Validación y Verificación 2019
Práctico 1
Testing de Unidad

Descargue practico1.zip del moodle de la materia, y realice los siguientes ejercicios, corrigiendo en todos los casos los defectos descubiertos por sus tests.

Ejercicio 1: Considere la clase *SimpleRoutines*, realice lo siguiente:

a) Provea varios tests positivos para *max*.
b) Implemente max para que sea genérica.
c) Provea varios tests positivos y negativos para *max*.
d) Corra todos los tests producidos, y depure sus implementaciones.

Ejercicio 2: Considere la clase *Point*, realice lo siguiente:

a) Construya tests para la clase *Point*
b) Implemente el método *equals* de la clase.
c) En caso de haber detectado bugs, corríjalos y vuelva a ejecutar los tests, hasta que todos pasen.

Ejercicio 3: Considere nuevamente la clase *SimpleRoutines*, realice lo siguiente:

a) Testee los métodos *largest* y *bubbleSort*, intentando proveer cierta variedad en las entradas posibles de cada método.
b) En caso de haber detectado bugs, corríjalos y vuelva a ejecutar los tests, hasta que todos pasen.

Ejercicio 4: Implemente en java el método de *selectionSort*, el cual ordena un arreglo de enteros. Utilice la implementación de *bubbleSort* del ejercicio anterior para testear su implementación.

Ejercicio 5. Tome el método *capicua* (clase *SimpleRoutines*). Realice las siguientes actividades.

1. ¿Cuál es el defecto en el programa?
2. Si es posible, provea un test JUnit que no ejecute el defecto
3. Si es posible, provea un test JUnit que ejecute el defecto y no resulte en una falla
4. Si es posible, provea un test JUnit que ejecute el defecto y produzca una falla
5. Corrija el defecto y vuelva a ejecutar los tests hasta que todos sean exitosos

Ejercicio 6. Escriba tests parametrizados (usando @Parameterized de JUnit) para el método *bubbleSort* del ejercicio 3.

Ejercicio 7. Re-implemente la función *fibonacci* (clase SimpleRoutines) para que pase el siguiente test:

```
@Test(timeout=1000)
 public void test() {
        long res = SimpleRoutines.myFibonacci(45);
        org.junit.Assert.assertEquals(res, 1134903170);
 }
```

Puede utilizar *fibonacci* para testear su implementación:

```
@Test
public void test2() {
        long res1 = SimpleRoutines.fibonacci(3);
        long res2 = SimpleRoutines.myFibonacci(3);
        org.junit.Assert.assertThat(res2, is(res1));
}
```

Ejercicio 8. Implemente el método *minedNeighbours* de MineField.java. Escriba tests JUnit para dicho método, utilizando el setup (Before) para proveer una configuración del tablero.

Ejercicio 9. Considera nuevamente la clase *Point*,

1. Escriba una teoría para testear el método *distanceTo* de la clase *Point*. Algunas propiedades que podría chequear son:
   -La distancia del punto *a* al punto *b* es igual a la distancia del punto *b* al punto *a*
   -Cuando dos **puntos** (x_1,y_1) y (x_2,y_2) se encuentran ubicados sobre el eje x (de las abscisas) o en una recta paralela a este eje, la **distancia** entre los **puntos** corresponde al valor absoluto de la diferencia de sus abscisas ($x_2 – x_1$)
   -Alguna otra propiedad que se le ocurra.

2. Escriba un generador de puntos para genera **n** puntos con valores aleatorios entre **min** y **max**.

Ejercicio 10. Considere la implementación de Listas sobre arreglos dada. Implemente un generador de listas que permite, mediante una teoría, testear el método *remove* para listas de diferentes tamaños y valores.

E**jercicio 11.** ¿Qué relación debe mantener el método hashCode con equals? Provea un método hashCode para Point y escriba una teoría (y un generador de puntos) para testear que esta relación se cumple para su método.