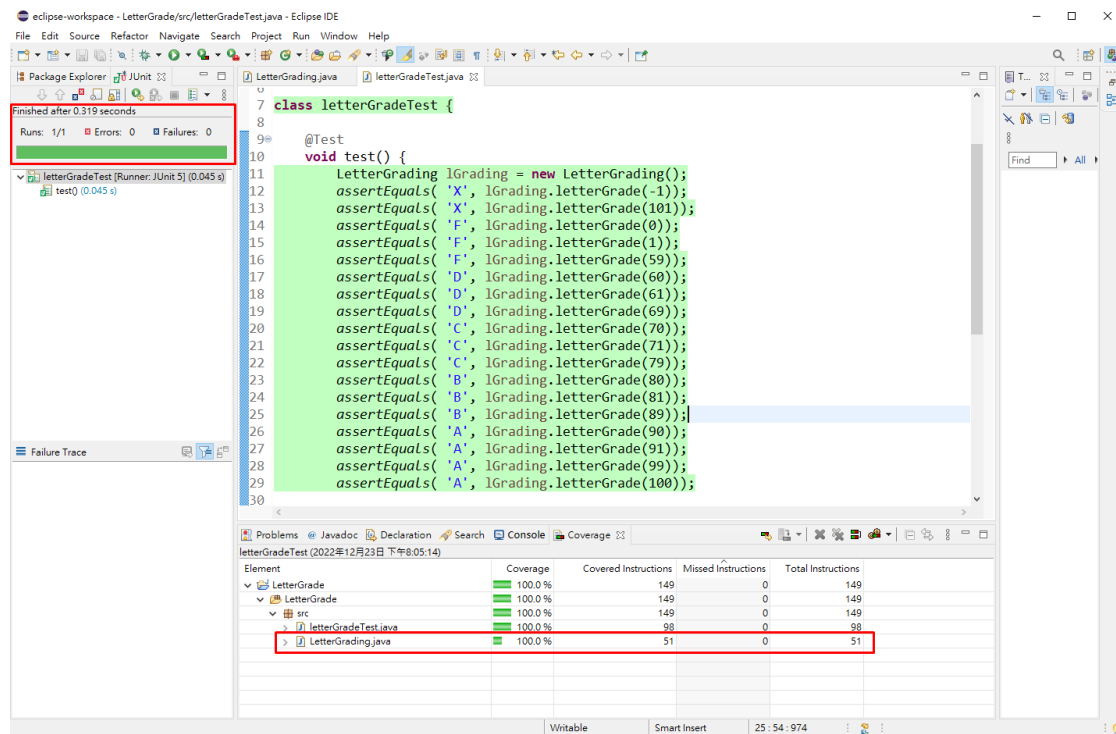
```

(a)

The screenshot shows the Eclipse IDE interface. The top part displays the JUnit test results for 'LetterGradeTest'. The test 'test()' passed successfully after 0.065 seconds. The bottom part shows the 'Coverage' tab, which provides a detailed breakdown of code coverage for the 'LetterGrade' package and its sub-packages.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
LetterGrade	100.0 %	129	0	129
LetterGrade	100.0 %	129	0	129
src	100.0 %	129	0	129
LetterGradeTest.java	100.0 %	78	0	78
LetterGrading.java	100.0 %	51	0	51

(b)



7. (20%) Illustrate the application of a configuration management (CM) tool, such as Git or GitHub, in software development. Note that you may integrate your IDE tool with your chosen CM tool, and you also need to create your own repository by using the chosen CM tool.
  - (a) (10%) Show the corresponding **screen snapshots** for using the CM tool to first **check-in** (i.e., push) the source code of `letterGrade.java` in problem 5 to your repository and then **check-out** (i.e., pull) the code to **add a main() function** given below so that the program can be executed and tested in console mode manually. **Commit the changed source code** which includes the added `main()` function to the repository when the manual testing has been performed successfully and the program is correct.
  - (b) (10%) Show the corresponding **screen snapshots** for using the CM tool to **check-out** (i.e., push) your source code of `letterGrade.java` committed in (a) and **add JUnit test cases** (i.e., test script) to test the program automatically. **Commit the source code and test cases** to the repository when all of the test cases are passed and the **statement coverage is 100%**.

```

public static void main(String[] args) {

    System.out.print("Enetr the score = ");
    try {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        score = Integer.parseInt(br.readLine());
        char grade = letterGrade(score);
        System.out.println("The grade of " + score + " is " + grade);
    } catch (NumberFormatException ex)
    { System.out.println("Not an
        integer!");
    } catch (IOException e)
    {e.printStackTrace();
    }
}

```

(a)

1. Using the CM tool to first **check-in** (i.e., push) the source code of letterGrade.java

```

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade
$ echo "# LetterGradeProject" >> README.md

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade
$ git init
Initialized empty Git repository in C:/Users/User/eclipse-workspace/LetterGrade/.git/

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (master)
$ git commit -m "first commit"
[master (root-commit) 84d849b] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (master)
$ git branch -M main

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git remote add origin https://github.com/henry27334/LetterGradeProject.git

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 233 bytes | 233.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/henry27334/LetterGradeProject.git
* [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

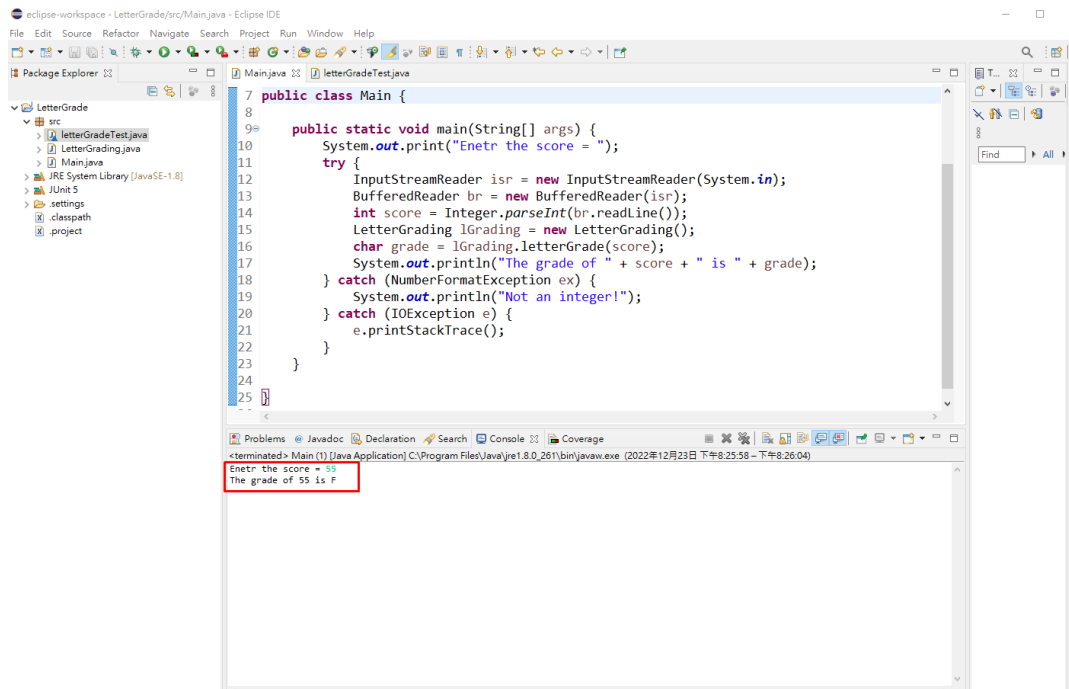
```

2. Check-out (i.e., pull) the code to add a main()function given below so that the program can be executed and tested in console mode manually

```

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git pull
Already up to date.

```



3. Commit the changed source code which includes the added main() function to the repository

```
Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git add .

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git commit -m "add main function"
[main 9ab2864] add main function
12 files changed, 164 insertions(+)
create mode 100644 .classpath
create mode 100644 .project
create mode 100644 .settings/org.eclipse.jdt.core.prefs
create mode 100644 bin/.classpath
create mode 100644 bin/.project
create mode 100644 bin/.settings/org.eclipse.jdt.core.prefs
create mode 100644 bin/src/LetterGrading.class
create mode 100644 bin/src/Main.class
create mode 100644 bin/src/LetterGradeTest.class
create mode 100644 src/LetterGrading.java
create mode 100644 src/Main.java
create mode 100644 src/LetterGradeTest.java

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git push
bash: git: command not found

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git push
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 4.02 KiB | 824.00 KiB/s, done.
Total 15 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/henry27334/LetterGradeProject.git
84d849b..9ab2864 main -> main
```

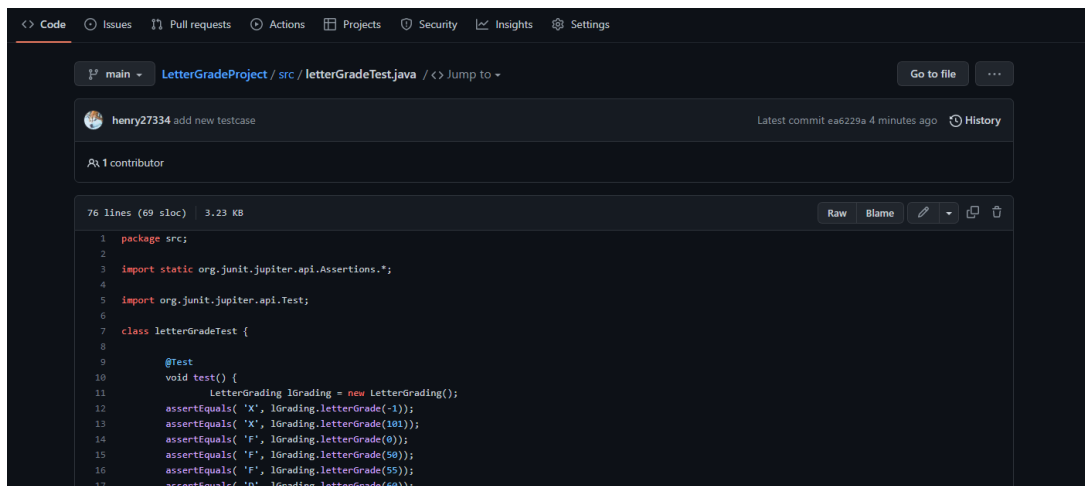


(b)

```
Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git add .

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git commit -m "add new testcase"
[main ea6229a] add new testcase
2 files changed, 29 insertions(+)

Si@TeaTalk MINGW64 ~/eclipse-workspace/LetterGrade (main)
$ git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 979 bytes | 979.00 KiB/s, done.
Total 7 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.
To https://github.com/henry27334/LetterGradeProject.git
9ab2864..ea6229a main -> main
```



The screenshot shows the GitHub web interface for the repository 'LetterGradeProject'. The commit 'add new testcase' by user 'henry27334' is displayed, with the latest commit 'ea6229a' made 4 minutes ago. Below the commit information, the source code for 'LetterGradeTest.java' is shown. The code is a Java test class with a package declaration, imports for JUnit and AssertJ, and a test method 'test()' that contains several assertions for the 'LetterGrading' class.

```
1 package src;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 class LetterGradeTest {
8
9     @test
10     void test() {
11         LetterGrading lGrading = new LetterGrading();
12         assertEquals('X', lGrading.letterGrade(1));
13         assertEquals('X', lGrading.letterGrade(101));
14         assertEquals('F', lGrading.letterGrade(0));
15         assertEquals('F', lGrading.letterGrade(50));
16         assertEquals('F', lGrading.letterGrade(55));
17         assertEquals('D', lGrading.letterGrade(60));
```