Ingeniería de Software
Año 2025
Práctico 6: Particionado del espacio de entradas

Ejercicio 0. Descargue el código provisto en el repositorio git a continuación:

https://github.com/ingenieria-de-software-unrc/practicos-ingenieria-25

La carpeta `6-partition` contiene el código complementario a esta práctica. Se proveen scripts gradle para construir (*build*) automáticamente el proyecto (`./gradlew build`), ejecutar el main (`./gradlew run`), y ejecutar los tests (`./gradlew test`).
Se recomienda utilizar algún IDE (por ejemplo, IntelliJ IDEA) para facilitar la codificación, el refactoring, la ejecución de tests y el debugging.

Importante: Tenga en cuenta los siguientes requisitos al realizar los ejercicios. Piense en un diseño para su aplicación y realice un diagrama de clases del diseño propuesto antes de comenzar a codificar. Mantenga el diagrama de clases actualizado con el diseño de su aplicación durante todo el ejercicio. Asegúrese que su diseño adhiere (en la medida de lo posible) a los principios de diseño vistos en la teoría. Utilice la metodología de TDD.

Ejercicio 1. Retomando el ejemplo de la teoría que consiste de una lista L y dos características sobre la misma: "L ordenada ascendentemente" y "L ordenada descendentemente". Cada característica tiene dos bloques (verdadero y falso). Provea tests para satisfacer el criterio All Combinations Coverage.

Ejercicio 2. Dada la siguiente especificación de un método search() que opera sobre listas:

```
/*
     Si list o element son null arroja una excepción
     (NullPointerException). Sino, si el elemento está en la lista
     retorna un índice de element en la lista; caso contrario
     retorna -1. Por ejemplo,
        search ([3,3,1], 3) = 0 o 1,
        search ([1,7,5], 2) = -1
*/
public static int search(List list, Object element)
```

y el siguiente modelo del espacio de entradas:

Caracteristica: C1= Ubicación del elemento en la lista.

Bloque 1: element está en la primera posición de la lista.
Bloque 2: element está en la última posición de la lista.
Bloque 3: element está en alguna posición de la lista que no es la primera ni la última.

Resolver los siguientes problemas.
a) C1 no cumple con la propiedad de no solapamiento (disjointness). De un ejemplo que ilustre esta situación.
b) C1 no cumple con la propiedad de completitud (completeness). De un ejemplo que ilustre esta situación.
c) Defina un nuevo modelo del espacio de entradas que capture el propósito de C1 y que satisfaga las propiedades de completitud y no solapamiento.

Ejercicio 3. Responda las siguientes preguntas para el método intersection() a continuación:

```
/* Si s1 o s2 son null, lanzar una NullPointerException.
   Sino retornar un Set igual a la intersección de s1 y s2.
 */
public Set intersection (Set s1, Set s2)
```

Característica: C1 = Validez de s1
Bloque 1: s1 = null
Bloque 2: s1 = {}
Bloque 3: s1 tiene al menos un elemento
Característica: C2 = Relación entre s1 y s2
Bloque 1: s1 y s2 representan el mismo conjunto
Bloque 2: s1 es un subconjunto de s2
Bloque 3: s2 es un subconjunto de s1
Bloque 4: s1 y s2 no tienen elementos en común

a) ¿La partición para C1 satisface la propiedad de completitud? Si la respuesta es negativa dar un contraejemplo.
b) ¿La partición para C1 satisface la propiedad de no solapamiento? Si la respuesta es negativa dar un contraejemplo.
c) ¿La partición para C2 satisface la propiedad de completitud? Si la respuesta es negativa dar un contraejemplo.
d) ¿La partición para C2 satisface la propiedad de no solapamiento? Si la respuesta es negativa dar un contraejemplo.
e) Revise las características para eliminar cualquier problema que encuentre.

Ejercicio 4. Asuma un método bajo test m con tres parámetros p1, p2 y p3. Asuma que las características en la tabla a continuación particionan el espacio de entradas de m de la siguiente manera: Value 1 particiona el dominio de v1, Value 2 particiona el dominio de v2, y Operation particiona el dominio de p3.

Characteristics	Block 1	Block 2	Block 3	Block 4
Value 1	< 0	0	> 0	
Value 2	< 0	0	> 0	
Operation	+	−	×	÷

a) Provea tests para satisfacer el criterio Each Choice Coverage.
b) Provea tests para satisfacer el criterio Base Choice Coverage. Asuma como base Value 1 > 0, Value 2 > 0 y Operation = +.
c) ¿Cuántos tests se necesitan para satisfacer All Combinations Coverage? (No hace falta listar los tests).
d) Escriba tests para satisfacer el criterio Pair-Wise Coverage.

Ejercicio 5. Derive tests para la clase `partition.practico.ejercicio5.BoundedQueue.java` usando la técnica de particionado del espacio de entradas. Considere los siguientes métodos de `BoundedQueue`:

```
public BoundedQueue (int capacity);
public void enqueue (Object X);
public Object dequeue ();
public boolean isEmpty ();
public boolean isFull ();
```

Provea tests para satisfacer el criterio Base Choice Coverage. Implemente los tests resultantes en JUnit.

Ejercicio 6. Considere el problema de buscar un patrón en un String. Una posible implementación (con su correspondiente especificación) se encuentra en la clase `partition.practico.ejercicio6.PatternIndex.java`. Usando la técnica de particionado del espacio de entradas y el criterio Pair-Wise Coverage derive tests para la

implementación de `PatternIndex`. Implemente los tests en JUnit. ¿Descubrieron sus tests fallas en la implementación?