

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

**NOMBRES ESTUDIANTES:** Sebastián Donoso

Ismael Toala

**FECHA:** 16-06-2025

**TEMA:** Examen 1 Bimestre

---

### PoliCampus

#### Introducción:

PoliCampus es una plataforma web desarrollada para la Escuela Politécnica Nacional con el fin de mejorar la experiencia universitaria mediante el acceso centralizado a información clave. El sistema integra servicios como cafetería, transporte, asociaciones y un mercado estudiantil digital, facilitando la vida académica y extracurricular del estudiante.

#### Objetivos:

- Mejorar la comunicación institucional con los estudiantes.
- Optimizar el tiempo mediante acceso rápido a servicios universitarios.
- Fomentar la participación estudiantil y el sentido de comunidad.
- Facilitar el intercambio económico entre estudiantes mediante PoliMarket.

### 1. Entregables del Proyecto

#### 1.1 Estrategia de Ramificación (Branching Strategy)

**Tipo de estrategia:** GitFlow

GitFlow es una estrategia robusta para el versionamiento y organización del desarrollo colaborativo en proyectos que utilizan Git. Permite aislar el desarrollo de nuevas funcionalidades (feature/) respecto a la rama de integración (develop) y producción (main). También contempla ramas específicas para correcciones urgentes (hotfix/) o preparación de releases (release/).

**Estructura de ramas:**

Rama	Propósito
main	Código estable en producción
develop	Integración de nuevas funcionalidades
feature/*	Desarrollo de funcionalidades (1 por HU)

release/*	Preparación de versiones estables
hotfix/*	Correcciones críticas en producción
bugfix/*	Correcciones menores en desarrollo

- Todo feature/\* debe hacer un **Pull Request (PR)** hacia develop tras ser revisado y aprobado por al menos un miembro del equipo técnico.
- Al finalizar un sprint, se crea una rama release/x.y.z desde develop. Tras ajustes y pruebas, esta rama se fusiona en main y en develop.
- En caso de errores en main, se crea un hotfix/\*, se corrige el problema y se fusiona de forma urgente a main y develop.
- Toda rama fusionada debe ser eliminada para mantener un historial limpio (git branch -d nombre-rama).

Rol	Responsabilidades clave en el flujo GitFlow
Developer	Crea ramas feature/*, hace commits, push y PR hacia develop.
Líder Técnico	Revisa PR, gestiona ramas release/*, aprueba merge a main y etiqueta versiones.
QA Tester	Verifica versiones en release/* antes de pasar a producción.
DevOps/CI Engineer	Automatiza el proceso: build, test, checkstyle y despliegue vía Jenkins y Docker.

**Estrategia GitFlow + Comandos GIT - Proyecto PoliCampus (Visual Refinado)**

El diagrama ilustra el flujo de trabajo de desarrollo de software, incluyendo roles como QA Tester, Developer y Líder Técnico, y acciones como crear release, git checkout, merge, push, pull, y parashar bug.

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

### 1.2 Plan de Mantenimiento

Tipo de Mantenimiento	Descripción	Funcionalidades Afectadas	Frecuencia Propuesta	Herramientas / Acciones
<b>Correctivo</b>	Resolución de errores identificados por usuarios o QA.	Todas (bienvenida, cafetería, polibus, poliMarket)	Según reporte (prioridad alta en hotfix)	Issues GitHub, revisión de logs, pruebas JUnit específicas, rama hotfix/*.
<b>Adaptativo</b>	Ajustes por cambios en dependencias o compatibilidad (Java, Maven, navegadores).	Todas, especialmente interfaz y backend.	Mensual o según actualización de dependencias	Actualización de pom.xml, pruebas de regresión, CI con Jenkins.
<b>Perfectivo</b>	Mejoras de rendimiento, experiencia de usuario (UX/UI), limpieza de código.	Cafetería (carga dinámica), PoliMarket (búsqueda y filtros).	Cada 2 sprints (quincenal)	Refactorización con SonarQube, optimización de consultas, lazy loading.
<b>Preventivo</b>	Acciones para evitar futuros errores: pruebas automáticas, refactorización, backup.	Backend (servicios Polibus y base de datos), controladores generales.	Final de cada sprint (semanal)	Aumento de cobertura de pruebas con JUnit, validaciones automáticas en Jenkins.

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

### 1.3 Scripts

**build.yml:**

```
name: Build Java WebApp + Docker

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    services:
      mysql:
        image: mysql:latest
        env:
          MYSQL_ROOT_PASSWORD: 1234
          MYSQL_DATABASE: javaweb
          MYSQL_USER: root
          MYSQL_PASSWORD: 1234
        ports:
          - 3307:3306
        options: >-
          --health-cmd="mysqladmin ping --silent"
          --health-interval=10s
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

--health-timeout=5s

--health-retries=3

steps:

- name: Clonar el repositorio

uses: actions/checkout@v3

- name: Configurar Java

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '21'

- name: Compilar con Maven

run: mvn -B package --file pom.xml

- name: Construir imagen de la app (Tomcat + WAR)

run: docker build -t policampus-webapp -f Webapp.Dockerfile .

- name: Construir imagen de la base de datos

run: docker build -t policampus-db -f Docker-database

- name: Verificar imágenes

run: docker images

Para automatizar el proceso de compilación y construcción de la aplicación Java Web, se implementó un workflow en GitHub Actions denominado Build Java WebApp + Docker. Este flujo se activa automáticamente cada vez que se realiza un push o un pull request a la rama main. El proceso se ejecuta en un entorno Ubuntu y despliega un contenedor de MySQL con una base de datos preconfigurada llamada javaweb, permitiendo simular las condiciones reales de ejecución. A continuación, se clona el repositorio, se configura Java 21 como versión base y se compila el proyecto mediante Maven. Posteriormente, se construyen dos imágenes Docker: una correspondiente a la aplicación web, utilizando el archivo Webapp.Dockerfile, y otra para la base de datos, basada en el archivo Docker-database.

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

Finalmente, se listan las imágenes generadas con el fin de verificar que el proceso se completó correctamente y que el sistema está listo para su ejecución en contenedores.

### Lint.yml:

```
name: Java Lint Checkstyle

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - name: Clonar el repositorio
        uses: actions/checkout@v3

      - name: Configurar Java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '21'

      - name: Instalar dependencias (sin ejecutar tests)
        run: mvn install -DskipTests

      - name: Ejecutar Checkstyle
        run: mvn checkstyle:check
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

También se configuró un segundo workflow denominado Java Lint Checkstyle, cuyo propósito es realizar una revisión automática del estilo del código fuente Java. Este flujo se activa igualmente ante cualquier cambio en la rama main. El proceso comienza clonando el repositorio y configurando Java en su versión 21. Posteriormente, se instalan las dependencias necesarias utilizando Maven, omitiendo la ejecución de pruebas, y finalmente se ejecuta el análisis de estilo mediante el comando `mvn checkstyle:check`. Esta automatización permite garantizar que el código cumple con las reglas de estilo establecidas (por ejemplo, las definidas por Google), asegurando así la calidad y la consistencia del código fuente a lo largo del desarrollo del proyecto.

### 1.4 Scripts/Test

Para garantizar la calidad y el correcto funcionamiento de la aplicación, se desarrollaron pruebas unitarias utilizando los frameworks JUnit y Mockito. En la clase `RutaControllerTest`, se emplea Mockito para simular el comportamiento del controlador de persistencia `RutaJpaController` y los objetos HTTP asociados a la servlet. Esta prueba valida que, al invocar el método `doGet`, se obtienen correctamente las rutas desde la base de datos simulada, se envían como atributo a la vista `polibus.jsp` y, en caso de ocurrir un error, se responde adecuadamente con un código HTTP 500.

Por su parte, las clases `RutaServiceParadaTest` y `RutaServiceTest` implementan pruebas parametrizadas para validar el comportamiento de los métodos de búsqueda dentro del servicio `RutaService`. En `RutaServiceParadaTest`, se verifica que la búsqueda por nombre de parada retorne correctamente todas las rutas que la contienen, contemplando tanto casos con múltiples coincidencias como aquellos sin resultados. En `RutaServiceTest`, se comprueba que la búsqueda por nombre de ruta devuelva la instancia esperada o null cuando no existe coincidencia.

Finalmente, la clase `RutaTest` contiene pruebas unitarias enfocadas en validar el correcto funcionamiento del modelo `Ruta`. Se comprueba el funcionamiento de su constructor, los métodos getters y setters, así como el resultado del método `toString()`. Además, se evalúa la integración con `RutaService` al realizar búsquedas por nombre y por parada, incluyendo escenarios límite como listas vacías o búsquedas sin resultados.

Estas pruebas permiten detectar errores de forma temprana, mantener la integridad de la lógica de negocio y garantizar que futuras modificaciones no afecten funcionalidades previamente validadas.

#### **RutaControllerTest.java:**

```
package com.PoliCampus.controladores;

import static org.mockito.Mockito.*;
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
import com.PoliCampus.modelo.Ruta;
import jakarta.servlet.ServletException;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.RequestDispatcher;
import persistencia.RutaJpaController;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class RutaControllerTest {

    @Mock
    private RutaJpaController mockRutaJpaController;

    @Mock
    private HttpServletRequest request;

    @Mock
    private HttpServletResponse response;
    @Mock
    private RequestDispatcher requestDispatcher;
    private RutaController rutaController;
    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);
        // Inyectar el mock en el controlador
        rutaController = new RutaController(mockRutaJpaController);
    }

    @Test
    public void testDoGetExitoso() throws Exception {
        System.out.println("Test 1: MOCKITO - RutaControllerTest");
        // Crear datos simulados con todos los parámetros del constructor
        List<Ruta> rutasSimuladas = new ArrayList<>();
        rutasSimuladas.add(new Ruta(1L, "Ruta 1", "Parada 1", "10:00 AM", "Teatro",
"https://linkalmapa1.com"));
        rutasSimuladas.add(new Ruta(2L, "Ruta 2", "Parada 2", "11:00 AM", "Sistemas",
"https://linkalmapa2.com"));
```



### CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
// Configurar el mock para devolver los datos simulados
when(mockRutaJpaController.obtenerTodasLasRutas()).thenReturn(rutasSimuladas);
when(request.getRequestDispatcher("polibus.jsp")).thenReturn(requestDispatcher);

// Llamar al método doGet
rutaController.doGet(request, response);

// Verificar que los atributos fueron establecidos correctamente
verify(request).setAttribute("rutas", rutasSimuladas);
verify(requestDispatcher).forward(request, response);

// Verificar que no ocurrió ningún error
verify(response, never()).sendError(anyInt(), anyString());
}

@Test
public void testDoGetConError() throws Exception {
    System.out.println("Test 2: MOCKITO - RutaControllerTest");
    // Simular una excepción en el acceso a la base de datos
    when(mockRutaJpaController.obtenerTodasLasRutas()).thenThrow(new
RuntimeException("Error en la BD"));
    // Llamar al método doGet
    rutaController.doGet(request, response);
    // Verificar que se envió el error correctamente
    verify(response).sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "Error en la
consulta");
}
}
```

#### RutaServiceParadaTest.java:

```
package com.PoliCampus.modelo;
import com.PoliCampus.modelo.Ruta;
import com.PoliCampus.modelo.RutaService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import java.util.Arrays;
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
import java.util.Collection;
import java.util.List;

import static org.junit.Assert.assertEquals;

@RunWith(Parameterized.class)
public class RutaServiceParadaTest {

    private final List<Ruta> rutas;
    private final String nombreParada;
    private final List<Ruta> rutasEsperadas;

    public RutaServiceParadaTest(List<Ruta> rutas, String nombreParada, List<Ruta>
rutasEsperadas) {
        this.rutas = rutas;
        this.nombreParada = nombreParada;
        this.rutasEsperadas = rutasEsperadas;
    }

    @Parameterized.Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {
                // Datos de prueba para el primer caso
                Arrays.asList(
                    new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
                    new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com"),
                ),
                "Parada A",
                Arrays.asList(new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"))
            },
            {
                // Datos de prueba para el segundo caso
                Arrays.asList(
                    new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
                    new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com"),
                ),
                "Parada D",
                Arrays.asList(new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com"))
            }
        })
    }
}
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
    },
    {
        // Datos de prueba para el tercer caso (parada en varias rutas)
        Arrays.asList(
            new Ruta(1L, "Ruta 1", "Parada A, Parada B, Parada E", "7:00 - 19:00", "Bloque
A", "mapa1.com"),
            new Ruta(2L, "Ruta 2", "Parada C, Parada D, Parada E", "8:00 - 20:00", "Bloque
B", "mapa2.com")
        ),
        "Parada E",
        Arrays.asList(
            new Ruta(1L, "Ruta 1", "Parada A, Parada B, Parada E", "7:00 - 19:00", "Bloque
A", "mapa1.com"),
            new Ruta(2L, "Ruta 2", "Parada C, Parada D, Parada E", "8:00 - 20:00", "Bloque
B", "mapa2.com")
        )
    },
    {
        // Datos de prueba para el cuarto caso (parada no encontrada)
        Arrays.asList(
            new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
            new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com")
        ),
        "Parada F",
        Arrays.asList() // Lista vacía
    }
});
}

@Test
public void testBuscarRutaPorParada() {
    RutaService service = new RutaService();
    List<Ruta> resultado = service.buscarRutaPorParada(rutas, nombreParada);

    assertEquals(rutasEsperadas.size(), resultado.size()); // Verificar el tamaño de las listas
    primero

    for (int i = 0; i < rutasEsperadas.size(); i++) {
        Ruta rutaEsperada = rutasEsperadas.get(i);
        Ruta rutaResultado = resultado.get(i);

        // Asegurar que ambos objetos Ruta tengan los mismos valores en sus atributos
        assertEquals(rutaEsperada.getIdBus(), rutaResultado.getIdBus());
    }
}
```

### CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
assertEquals(rutaEsperada.getNombreRuta(), rutaResultado.getNombreRuta());
assertEquals(rutaEsperada.getParadas(), rutaResultado.getParadas());
assertEquals(rutaEsperada.getHorario(), rutaResultado.getHorario());
assertEquals(rutaEsperada.getUbicacion(), rutaResultado.getUbicacion());
assertEquals(rutaEsperada.getMapaUrl(), rutaResultado.getMapaUrl());
    }
}
}
```

#### RutaServiceTest.java:

```
package com.PoliCampus.modelo;
import com.PoliCampus.modelo.Ruta;
import com.PoliCampus.modelo.RutaService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;

import static org.junit.Assert.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;

@RunWith(Parameterized.class)
public class RutaServiceTest {
    private final List<Ruta> rutas;
    private final String nombreRuta;
    private final Ruta rutaEsperada;

    public RutaServiceTest(List<Ruta> rutas, String nombreRuta, Ruta rutaEsperada) {
        this.rutas = rutas;
        this.nombreRuta = nombreRuta;
        this.rutaEsperada = rutaEsperada;
    }

    @Parameterized.Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {
                // Datos de prueba para el primer caso
                Arrays.asList(
                    new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com")
    },
    "Ruta 1",
    new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com")
    },
    {
        // Datos de prueba para el segundo caso
        Arrays.asList(
            new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
            new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com")
        ),
        "Ruta 2",
        new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com")
    },
    {
        // Datos de prueba para el tercer caso (ruta no encontrada)
        Arrays.asList(
            new Ruta(1L, "Ruta 1", "Parada A, Parada B", "7:00 - 19:00", "Bloque A",
"mapa1.com"),
            new Ruta(2L, "Ruta 2", "Parada C, Parada D", "8:00 - 20:00", "Bloque B",
"mapa2.com")
        ),
        "Ruta 3",
        null
    }
    });
}

@Test
public void testBuscarRutaPorNombre() {
    RutaService service = new RutaService();
    Ruta resultado = service.buscarRutaPorNombre(rutas, nombreRuta);

    if (rutaEsperada != null) {
        // Asegurar que ambos objetos Ruta tengan los mismos valores en sus atributos
        assertEquals(rutaEsperada.getIdBus(), resultado.getIdBus());
        assertEquals(rutaEsperada.getNombreRuta(), resultado.getNombreRuta());
        assertEquals(rutaEsperada.getParadas(), resultado.getParadas());
        assertEquals(rutaEsperada.getHorario(), resultado.getHorario());
        assertEquals(rutaEsperada.getUbicacion(), resultado.getUbicacion());
    }
}
```

### CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
        assertEquals(rutaEsperada.getMapaUrl(), resultado.getMapaUrl());
    } else {
        // Verificar que el resultado sea nulo si no se espera encontrar una ruta
        assertNull(resultado);
    }
}
}
```

#### RutaTest.java:

```
package com.PoliCampus.modelo;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.List;

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

public class RutaTest {

    @Test
    public void testConstructor() {
        // Usamos todos los parámetros requeridos por el constructor
        Ruta ruta = new Ruta(1L, "Ruta 1", "Parada 1", "10:00 AM", "Teatro",
"https://linkalmapa1.com");

        // Verificamos que todos los parámetros se hayan asignado correctamente
        assertEquals(1L, ruta.getIdBus());
        assertEquals("Ruta 1", ruta.getNombreRuta());
        assertEquals("Parada 1", ruta.getParadas());
        assertEquals("10:00 AM", ruta.getHorario());
        assertEquals("Teatro", ruta.getUbicacion()); // Verificamos el nuevo campo
        assertEquals("https://linkalmapa1.com", ruta.getMapaUrl()); // Verificamos el nuevo campo
    }

    @Test
    public void testSettersAndGetters() {
        System.out.println("Test 1: Unit Test - RutaTest");
        // Creamos una nueva instancia con el constructor por defecto
        Ruta ruta = new Ruta();

        // Usamos los setters para asignar valores
        ruta.setIdBus(2L);
        ruta.setNombreRuta("Ruta 2");
        ruta.setParadas("Parada 2");
        ruta.setHorario("11:00 AM");
    }
}
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
ruta.setUbicacion("Sistemas");
ruta.setMapaUrl("https://linkalmapa2.com");

// Verificamos que los valores se hayan asignado correctamente
assertEquals(2L, ruta.getIdBus());
assertEquals("Ruta 2", ruta.getNombreRuta());
assertEquals("Parada 2", ruta.getParadas());
assertEquals("11:00 AM", ruta.getHorario());
assertEquals("Sistemas", ruta.getUbicacion()); // Verificamos el nuevo campo
assertEquals("https://linkalmapa2.com", ruta.getMapaUrl()); // Verificamos el nuevo campo
}

@Test
public void testToString() {
    System.out.println("Test 2: Unit Test - RutaTest");
    // Creamos una nueva instancia con todos los parámetros
    Ruta ruta = new Ruta(1L, "Ruta 1", "Parada 1", "10:00 AM", "Teatro",
"https://linkalmapa1.com");

    // Verificamos el resultado del método toString()
    String expected = "Nombre de la Ruta: Ruta 1, Paradas: Parada 1, Horario: 10:00 AM,
Ubicación: Teatro, Mapa URL: https://linkalmapa1.com";
    assertEquals(expected, ruta.toString());
}

@Test
public void testBuscarRutaPorNombre() {
    RutaService rutaService = new RutaService();

    // Creamos una lista de rutas
    List<Ruta> rutas = new ArrayList<>();
    rutas.add(new Ruta(1L, "Ruta 1", "Parada 1", "10:00 AM", "Teatro",
"https://linkalmapa1.com"));
    rutas.add(new Ruta(2L, "Ruta 2", "Parada 2", "11:00 AM", "Sistemas",
"https://linkalmapa2.com"));
    rutas.add(new Ruta(3L, "Ruta 3", "Parada 3", "12:00 PM", "Biblioteca",
"https://linkalmapa3.com"));

    // Buscamos una ruta por su nombre
    Ruta rutaEncontrada = rutaService.buscarRutaPorNombre(rutas, "Ruta 2");

    // Verificamos que la ruta encontrada sea la correcta
    assertNotNull(rutaEncontrada);
    assertEquals(2L, rutaEncontrada.getIdBus());
    assertEquals("Ruta 2", rutaEncontrada.getNombreRuta());
```

## CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
}  
@Test  
public void testBuscarRutaPorNombreNoExistente() {  
    System.out.println("Test 5: Unit Test - RutaTest");  
    RutaService rutaService = new RutaService();  
  
    // Creamos una lista de rutas  
    List<Ruta> rutas = new ArrayList<>();  
    rutas.add(new Ruta(1L, "Ruta 1", "Parada 1", "10:00 AM", "Teatro",  
"https://linkalmapa1.com"));  
    rutas.add(new Ruta(2L, "Ruta 2", "Parada 2", "11:00 AM", "Sistemas",  
"https://linkalmapa2.com"));  
  
    // Buscamos una ruta por un nombre que no existe  
    Ruta rutaEncontrada = rutaService.buscarRutaPorNombre(rutas, "Ruta 3");  
  
    // Verificamos que la ruta encontrada sea nula  
    assertNull(rutaEncontrada);  
}  
  
@Test  
public void testBuscarRutaPorParadaExistente() {  
    System.out.println("Test 6: Unit Test - RutaTest");  
    RutaService rutaService = new RutaService();  
  
    // Creamos una lista de rutas con varias paradas  
    List<Ruta> rutas = new ArrayList<>();  
    rutas.add(new Ruta(1L, "Ruta 1", "Parada A, Parada B, Parada C", "10:00 AM", "Teatro",  
"https://linkalmapa1.com"));  
    rutas.add(new Ruta(2L, "Ruta 2", "Parada B, Parada D", "11:00 AM", "Sistemas",  
"https://linkalmapa2.com"));  
  
    // Buscamos rutas por una parada que existe en ambas rutas  
    List<Ruta> rutasEncontradas = rutaService.buscarRutaPorParada(rutas, "Parada B");  
  
    // Verificamos que se encuentren ambas rutas  
    assertEquals(2, rutasEncontradas.size());  
}  
  
@Test  
public void testBuscarRutaPorParadaNoExistente() {  
    System.out.println("Test 7: Unit Test - RutaTest");  
    RutaService rutaService = new RutaService();  
  
    // Creamos una lista de rutas con varias paradas
```



### CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE (ISWD633)

```
List<Ruta> rutas = new ArrayList<>();
rutas.add(new Ruta(1L, "Ruta 1", "Parada A, Parada B, Parada C", "10:00 AM", "Teatro",
"https://linkalmapa1.com"));
rutas.add(new Ruta(2L, "Ruta 2", "Parada B, Parada D", "11:00 AM", "Sistemas",
"https://linkalmapa2.com"));

// Buscamos rutas por una parada que no existe
List<Ruta> rutasEncontradas = rutaService.buscarRutaPorParada(rutas, "Parada Z");

// Verificamos que la lista de rutas encontradas esté vacía
assertTrue(rutasEncontradas.isEmpty());
}

@Test
public void testBuscarRutaPorParadaConListaVacía() {
    System.out.println("Test 8: Unit Test - RutaTest");
    RutaService rutaService = new RutaService();

    // Creamos una lista de rutas vacía
    List<Ruta> rutas = new ArrayList<>();

    // Buscamos rutas por una parada en una lista vacía
    List<Ruta> rutasEncontradas = rutaService.buscarRutaPorParada(rutas, "Parada A");

    // Verificamos que la lista de rutas encontradas esté vacía
    assertTrue(rutasEncontradas.isEmpty());
}
}
```