Correction Examen Testing :

**1 Tests unitaires**

@Test

public void testDifferentZero()

 {

 for(int i=1; i<1000; ++i) {

 assertEquals(1/i, inverse(i), 0);

 assertEquals(-1/i, inverse(-i), 0);

 }

}

@Test(expected=IllegalArgumentException)

public void testNullValue() {

 assertEquals(0, inverse(0), 0);

}

En cas d'exception soulevée, @Test est capable de définir une exception à supporter. Beaucoup n'ont pas vu cela, puisque je l'ai donné au cas par cas. De ce fait beaucoup ont eu bon avec des versions différentes !

**2 Tests de couverture**

1) La ligne 7 n'était pas couverte car la boucle for n'est pas correcte :

* juste avant la boucle, nous avons : x=1 et y=-1
* L'arrêt de la boucle for étant définie par x<y, elle il est donc toujours faux.

2) Il y a de nombreuses façon de procéder :

 - changer la valeur de y, ou celle de x

 - changer la condition d'arrêt (beaucoup ont vu qu'il serait préférable d'avoir i<y par ex.)

ex : for(int i=y; i<x; ++i) fonctionne.

**3 Test du Tri Shell**

1 et 2) Je n'ai pas de quoi réaliser de graphique sur ce pc. Sachez que la plupart vous avez correctement répondu aux deux questions, et, vu que nous ne l'avions pas vu en cours, j'ai de plus été très souple. Excepté quelques cas, vous vous en êtes donc tous bien sorti.

Le bon nombre était de 5. Encore une fois, même avec une mauvaise valeur, cela n'indique pas que vous n'ayez pas eu une bonne note...

3) Il fallait ici déduire du nombre cyclomatique (par exemple 7), qu'il y avait donc 7 tests unitaires à faire. Tout simplement. Les tests ensuite n'étaient que des variables modifiées au niveau des paramètres !

**4 Problème**

1) Nous avons vu en cours différent moyen pour réaliser cela. Beaucoup de réponses différentes ont étés acceptées, mais la principale était bien sur celle vue lors du TP de "pair coding", puisqu'il était question à ce moment de définir des interfaces, et que l'un créé le code du logiciel, et l'autre, à l'aide de l'interface, code les tests unitaires correspondant.

Mockito était aussi acceptable, même si nous ne l'avons pas vu en cours.

2) N'ayant pas de logiciel de graphique, voici une ébauche :

Un étudiant est représenté avec sa promo : donc un étudiant apparait plusieurs fois suivant l'année (simplification il faudrait deux tables ici)...

Etudiant(id, nom, prenom, promotion)

La note minimum étant soumise à changement, il faut donc pouvoir la changer en fonction de l'année. Cela permet de gérer un changement de note, et une note minimale différente (ex : Stage à 12 minimum)

Matière (id, nom, annee, noteMinimum, note)

Idem pour UV les paramètres nécessaire sont identiques, avec semestre en plus...

UV(id, nom, annee, noteMinimum, note, semestre, Liste de matières)

Permet de lier un semestre à l'étudiant, et de définir si c'est le premier ou second semestre

SemestreUV(UV, Etudiant, premierSemetre (boolean)

Il faut enfin rajouter passage() dans étudiant, et nous obtenons un système suffisamment souple pour réaliser les questions.

Une version plus complète était bien sur préférable, étant donné que nous souhaitions obtenir quelque chose de modulable, mais ceci était déjà suffisant.

PS : il y a des dizaines de manières de faire, la correction n'est donc pas la seule valable.

3) Il suffisait ici d'appeler passage en créant les objets nécessaires, et en plaçant les notes aux endroits attendus. Ceci ne représentait pas de difficulté particulière hormis la longueur (cf exo 1 pour un exemple de test unitaire).