Práctica Sesión 18
Testing avanzado en Java (IntelliJ + Maven + JUnit)

Mini-sistema de pedidos en Java que permite:

1. Crear productos con validación básica.
2. Crear pedidos que contienen productos.
3. Guardar pedidos en un repositorio en memoria.
4. Calcular el total aplicando IVA y gastos de envío.

Después tendrás que crear pruebas que demuestren:

- **Pruebas de integración:** comprobar que varias clases funcionan juntas
- **Pruebas de regresión:** comprobar que un cambio incorrecto rompe una funcionalidad y que los tests lo detectan.
- **Pruebas no funcionales:**
  - rendimiento (que el cálculo sea rápido)
  - validación de entradas (seguridad básica)

PROYECTO PASO A PASO EN INTELLIJ

Crear proyecto (IntelliJ)

1. **File → New → Project**
2. Elige **Maven**
3. Name: Sesion18_Pedidos
4. JDK: el que tengáis instalado (recomendado 17 o 21)
5. Finish

POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>Sesion18_Pedidos</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>23</maven.compiler.source>
    <maven.compiler.target>23</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
```

```
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.10.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.2.5</version>
      </plugin>
    </plugins>
  </build>

</project>
```

No te olvides de Sincronizar el proyecto.

Crear paquetes

- En src/main/java crea paquete: **app**
- En src/test/java crea paquete: **test**

CÓDIGO (src/main/java/app)

Producto.java

Qué hace

Representa un producto con:

- nombre
- precio

Incluye **validación**:

- nombre no vacío
- precio > 0

```
package app;

public class Producto {
  private final String nombre;
  private final double precio;

  public Producto(String nombre, double precio) {
    if (nombre == null || nombre.isBlank()) {
```

```
            throw new IllegalArgumentException("Nombre inválido");
        }
        if (precio <= 0) {
            throw new IllegalArgumentException("Precio inválido");
        }
        this.nombre = nombre;
        this.precio = precio;
    }

    public String getNombre() { return nombre; }
    public double getPrecio() { return precio; }
}
```

Esto es seguridad básica: validar entradas para evitar valores inválidos.

Pedido.java

Qué hace

Un pedido es una lista de productos.

- permite añadir productos
- permite obtener la lista de productos (solo lectura)

```
package app;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Pedido {
    private final List<Producto> productos = new ArrayList<>();

    public void addProducto(Producto p) {
        if (p == null) throw new IllegalArgumentException("Producto null");
        productos.add(p);
    }

    public List<Producto> getProductos() {
        return Collections.unmodifiableList(productos);
    }
}
```

unmodifiableList evita que alguien cambie la lista por fuera (buena práctica).

PedidoRepository.java

Qué hace

- guardar pedidos
- listar pedidos

```
package app;

import java.util.List;

public interface PedidoRepository {
   void guardar(Pedido pedido);
   List<Pedido> listar();
}
```

Esto permite cambiar repositorio en el futuro (memoria / fichero / BBDD) sin tocar el servicio.

InMemoryPedidoRepository.java

Qué hace
Implementa el repositorio en memoria usando una lista.

```
package app;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class InMemoryPedidoRepository implements PedidoRepository {
   private final List<Pedido> pedidos = new ArrayList<>();

   @Override
   public void guardar(Pedido pedido) {
      if (pedido == null) throw new IllegalArgumentException("Pedido null");
      pedidos.add(pedido);
   }

   @Override
   public List<Pedido> listar() {
      return Collections.unmodifiableList(pedidos);
   }
}
```

Sirve para pruebas reales sin base de datos.

PedidoService.java

Qué hace

Es la lógica del negocio:

- calcula total con IVA y envío
- valida que el pedido no esté vacío
- guarda el pedido en el repositorio

```
package app;

public class PedidoService {
   private final PedidoRepository repo;
   private static final double IVA = 0.21;
   private static final double ENVIO = 5.0;

   public PedidoService(PedidoRepository repo) {
      this.repo = repo;
   }

   public double calcularTotal(Pedido pedido) {
      double subtotal = 0;
      for (Producto p : pedido.getProductos()) {
         subtotal += p.getPrecio();
      }
      return subtotal * (1 + IVA) + ENVIO;
   }

   public double crearYGuardarPedido(Pedido pedido) {
      if (pedido.getProductos().isEmpty()) {
         throw new IllegalArgumentException("Pedido vacío");
      }
      repo.guardar(pedido);
      return calcularTotal(pedido);
   }
}
```

- cálculo (funcional)
- validación (seguridad básica)
- integración con repositorio

TESTS (src/test/java/test)

ProductoTest.java (UNITARIO)

Prueba una clase sola: Producto.

```
package test;

import app.Producto;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class ProductoTest {

   @Test
   void noPermiteNombreVacio() {
      assertThrows(IllegalArgumentException.class, () -> new Producto("", 10));
   }

   @Test
   void noPermitePrecioNegativo() {
      assertThrows(IllegalArgumentException.class, () -> new Producto("Pan", -1));
   }

   @Test
   void creaProductoCorrecto() {
      Producto p = new Producto("Pan", 1.5);
      assertEquals("Pan", p.getNombre());
      assertEquals(1.5, p.getPrecio());
   }
}
```

PedidoIntegracionTest.java (INTEGRACIÓN)

Comprueba el flujo completo:
Producto → Pedido → Service → Repository

```
package test;

import app.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PedidoIntegracionTest {

   @Test
```

```
    void flujoCompleto_crearCalcularGuardar_listar() {
        PedidoRepository repo = new InMemoryPedidoRepository();
        PedidoService service = new PedidoService(repo);

        Pedido pedido = new Pedido();
        pedido.addProducto(new Producto("Teclado", 20));
        pedido.addProducto(new Producto("Raton", 10));

        double total = service.crearYGuardarPedido(pedido);

        assertTrue(total > 0);
        assertEquals(1, repo.listar().size());
        assertEquals(2, repo.listar().get(0).getProductos().size());
    }
}
```

Esto es integración porque prueba la colaboración entre módulos.

RegresionTest.java (REGRESIÓN)

Asegura que un total concreto no cambia con el tiempo.

```
package test;

import app.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class RegresionTest {

    @Test
    void totalDe30euros_debeSer_41_3() {
        PedidoService service = new PedidoService(new InMemoryPedidoRepository());

        Pedido pedido = new Pedido();
        pedido.addProducto(new Producto("A", 20));
        pedido.addProducto(new Producto("B", 10));

        double total = service.crearYGuardarPedido(pedido);

        // 30 * 1.21 + 5 = 41.3
        assertEquals(41.3, total, 0.0001);
    }
}
```

Cambia en PedidoService el IVA a 0.10 y ejecuta tests:

- ¿Falla el test?

PerformanceTest.java (NO FUNCIONAL: rendimiento)

Mide tiempo aproximado de cálculo.

```
package test;

import app.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PerformanceTest {

   @Test
   void calcularTotal_deberiaSerRapido() {
      PedidoService service = new PedidoService(new InMemoryPedidoRepository());

      Pedido pedido = new Pedido();
      for (int i = 0; i < 1000; i++) {
         pedido.addProducto(new Producto("P" + i, 1));
      }

      long inicio = System.nanoTime();
      double total = service.calcularTotal(pedido);
      long fin = System.nanoTime();

      long tiempoMs = (fin - inicio) / 1_000_000;

      assertTrue(total > 0);
      assertTrue(tiempoMs < 50, "Demasiado lento: " + tiempoMs + " ms");
   }
}
```

Si algún PC es lento, sube el umbral a 100 ms.

SeguridadBasicaTest.java (NO FUNCIONAL: “seguridad” por validación)

Evita estados inválidos.

```
package test;

import app.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class SeguridadBasicaTest {

   @Test
   void noPermitePedidoVacio() {
      PedidoService service = new PedidoService(new InMemoryPedidoRepository());
      Pedido pedido = new Pedido();

      assertThrows(IllegalArgumentException.class, () ->
service.crearYGuardarPedido(pedido));
   }

   @Test
   void noPermiteProductoConNombreEnBlanco() {
      assertThrows(IllegalArgumentException.class, () -> new Producto("   ", 10));
   }
}
```

CÓMO EJECUTAR TODO EN INTELLIJ

1. Clic derecho sobre src/test/java
2. **Run ‘All Tests’** o en los play junto al número de línea.

Resumen

- **Integración:** las clases deben funcionar juntas.
- **Regresión:** un cambio puede romper lo anterior y los tests lo detectan.
- **No funcionales:** no vale solo funciona, debe ser rápido y robusto.