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

CASOS	ENTRADA	RESULTADO ESPERADO	ORACULO	JUSTIFICACION
1	(100,20)	80	cálculo manual: 100 - (100 *0,2)	Caso Estandar
2	(100, 0)	100	Valor neutro: Un descuento de 0% no debe alterar el precio original.	Límite inferior del descuento.
3	(100, 100)	0	Límite máximo: Un descuento del 100% hace que el producto sea gratuito.	Límite superior del descuento.
4	(100, -10)	Error	Validación de dominio: No se permiten porcentajes de descuento negativos.	Valor negativo (entrada inválida).
5	(100, 150)	Error	Regla de negocio: El descuento no puede ser superior al 100% del valor.	Descuento superior al 100%.
6	(0, 20)	0	Elemento absorbente: Si el precio base es 0, cualquier descuento resulta en 0.	Límite inferior del precio base.
7	(100, 50)	50	Cálculo manual: $100 - (100 \times 0.50)$. Verificación de punto medio.	Descuento típico del 50%

Ejercicio 2

Contesta las siguientes preguntas relacionadas con el método calcularPrecioFinal

- ¿Qué ocurre si descuento es negativo?

Nos da error, ya que no se puede sacar descuento de un numero negativo

- ¿Y si es mayor que 100?

Nos da error porque no podemos aplicar un descuento mayor a la totalidad del precio

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

No. Porque no tendria ningun sentido sacar un descuento imposible.

Ejercicio 3

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

```
    public static int calcularPrecioFinal(int precioBase, int descuento ){  1 usage  new *
        if (precioBase < 0 || descuento < 0 || descuento > 100) {
            throw new IllegalArgumentException("Valores de entrada fuera de rango");
        }
        return precioBase - (precioBase * descuento / 100);
    }
}
```

Bloque 2. El cazador de bugs

Observa el siguiente método:

```
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

CASO	ENTRADA	RESULTADO OBTENIDO	RESULTADO CORRECTO	FALLO
1	[-5, -10, -2]	0	-2	Inicializacion erronea
2	[]	0	Error	No trabaja con vacios

3	[-1]	0+	-1	Logica

Ejercicio 5

Escribe una versión correcta del método maximo

```
    public static int maximo(int[] datos) {  1 usage   new *
        if (datos == null || datos.length == 0) {
            throw new IllegalArgumentException("El array no puede estar vacío");
        }
        int max = datos[0]; // Inicialización correcta
        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.

Orden: Para todo se utiliza la i, resultado[i] <= resultado[i+1]

Permutación: El resultado debe tener los mismo elementos que la entrada

Tamaño: La longitud de la array que quede, debe ser igual a la que entre

Ejercicio 7

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

Es mejor porque permite verificar la corrección sin conocer el resultado exacto de antemano. Si probamos con un array de 10,000 números aleatorios, es imposible que un humano escriba el resultado a mano, pero es muy fácil que un programa verifique si el array final está ordenado y tiene los mismos elementos.

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.

Test 1: Entrada = -5. Por lo que al ser menor que 0 devuelve error.

Test 2: Entrada = 5. Por lo que al ser menor que 12 y mayor a 0 devuelve niño.

Test 3: Entrada = 15. Devuelve adolescente.

Test 4: Entrada = 18. Devuelve adulto.

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

Pago con tarjeta: Sin esto, no hay negocio.

Mostrar catálogo: El usuario necesita ver qué comprar antes de cualquier otra cosa.

Login: Necesario para gestionar pedidos y seguridad, aunque algunos catálogos son públicos.

Recuperar contraseña: Vital para no perder clientes, pero secundaria al flujo de compra principal.

Cambiar avatar: Es cosmético. Si falla, la tienda sigue vendiendo.