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

Contenido

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 UsuarioSvcImplTest

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 CobroSvcImplTest

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	**9**
Tests Exitosos	**9**
Tests Fallidos	**0**
Tiempo de Ejecución	1.895 segundos
Tasa de Éxito	**100%**

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
CobroSvcImpl	✓ Cubierto
UsuarioSvcImpl	✓ 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

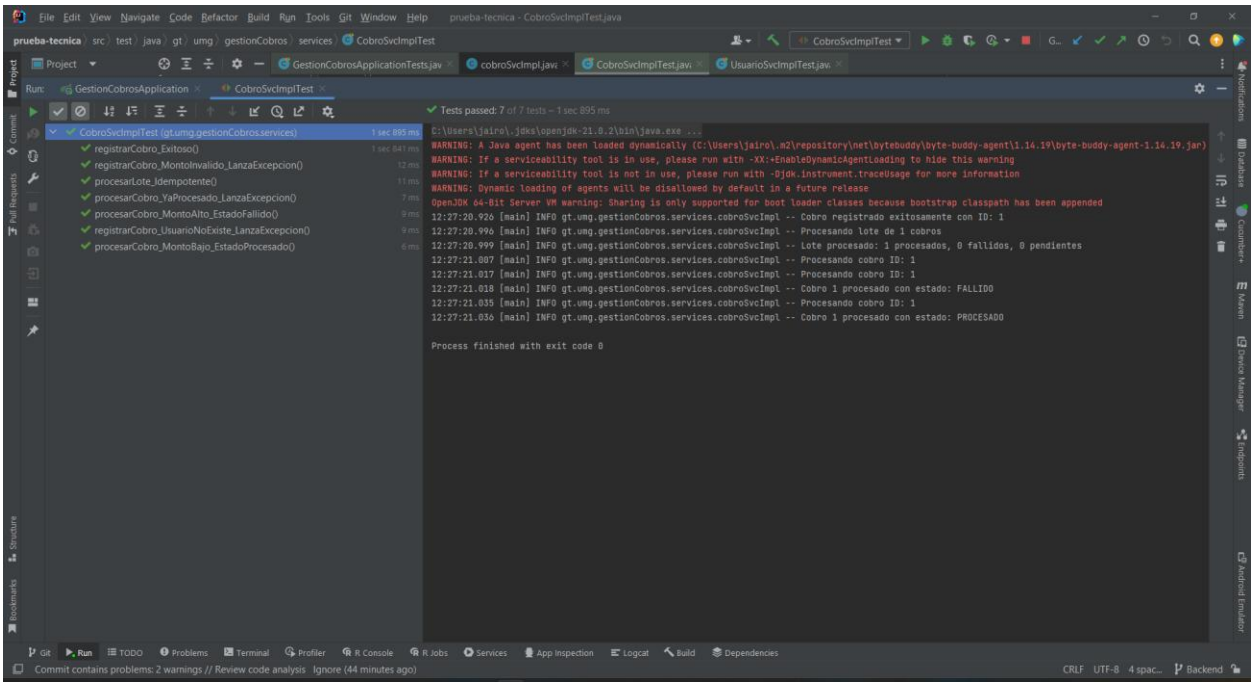

Ilustración 1: Se validan todos los escenarios ejecutados éxitosamente.

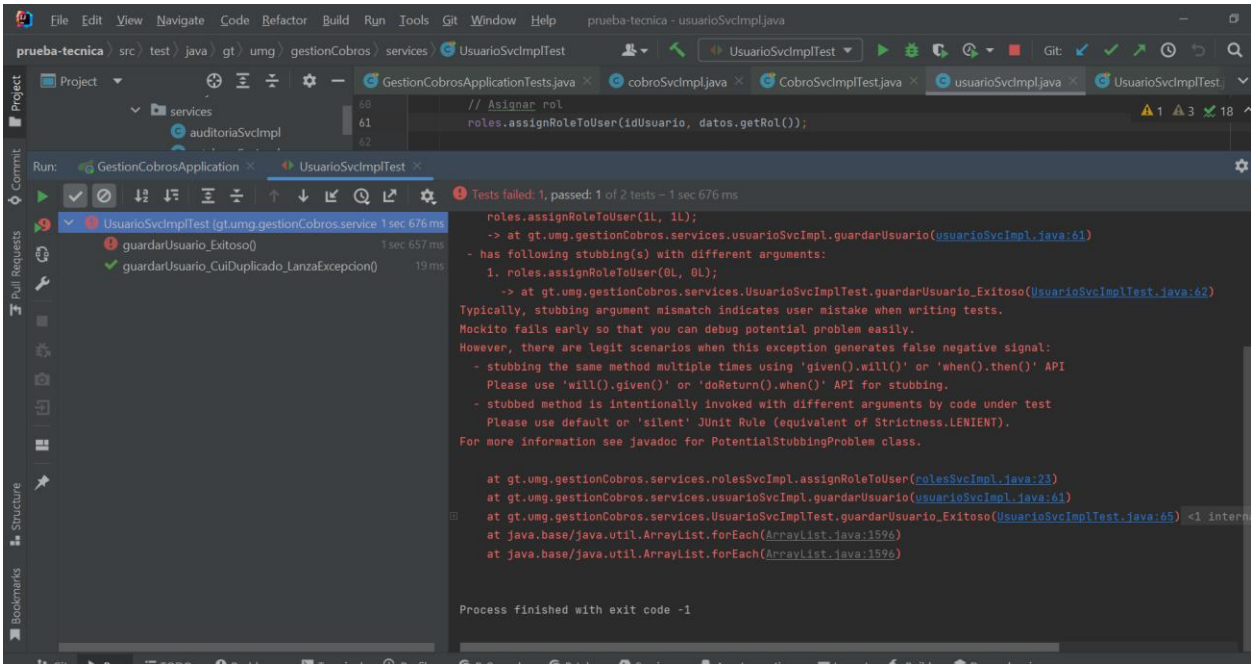

Ilustración 2: En este caso aparece error debido a que se hace rollback al método guardadUsuario_Exitoso() ya que se hace la prueba con un usuario existente.

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