Ejercicios Tema 5. Diseño y realización de pruebas

Bloque 1. Rompe el programa

Observa el siguiente método:

```
public static int calcularPrecioFinal(int precioBase, int descuento) {
   return precioBase - (precioBase * descuento / 100);
}
```

Ejercicio 1

Diseña al menos 7 casos de prueba distintos para el método `calcularPrecioFinal`

Para cada uno indica:

- Entrada
- Resultado esperado
- Oráculo

Caso	Descripción	Entrada (Precio, Desc)	Resultado Esperado	Oráculo (Lógica)
1	Caso estándar (feliz)	`100, 20`	`80`	$100 - 20\%$ es 80.
2	Descuento cero	`50, 0`	`50`	El precio no debe cambiar.
3	Descuento total	`200, 100`	`0`	El producto sale gratis.
4	Límite: Precio 0	`0, 50`	`0`	50% de 0 sigue siendo 0.
5	Truncamiento entero	`10, 33`	`7`	Matemáticamente es 6.7, pero en Java `int` 330/100 = 3. $10-3=7$.
6	Límite: Descuento negativo	`100, -10`	`110`	(Lógica actual): Menos por menos es más. Sube el precio.
7	Límite: Descuento > 100	`100, 150`	`-50`	(Lógica actual): El precio final es negativo.

Ejercicio 2

Contesta las siguientes preguntas relacionadas con el método calcularPrecioFinal

- ¿Qué ocurre si descuento es negativo?

El precio final aumenta en lugar de disminuir. Al restar un número negativo, se suma (Matemática: $100 - (-10) = 110$). ¿Y si es mayor que 100?

- ¿Y si es mayor?

El precio final resulta negativo. Significaría que la tienda le debe dinero al cliente por llevarse el producto.

- ¿Crees que el método debería permitir valores negativos o mayores que 100?

No. Desde el punto de vista de negocio y seguridad, un descuento no debería ser negativo (a menos que sea un recargo explícito) ni mayor al 100% (a menos que sea una promoción especial de "te pagamos", lo cual es muy raro). El método debería validar las entradas.

Ejercicio 3

Propón una mejora del método calcularPrecioFinal que haga que su comportamiento sea más seguro.

Para hacerlo más seguro, aplicamos una cláusula de guarda ("Fail Fast") para validar los parámetros antes del cálculo.

```java
package com.joanmalonda.tema4gradle;

public class Main {
    static void main() {

    }

    public static int calcularPrecioFinal(int precioBase, int descuento) {
        // Validación: El precio no puede ser negativo
        if (precioBase < 0) {
            throw new IllegalArgumentException("El precio base no puede ser negativo");
        }
        // Validación: El descuento debe estar entre 0 y 100
        if (descuento < 0 || descuento > 100) {
            throw new IllegalArgumentException("El descuento debe estar entre 0 y 100");
        }

        return precioBase - (precioBase * descuento / 100);
    }
}
```

Bloque 2. El cazador de bugs

Observa el siguiente método:

```java
public static int maximo(int[] datos) {
  int max = 0;
```

```
    for (int i = 0; i < datos.length; i++) {
        if (datos[i] > max) {
            max = datos[i];
        }
    }
    return max;
}
```

Ejercicio 4

Encuentra al menos 3 entradas de datos donde el método maximo falle.

Para cada entrada indica:

- Resultado obtenido
- Resultado correcto
- Tipo de fallo

Entrada datos[]	Resultado Obtenido	Resultado Correcto	Tipo de Fallo
`{-5, -10, -20}`	`0`	`-5`	**Fallo de inicialización:** El 0 (valor inicial) es mayor que todos los datos, pero no está en el array.
`{-100}`	`0`	`-100`	**Fallo de inicialización:** Mismo caso con un solo elemento negativo.
{} (Array vacío)	`0`	`Excepción` o `null`	**Fallo de diseño:** Retorna 0 como si fuera un valor válido, cuando no hay máximo en un conjunto vacío.

Ejercicio 5

Escribe una versión correcta del método maximo

```java
public static int maximo(int[] datos) {  no usages  new *
    // Manejo de caso borde: Array vacío o nulo
    if (datos == null || datos.length == 0) {
        throw new IllegalArgumentException("El array no puede estar vacío");
    }

    // Inicializamos con el primer elemento real del array
    int max = datos[0];

    // Empezamos el bucle en 1 porque ya tenemos el 0
    for (int i = 1; i < datos.length; i++) {
        if (datos[i] > max) {
            max = datos[i];
        }
    }
    return max;
}
```

Bloque 3. Diseña el oráculo

Observa el siguiente método:

```
public static int[] ordenar(int[] datos) {
  // algoritmo desconocido
}
```

Ejercicio 6

Define 3 propiedades que siempre debe cumplir el resultado del método ordenar.

- Conservación de longitud: resultado.length == entrada.length.
- Integridad de los datos (Permutación): El array de salida debe contener exactamente los mismos números que el de entrada (mismas cantidades de cada número), solo que en diferente posición.
- Orden ascendente: Para todo i tal que 0 <= i < longitud - 1, se debe cumplir que resultado[i] <= resultado[i+1].

Ejercicio 7

Explica por qué en el método ordenar es mejor usar oráculos por propiedades que valores concretos.

Es mejor usar propiedades porque:

- Cobertura: Permite probar con arrays generados aleatoriamente (fuzz testing). Si tuvieras que escribir el "resultado esperado" exacto para un array aleatorio de 1.000 números, tardarías horas

calculándolo a mano.

- Robustez: Al verificar propiedades (como "está ordenado"), no necesitas saber cuál es el resultado exacto dato por dato, sino que el resultado cumple las reglas del universo lógico del problema.

Bloque 4. Caminos y decisiones

A la vista del siguiente método:

```
public static String clasificarEdad(int edad) {
   if (edad < 0) {
      return "ERROR";
   } else if (edad < 12) {
      return "NIÑO";
   } else if (edad < 18) {
      return "ADOLESCENTE";
   } else {
      return "ADULTO";
   }
}
```

Ejercicio 8

Calcula el número mínimo de tests necesarios para el método clasificarEdad. Después propón tantos tests como hayas obtenido.

Número mínimo de tests: 4

Propuesta de tests para cubrir el 100% de los caminos:

- Camino ERROR: Entrada -5 (Cualquier valor < 0).
- Camino NIÑO: Entrada 10 (Cualquier valor >= 0 y < 12).
- Camino ADOLESCENTE: Entrada 15 (Cualquier valor >= 12 y < 18).
- Camino ADULTO: Entrada 30 (Cualquier valor >= 18).

Bloque 6. Prioriza como un profesional

Se quiere desarrollar una aplicación Web de compra online.

Ejercicio 10

Ordena de mayor a menor prioridad:

- Login
- Mostrar catálogo

- Pago con tarjeta
- Cambiar avatar
- Recuperar contraseña

1. Mostrar catálogo
2. Pago con tarjeta
3. Login
4. Recuperar contraseña
5. Cambiar avatar