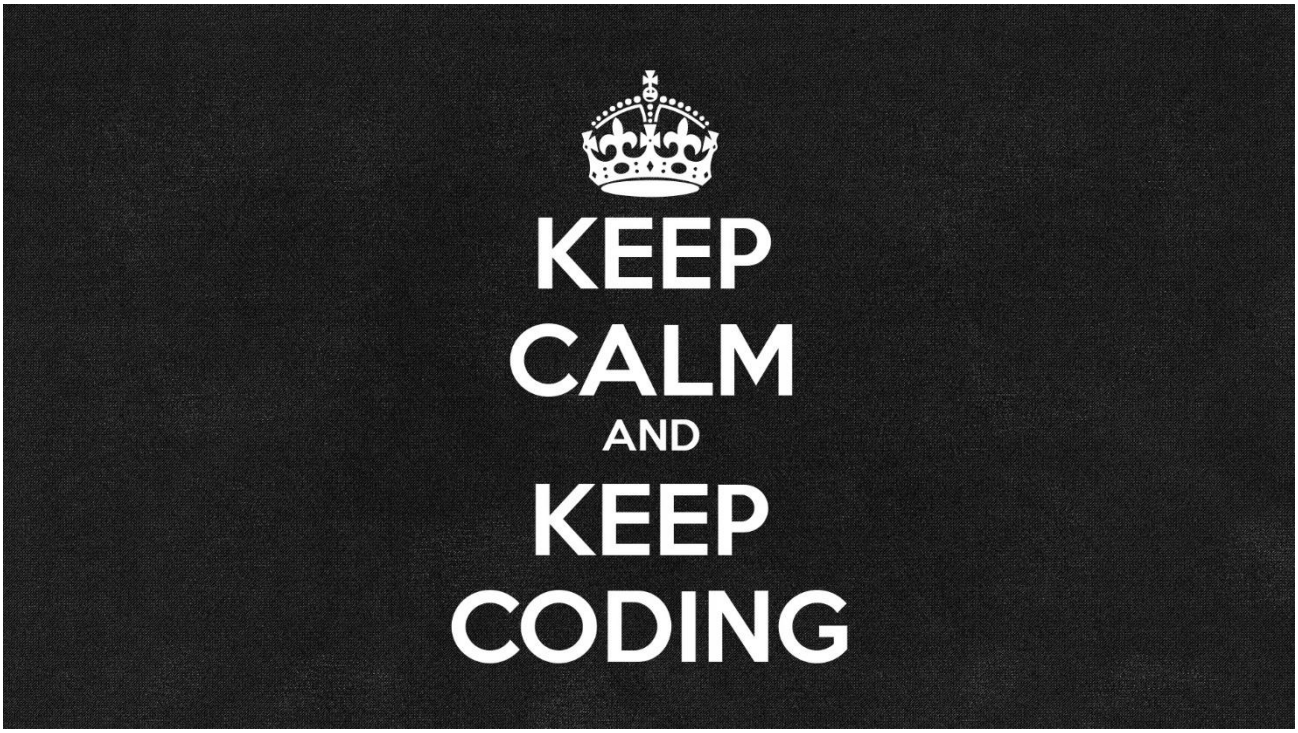


PRISES DE NOTES DE L'AMPHI 7 DE CPOA

TESTS 3/11/2017

PRISES DE NOTES DE L'AMPHI 7 DE CPOA

ASFAR MOHAMED RAFFIQUE 3B1

CPOA AMPHI n°7 du 03/11/2017– Tests

Rappel :

Objets immuables : C'est un objet qu'on ne peut pas changer, champs constants en JAVA, facilite l'alimentation sur ces objets, partager les données entre 2 objets car ça ne change jamais, toutes les propriétés qu'on peut établir sont vraies.

L'immuabilité n'implique pas que l'objet, tel qu'il est stocké en mémoire, ne puisse être réécrit. Il s'agit plutôt d'une directive de compilation indiquant ce qu'un programmeur a le droit de faire via l'interface normale de l'objet, et non ce qu'il est possible de faire dans l'absolu (par exemple en contournant le système de typage ou en violant les règles de gestion des const en C ou en C++).

Inconvénients : Si nous souhaitons changer un objet, il faut changer les champs.

Debugging : la principale technique c'est de former une hypothèse et tester. Un débogueur (ou débugueur, de l'anglais debugger) est un logiciel qui aide un développeur à analyser les bogues d'un programme. Pour cela, il permet d'exécuter le programme pas-à-pas, d'afficher la valeur des variables à tout moment, de mettre en place des points d'arrêt sur des conditions ou sur des lignes du programme ...

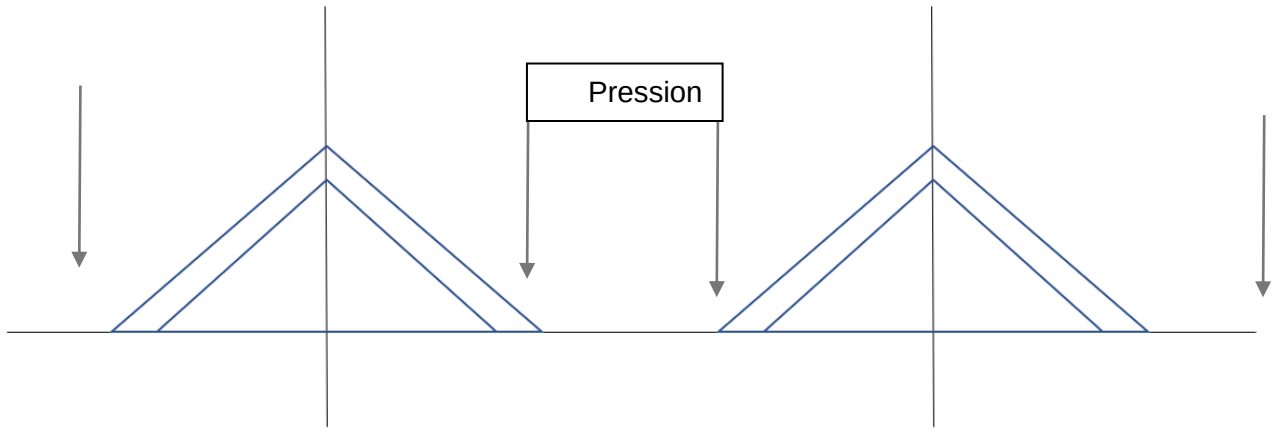
Les tests

Définition : Un test est un ensemble de cas à tester (état de l'objet à tester avant exécution du test, actions ou données en entrée, valeurs ou observations attendues, et état de l'objet après exécution), éventuellement accompagné d'une procédure d'exécution (séquence d'actions à exécuter). Il est lié à un objectif.

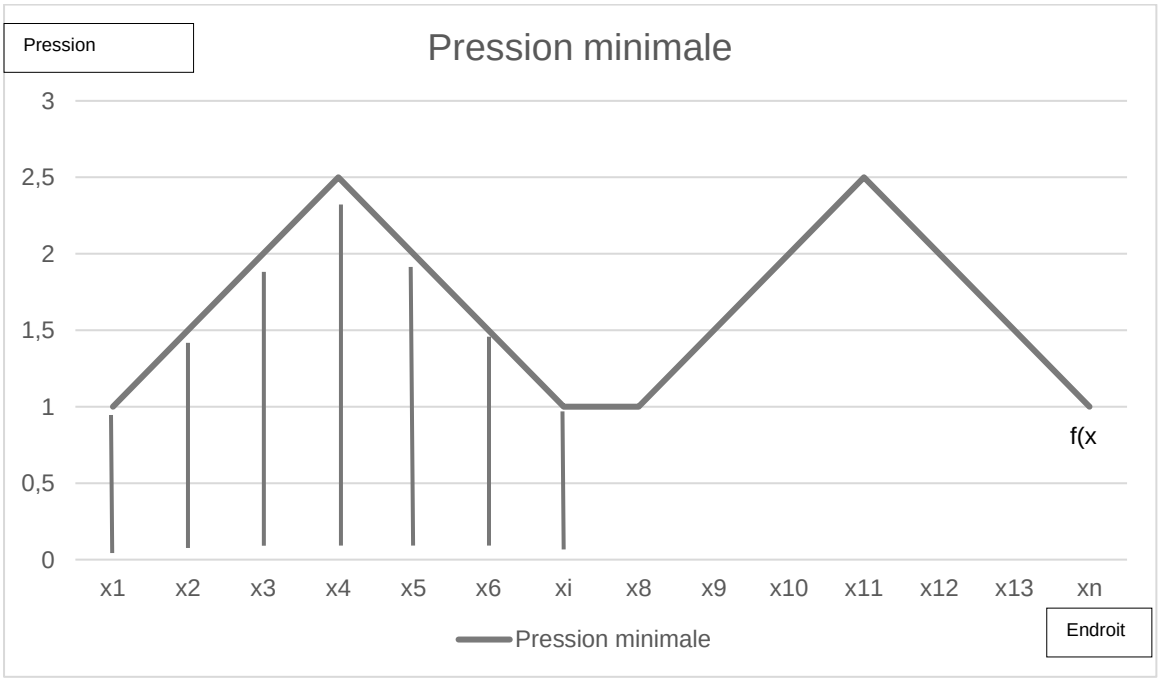
Testing : faire des tests ce n'est pas que pour l'informatique, on veut savoir si notre structure va fonctionner ou pas (exemple : un pont).

Les tests de vérification ou de validation visent ainsi à vérifier que ce système réagit de la façon prévue par ses développeurs (spécifications) ou est conforme aux besoins du client, respectivement. Le but d'un test est de vérifier qu'une fonctionnalité fait ce que l'on attend d'elle.

Exemple du pont :



On peut effectuer des tests du pont en mettant de la pression à plusieurs endroits.



f(xi)>=min+c

∃ c ∀x f(x)>=min

Dans les systèmes informatiques, c'est différent car ici nous voulons tester un nombre infini de x.

Les problèmes pour les tests informatiques :

- Il n'y a pas de continuité

- Les espaces sont finis mais très grands

- Tests qui prennent beaucoup de temps même avec les meilleurs ordinateur

Exemple :

```
import java.util.Random;

public class Carre {
        // retourne le carre de l'entier donne en parametre
        static int carre(int x) {
                if(x % 2 == 0)
                        return x*x;
                else
                        return 42;
        }
        public static void main(String[] args) {
                Random rand = new Random();
                for(int i = 0; i < 2000; ++i) {
                        int x = rand.nextInt();
                        System.out.println(x);
                        assert(carre(x) == x*x);
                }
                System.out.println("OK");
        }
}
```

Problème, on ne peut pas voir l'intérieur, notion de « black box testing »

-ea : enable assertion

Si on ne possède pas le code source et qu'il y a une erreur, on ne sait pas ce qui provoque l'erreur.

Comment choisir les tests ?

int f(int x, int y)

Si x>0 et y<0, alors renvoyer y-x

Tests :

1) x=0, y<=0

2) x=0, y<=0

3) x=0, y=0

4) x=0, y=0

Il faut toujours tester les cas extrêmes :

If (x>=){

} else {

}

• couvrir les différents cas de la spécification (black-box)

• couvrir les différentes branches du code (glass-box)

• cas de bord

• cas extrêmes (stress testing)

• aléatoirement (!) : mieux quand on peut voir, quand on a aucune idée de ce qu'on veut tester

Quand écrire les tests ?

• dès qu'on a des spécifications stables

• aller de 0% de tests corrects à 100%

• quand on a un premier bug, il peut être trop tard pour une solution simple

Comment tester ?

Outil auto (JUnit) -> pour tester un objet ou une méthode

Annotation @test

Résumés des tests affichés dans la colonne de gauche.

Toute assertion commence avec : « assert »

Plusieurs frameworks open source sont utilisables dans le monde Java notamment :

JUnit : C'est le plus ancien et le plus répandu ce qui en fait un standard de facto

JUnit est à l'origine de plusieurs frameworks similaires pour différentes plate-formes ou langages notamment nUnit (.Net), dUnit (Delphi), cppUnit (C++), ... Tous ces frameworks sont regroupés dans une famille nommée xUnit.

JUnit est un framework open source pour réaliser des tests unitaires sur du code Java. Le principal intérêt est de s'assurer que le code répond toujours au besoin même après d'éventuelles modifications.

Le but est d'automatiser les tests. Ceux-ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les comparent avec ces résultats.

Avec Junit, l'unité de tests est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :

création d'une instance de la classe et de tout autre objet nécessaire aux tests appel de la méthode à tester avec les paramètres du cas de test comparaison du résultat obtenu avec le résultat attendu : en cas d'échec, une exception est levée.

Exemple : Assertions.assertEquals(1, 0, t.getA()) ;

La classe triangleTest.java

```
import static org.junit.jupiter.api.Assertions.*;

import java.util.stream.Stream;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.ValueSource;

class TriangleTest {

	@ParameterizedTest
	@MethodSource(value = "genererParams")
	final void testGetA(double a) {
		Triangle t = new Triangle(a, 2.0, 3.0);
		Assertions.assertEquals(a, t.getA());
	}

	static Stream<Arguments> genererParams() {
		Stream.Builder<Arguments> sb = Stream.builder();

		for(int i = 0; i < 1000; ++i) {
			sb.accept(Arguments.of(Math.random()));
		}

		return sb.build();

	}

}
```