

# Laboratorio 1. Polimorfismo en Java, Control de Dependencias y Pruebas Unitarias

## Objetivo

Comprender el concepto de **polimorfismo** en Java mediante la implementación de un **sistema de procesamiento de pagos** utilizando **Java, Maven** y **JUnit**:

- Definir una abstracción común para diferentes tipos de pagos y programar algunas implementaciones.
- Crear y ejecutar pruebas unitarias con JUnit para validar el comportamiento de las clases de dominio.
- Desarrollar proyectos separados y gestionarlos mediante dependencias en Maven.

#### Marco Contextual

Leer el documento Polimorfismo, Tipado y Programación Orientada a Objetos

Leer el documento Pruebas Unitarias con JUnit

### Contexto del Problema

En el mundo del desarrollo de software comercial, es común encontrar aplicaciones que requieren manejar múltiples tipos de pagos. Cada tipo de pago puede tener reglas y procesos únicos, pero todos deben seguir una estructura común para facilitar su integración y mantenimiento.

En este laboratorio, implementaremos un sistema de procesamiento de pagos que soporta:

- Pago con Tarjeta de Crédito
- Pago por Transferencia Bancaria
- Pago con Criptomonedas

Utilizaremos el concepto de polimorfismo para definir una interfaz común (Pago) y varias clases concretas que representen los diferentes tipos de pagos.

Adicionalmente, aprenderemos a utilizar **Maven** para organizar nuestros proyectos en módulos separados y a realizar pruebas unitarias con **JUnit** para garantizar la calidad del código.



# 1.1 Descripción del Sistema

#### 1.1.1 Interfaz Pago

La interfaz Pago define tres métodos básicos que deben implementar todos los tipos de pago:

```
public interface Pago {
  public boolean validar();
  public void procesar();
  public String obtenerDetalle();
}
```

Cada clase que implemente esta interfaz deberá proporcionar su propia lógica para estos métodos.

## 1.1.2 Clases Concretas

- 1. **PagoTarjetaCredito**: Verifica el número de tarjeta y simula el cobro a través de una pasarela de pagos (Se simula con otra clase).
- 2. **PagoTransferenciaBancaria**: Verifica el número de cuenta bancaria y simula la transferencia de fondos.
- 3. **PagoCriptomoneda**: Valida la dirección de la billetera y confirma el pago en la blockchain (se simula blockchain).

# 1.2 Estructura del Proyecto con Maven

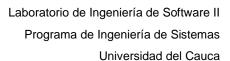
Crearemos tres proyectos Maven:

- 1. p-common: Contiene la interfaz Pago
- p-plug-in: proyecto que depende en tiempo de compilación de p-common y contiene las implementaciones concretas (PagoTarjetaCredito, PagoTransferenciaBancaria, PagoCriptomoneda).
- 3. p-core: Proyecto que depende en tiempo de compilación de p-common y contiene la lógica de la aplicación principal para procesar diferentes tipos de pagos. En tiempo de ejecución depende de los plug-ins que sean necesarios.

# 1.2.1 Configuración de Maven

## Proyecto p-common:

- o Grupo: co.edu.unicauca.pagos
- o Artefacto: p-common





o Versión: 1.0-SNAPSHOT

Proyecto p-core:

o Grupo: edu.co.unicauca.pagos

Artefacto: p-coreVersión: 1.0-SNAPSHOT

Proyecto p-plugin-bch:

Grupo: edu.co.unicauca.pagos
 Artefacto: p-plugin-bch
 Versión: 1.0-SNAPSHOT

#### 1.3 Tareas del Laboratorio

- 1. Crear los tres proyectos y distribuir las responsabilidades:
  - o Definir la interfaz Pago.
  - o Implementar las clases PagoTarjetaCredito, PagoTransferenciaBancaria y PagoCriptomoneda.
  - o Implementar una clase principal que permita simular el procesamiento de diferentes tipos de pagos.
  - o Configurar las dependencias hacia p-common y hacia el plug-in en el respectivo archivo pom.xml.
  - o Mostrar el resultado de la validación y procesamiento de cada pago.

# 2. Implementar Pruebas Unitarias con JUnit

- o Escribir pruebas unitarias para cada clase concreta de cada plug-in.
- Asegurar que las pruebas validen correctamente el comportamiento de los métodos validar(), procesar(), y obtenerDetalle().

# 3. Documentación y Entrega

- o Subir el código a un repositorio GitHub y entregar el link
- o Asegurar que las pruebas unitarias se ejecuten correctamente.
- Documentar el proceso en el archivo README.md, explicando cómo compilar y ejecutar el proyecto.

## 1.4 Ejemplo de Código (Tomarlo de punto de partida)

## 1.4.1 Clase PagoTarjetaCredito

```
public class PagoTarjetaCredito implements Pago {
   private String numeroTarjeta;
   private double monto;

public PagoTarjetaCredito(String numero, double monto) {
    this.numeroTarjeta = numeroTarjeta;
    this.monto = monto;
}
```



@Override

```
public boolean validar() {
    return numeroTarjeta != null && numeroTarjeta.length() == 16;
  @Override
  public void procesar() {
    if (validar()) {
      System.out.println("Procesando pago con tarjeta de crédito por: " + monto);
      System.out.println("Número de tarjeta inválido.");
    }
  }
  @Override
  public String obtenerDetalle() {
    return "Pago con tarjeta de crédito - Monto: " + monto;
  }
}
        Prueba Unitaria con JUnit
1.4.2
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class PagoTarjetaCreditoTest {
  public void testValidarNumeroTarjetaValido() {
    PagoTarjetaCredito pago = new PagoTarjetaCredito("1234567812345678", 100.0);
    assertTrue(pago.validar());
  }
  @Test
  public void testProcesarPagoInvalido() {
    PagoTarjetaCredito pago = new PagoTarjetaCredito("123", 100.0);
    assertFalse(pago.validar());
  }
```

# 1.5 Condiciones de Entrega

}

- Repositorio GitHub: El código debe ser subido a un repositorio público o privado (con acceso compartido).
- **Pruebas Unitarias**: Todas las clases de servicio de pago deben tener pruebas unitarias completas.



Laboratorio de Ingeniería de Software II Programa de Ingeniería de Sistemas Universidad del Cauca

del C	auca	Oniversidad der Gadea
•	<b>Documentación</b> : El repositorio debe incluir un archivo README.md co para compilar, ejecutar y probar el proyecto.	n instrucciones claras