Validación y Verificación 2019

Práctico 3

Testing de Mutación

Ejercicio 1: Dado los siguientes mutantes:

Si es posible, encontrar entradas de *test* que satisfagan los siguientes ítems:

a) Una entrada de *test* que no alcance el mutante.

b) Una entrada de *test* que alcance la mutación pero no produzca infección para el mutante.

c) Una entrada de *test* que mate débilmente al mutante.

d) Una entrada de *test* que mate fuertemente al mutante.

```
//Pos: Si numbers es null arroja IllegalArgumentException
//sino retorna la última ocurrencia de val en numbers[]
//Si val no está en numbers[] retorna -1
```

```
public static int findVal(int numbers [], int val){
	if(numbers ==null){
		throw new IllegalArgumentException ("entrada inválida");
	}
	int findVal = -1;
	for (int i=0; i<numbers.length; i++)
		if (numbers [i] == val)
			findVal = i;
	return findVal;
}
```

Programa mutante 1:

```
public static int findVal(int numbers[], int val){
	if(numbers ==null){
		throw new IllegalArgumentException("entrada inválida");
	}
	int findVal = -1;
	for (int i=0+1; i<numbers.length; i++)
		if (numbers [i] == val)
			findVal = i;
	return findVal;
}
```

Programa mutante 2:

```
public static int findVal(int numbers[], int val){
	if(numbers ==null){
		throw new IllegalArgumentException("entrada inválida");
	}
	int findVal = -1;
	for (int i=0; i<numbers.length; i++)
		if (numbers [i] != val)
			findVal = i;
	return findVal;
}
```

```
//Post: Si x es null arroja IllegalArgumentException
//sino retorna la cantidad de números pares en array
```

```
public static int contarPares(int[] array) {
	if(array ==null){
		throw new IllegalArgumentException("entrada invalida");
	}
	int cantPares = 0;
	for (int i=0; i<array.length; i++) {
	    if (array[i] % 2 == 0) cantPares++;
	}
	return cantPares;
}
```

programa mutante 1:

```
public static int contarPares(int[] array) {
	if(array ==null){
		throw new IllegalArgumentException("entrada invalida");
	}
	int cantPares = 1;
	for (int i=0; i<array.length; i++) {
	    if (array[i] % 2 == 0) cantPares++;
	}
	return cantPares;
}
```

programa mutante 2:

```
public static int contarPares(int[] array) {
	if(array ==null){
		throw new IllegalArgumentException("entrada invalida");
	}
	int cantPares = 0;
	for (int i=0; i<=array.length; i++) {
	    if (array[i] % 2 == 0) cantPares++;
	}
	return cantPares;
}
```

```
programa mutante 3:
public static int contarPares(int[] array) {
       if(array ==null){
              throw new IllegalArgumentException("entrada invalida");
       }
       int cantPares = 0;
       for (int i=0; i<=array.length; i++) {
           if (array[i] % 2 != 0) cantPares++;
       }
       return cantPares;
}
```

Ejercicio 2: Instale ***Pittest*** (http://pitest.org/). Genere automáticamente mutantes para la clase *BoundedQueue* y la suite de test *BoundedQueueTest* provistas en practico3.zip. Mida cobertura de mutación de la suite provista en el mismo paquete:

a) ¿Cuántos mutantes se obtuvieron?
b) Identifique mutantes muertos, mutantes vivos y no cubiertos.
c) ¿Qué *mutation Score* se obtuvo?
d) Mejore el *mutation score* extendiendo la suite de *test* dada.

Ejercicio 3: Utilice el método *mergeSort* de la clase *Sorting.java* y la *suite de test* SortingTest.java provistos en practico3.zip para correr ***Pittest***. Analice la salida obtenida.

a) ¿Cuántos mutantes se obtuvieron?
b) Identifique mutantes muertos, mutantes vivos y no cubiertos.
c) ¿Qué *Mutation Score* se obtuvo?.
d) Agregue los test necesarios para mejorar el *Mutation Score* obtenido.