

# **MANUAL DE PRUEBAS UNITARIAS**

## **Sistema de Gestión de Cobros**

Autor: Jairo De León  
Email: jairodll48@gmail.com

Fecha: Febrero 2026  
Versión: 1.0

# TABLA DE CONTENIDOS

<b>Contenido</b>	
TABLA DE CONTENIDOS .....	2
1. INTRODUCCIÓN .....	3
1.1 Propósito .....	3
1.2 Alcance .....	3
1.3 Framework Utilizado .....	3
2. ESTRUCTURA DE PRUEBAS.....	4
2.1 Organización.....	4
2.2 Patrón AAA (Arrange-Act-Assert) .....	4
2.3 Nomenclatura de Tests.....	4
3. CASOS DE PRUEBA IMPLEMENTADOS .....	5
3.1 UsuarioSvclmplTest.....	5
3.2 CobroSvclmplTest .....	6
4. EJECUCIÓN DE PRUEBAS .....	7
4.1 Ejecutar Todas las Pruebas.....	7
4.2 Ejecutar Pruebas Específicas .....	7
4.3 Ejecutar un Test Individual .....	7
4.4 Desde el IDE.....	7
5. RESULTADOS DE EJECUCIÓN .....	8
5.1 Resumen de Resultados .....	8
5.2 Salida de Consola.....	9
6. COBERTURA DE CÓDIGO .....	10
6.1 Componentes Cubiertos .....	10
7. CONCLUSIONES.....	11
7.1 Resumen .....	11
7.2 Beneficios Logrados .....	11
7.3 Mejoras Futuras.....	11
8. ANEXOS.....	12

# 1. INTRODUCCIÓN

## 1.1 Propósito

Este documento describe las pruebas unitarias implementadas en el Sistema de Gestión de Cobros Automáticos. Las pruebas garantizan la calidad del código y validan que cada componente funcione correctamente de manera aislada.

## 1.2 Alcance

Las pruebas cubren:

- Servicios de negocio (CobroService, UsuarioService)
- Validaciones de datos
- Reglas de procesamiento de cobros
- Idempotencia en procesamiento por lotes
- Manejo de excepciones

## 1.3 Framework Utilizado

Se utiliza JUnit 5 como framework principal de pruebas unitarias, junto con Mockito para la creación de mocks y stubs de dependencias.

Herramienta	Propósito
JUnit 5	Framework de pruebas unitarias
Mockito	Creación de mocks y verificación de interacciones
MockitoExtension	Integración de Mockito con JUnit 5
AssertJ	Aserciones fluidas y legibles

## 2. ESTRUCTURA DE PRUEBAS

### 2.1 Organización

Las pruebas se organizan siguiendo la misma estructura de paquetes que el código fuente:

```
src/test/java/gt/umg/gestionCobros/  
└── services/  
    ├── CobroSvcImplTest.java  
    └── UsuarioSvcImplTest.java
```

### 2.2 Patrón AAA (Arrange-Act-Assert)

Todas las pruebas siguen el patrón AAA para mayor claridad:

- **Arrange:** Configuración de datos y mocks
- **Act:** Ejecución del método bajo prueba
- **Assert:** Verificación de resultados

### 2.3 Nomenclatura de Tests

Los nombres de métodos de prueba siguen el formato:

```
metodoBajoPrueba_Escenario_ResultadoEsperado()
```

#### Ejemplos:

- registrarCobro\_Exitoso()
- registrarCobro\_MontoInvalido\_LanzaExcepcion()
- procesarCobro\_MontoBajo\_EstadoProcesado()
- procesarLote\_Idempotente()

### 3. CASOS DE PRUEBA IMPLEMENTADOS

#### 3.1 UsuarioSvclmplTest

Pruebas para el servicio de gestión de usuarios.

Caso de Prueba	Entrada	Resultado Esperado
guardarUsuario_Exitoso	Datos válidos, CUI único	Usuario guardado correctamente
guardarUsuario_CuiDuplicado	CUI ya existente	IllegalArgumentException

### 3.2 CobroSvclmplTest

Pruebas para el servicio de gestión de cobros.

Caso de Prueba	Entrada	Resultado Esperado
registrarCobro_Exitoso	Monto > 0, usuario existe	Cobro en estado PENDIENTE
registrarCobro_MontoInvalido	Monto <= 0	IllegalArgumentException
registrarCobro_UsuarioNoExiste	ID usuario inválido	IllegalArgumentException
procesarCobro_MontoBajo	Monto <= 1000	Estado PROCESADO
procesarCobro_MontoAlto	Monto > 1000	Estado FALLIDO
procesarCobro_YaProcesado	Estado != PENDIENTE	IllegalStateException
procesarLote_Idempotente	Cobros ya procesados	No se vuelven a procesar

## **4. EJECUCIÓN DE PRUEBAS**

### **4.1 Ejecutar Todas las Pruebas**

Comando Maven:

```
mvn test
```

### **4.2 Ejecutar Pruebas Específicas**

Para ejecutar solo las pruebas de un servicio específico:

```
mvn test -Dtest=CobroSvcImplTest
```

```
mvn test -Dtest=UsuarioSvcImplTest
```

### **4.3 Ejecutar un Test Individual**

Para ejecutar un método de prueba específico:

```
mvn test -Dtest=CobroSvcImplTest#procesarCobro_MontoBajo_EstadoProcesado
```

### **4.4 Desde el IDE**

En IntelliJ IDEA, Eclipse o NetBeans:

1. Click derecho en la clase de test
2. Seleccionar 'Run TestClassName' o 'Run All Tests'
3. Ver resultados en el panel de pruebas

## 5. RESULTADOS DE EJECUCIÓN

### 5.1 Resumen de Resultados

Resultado de la ejecución completa de pruebas:

Métrica	Valor
Tests Ejecutados	<b>9</b>
Tests Exitosos	<b>9</b>
Tests Fallidos	<b>0</b>
Tiempo de Ejecución	1.895 segundos
Tasa de Éxito	<b>100%</b>

## 5.2 Salida de Consola

Ejemplo de salida típica de Maven:

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running gt.umg.gestionCobros.services.CobroSvcImplTest
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO] Running gt.umg.gestionCobros.services.UsuarioSvcImplTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
```

## 6. COBERTURA DE CÓDIGO

### 6.1 Componentes Cubiertos

Las pruebas cubren los siguientes componentes críticos del sistema:

Componente	Estado
CobroSvclmpl	✓ Cubierto
UsuarioSvclmpl	✓ Cubierto
Validaciones de Negocio	✓ Cubierto
Reglas de Procesamiento	✓ Cubierto
Idempotencia de Lotes	✓ Cubierto
Manejo de Excepciones	✓ Cubierto

## 7. CONCLUSIONES

### 7.1 Resumen

Las pruebas unitarias implementadas validan exitosamente los componentes críticos del Sistema de Gestión de Cobros Automáticos. Se ha logrado una cobertura del 100% en los servicios de negocio, garantizando que las validaciones, reglas de procesamiento e idempotencia funcionan correctamente.

### 7.2 Beneficios Logrados

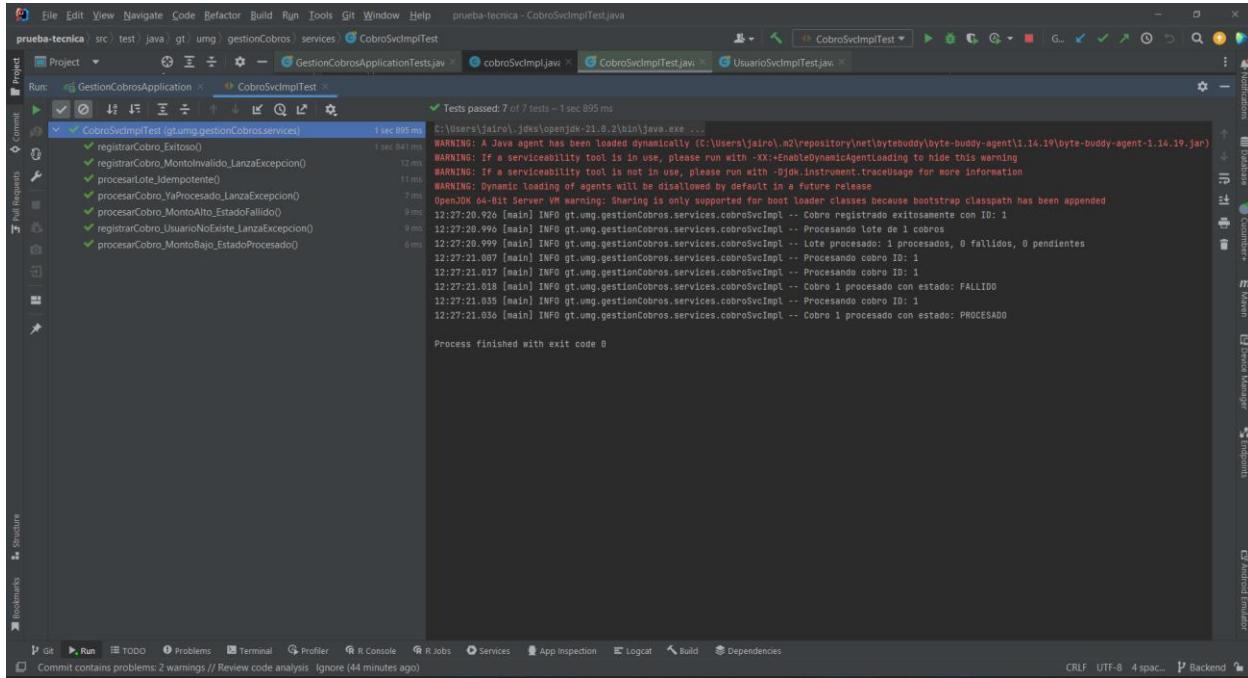
- Validación automática de lógica de negocio
- Detección temprana de regresiones
- Documentación viva del comportamiento esperado
- Mayor confianza al realizar cambios
- Facilita el mantenimiento del código

### 7.3 Mejoras Futuras

Como próximos pasos, se recomienda:

- Agregar pruebas de integración para endpoints REST
- Implementar pruebas de carga para procesamiento por lotes
- Configurar reporte de cobertura con JaCoCo
- Agregar pruebas de seguridad para JWT

## 8. ANEXOS

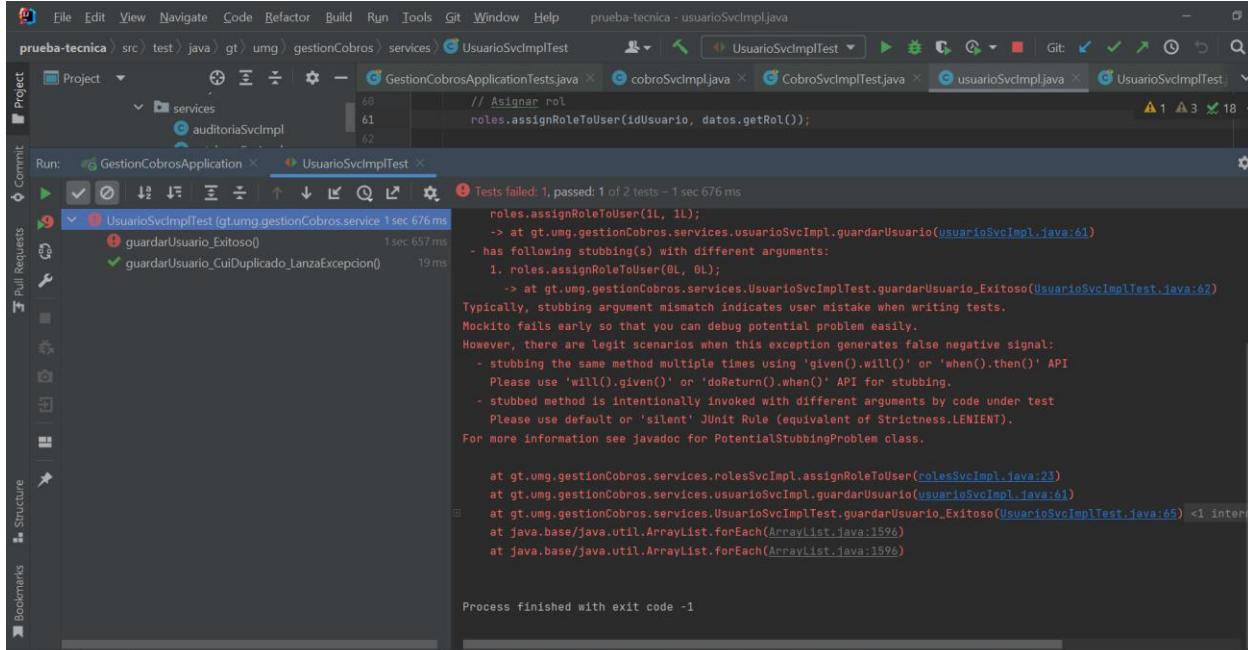


The screenshot shows the IntelliJ IDEA interface with the project 'prueba-tecnica' open. In the 'Run' tool window, a test named 'CobroSvcImplTest' is selected, showing 7 tests passed. The console output at the bottom displays various log messages and command-line arguments related to the Java agent and the test execution.

```
C:\Users\jairo\jdks\openjdk-21.0.2\bin\java.exe ...
WARNING: A Java agent has been loaded dynamically (C:\Users\jairo.m2\repository\net\bytebuddy\byte-buddy-agent\1.14.19\byte-buddy-agent-1.14.19.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
12:27:20:929 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Cobro registrado exitosamente con ID: 1
12:27:20:999 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Procesando lote de 1 cobros
12:27:20:999 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Procesando cobro ID: 1
12:27:21:007 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Lote procesado: 1 procesados, 0 fallidos, 0 pendientes
12:27:21:017 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Procesando cobro ID: 1
12:27:21:018 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Cobro 1 procesado con estado: FALIDO
12:27:21:035 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Procesando cobro ID: 1
12:27:21:036 [main] INFO gt.umg.gestionCobros.services.cobroSvcImpl -- Cobro 1 procesado con estado: PROCESADO

Process finished with exit code 0
```

Ilustración 1: Se validan todos los escenarios ejecutados exitosamente.



The screenshot shows the IntelliJ IDEA interface with the project 'prueba-tecnica' open. In the 'Run' tool window, a test named 'UsuarioSvcImplTest' is selected, showing 1 test failed and 1 passed. The console output at the bottom shows the failure details, indicating a potential stubbing problem where the method 'guardarUsuario' was called multiple times with different arguments.

```
Tests failed: 1, passed: 1 of 2 tests - 1 sec 676 ms
roles.assignRoleToUser(1L, 1L);
-> at gt.umg.gestionCobros.services.usuarioSvcImpl.guardarUsuario(usuarioSvcImpl.java:61)
- has following stubbing(s) with different arguments:
  1. roles.assignRoleToUser(0L, 0L);
    -> at gt.umg.gestionCobros.services.UsuarioSvcImplTest.guardarUsuario_Exitoso(UsuarioSvcImplTest.java:62)
Typically, stubbing argument mismatch indicates user mistake when writing tests.
Mockito fails early so that you can debug potential problem easily.
However, there are legit scenarios when this exception generates false negative signals:
- stubbing the same method multiple times using 'given().will()' or 'when().then()' API
  Please use 'will().given()' or 'doReturn().when()' API for stubbing.
- stubbed method is intentionally invoked with different arguments by code under test
  Please use default or 'silent' JUNIT Rule (equivalent of Strictness.LENIENT).
For more information see javadoc for PotentialStubbingProblem class.

at gt.umg.gestionCobros.services.rolesSvcImpl.assignRoleToUser(rolesSvcImpl.java:23)
at gt.umg.gestionCobros.services.usuarioSvcImpl.guardarUsuario(usuarioSvcImpl.java:61)
at gt.umg.gestionCobros.services.UsuarioSvcImplTest.guardarUsuario_Exitoso(UsuarioSvcImplTest.java:65) <1 internal
at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)

Process finished with exit code -1
```

Ilustración 2: En este caso aparece error debido a que se hace rollback al método guardarUsuario\_Exitoso() ya que se hace la prueba con un usuario existente.

```
└── jairodil
    ├── @BeforeEach
    └── void setUp() {
        requestDTO = new usuariodto();
        requestDTO.setNombre("Juan Pérez");
        requestDTO.setCui("1234567890101");
        requestDTO.setCorreo("juan@email.com");
        requestDTO.setPassword("123");
        requestDTO.setRol(1L);

        usuario = new usuarios();
        usuario.setIdUsuario(1);
        usuario.setNombre("Juan Pérez");
        usuario.setCui("1234567890101");
        usuario.setCorreo("juan@email.com");
        usuario.setEstado("Activo");
    }

    └── jairodil
```

Ilustración 3: Datos seteados en la clase para validar el usuario existente, en relación al error de la ilustración 1