MASCOTAS EXÓTICAS

Documentación de Pruebas Unitarias — Clase ControlMascota

1. Introducción

Para validar el correcto funcionamiento de la lógica implementada en el controlador ControlMascota se desarrollaron pruebas unitarias automatizadas. Las pruebas se implementaron principalmente con JUnit (versión 4) y Mockito para la simulación (mocking) de dependencias, y se ejecutaron en un proyecto basado en Apache Ant.

2. Objetivo

- Verificar que el controlador cumpla su rol dentro del patrón MVC.
- Asegurar que no se registren mascotas duplicadas.
- Garantizar que las operaciones de inserción, modificación y eliminación deleguen correctamente al DAO.
- Comprobar que las excepciones se manejen adecuadamente en condiciones inválidas.
- Validar la delegación de serialización al servicio correspondiente sin acceder a archivos reales.

3. Metodología

Se aplicó aislamiento del componente bajo prueba reemplazando las dependencias externas por mocks de Mockito. De esta forma las pruebas se enfocan únicamente en la lógica de ControlMascota, sin necesidad de base de datos ni archivos.

Ventajas:
- Independencia de infraestructuras externas.
- Ejecución rápida y determinista.
- Mayor cobertura de caminos lógicos en la clase de control.

4. Escenarios de prueba definidos

Los escenarios definidos y probados fueron:

• Caso exitoso: agregar una nueva mascota válida.

• Caso de error: intento de agregar una mascota duplicada (debe lanzar IllegalArgumentException).

• Modificación válida: actualizar mascota existente.

• Modificación inválida: intentar modificar mascota inexistente (lanza excepción).

• Eliminación: eliminar mascota por apodo delegando al DAO.

• Listado: listar todas las mascotas.

• Serialización: delegación al servicio de serialización sin acceso a archivos reales.

5. Esquema de Pruebas JUnit

El proyecto utiliza JUnit 4. A continuación se detallan las anotaciones y su uso en las pruebas:

Anotaciones usadas:
- @Before: inicializa los mocks y la instancia de ControlMascota antes de cada prueba. Garantiza independencia entre tests.
- @After: reservado para limpieza posterior a cada prueba (no se requirió en este caso porque no se abrieron recursos externos).
- @Test: define los métodos de prueba para cada escenario funcional.

Sobre @BeforeAll / @AfterAll:
- @BeforeAll y @AfterAll son anotaciones de JUnit 5 (Jupiter). Su equivalente en JUnit 4 es @BeforeClass y @AfterClass, los cuales deben aplicarse a métodos estáticos. - En este proyecto no se implementaron @BeforeClass ni @AfterClass porque no existió la necesidad de inicializar recursos globales compartidos (por ejemplo, servidores, bases de datos o archivos a nivel de suite de pruebas). Usar @Before por prueba asegura un entorno limpio y aislado en cada ejecución.

Si se migrara a JUnit 5, se podría usar @BeforeAll/@AfterAll sin cambiar la lógica de las pruebas; solo cambiarían las anotaciones y posiblemente la visibilidad/estaticidad donde fuera necesario.

6. Datasets de Prueba

Se definieron datasets fijos y reproducibles para cada prueba, representados por instancias de MascotaVO.

Dataset base (ejemplo):

MascotaVO mascota = new MascotaVO("Luna", "Ave", "Psittacidae", "Hembra", "Loro", "Herbívoro", "Lunita");

Tabla de datasets y resultados esperados:

Escenario	Dataset utilizado	Resultado esperado
Adicionar mascota válida	Luna (Lunita)	true

Adicionar duplicado	Luna (Lunita) ya existe	IllegalArgumentException
Modificar existente	Luna	true
Eliminar por apodo	"Lunita"	true
Listar todas	Lista con Luna	Lista tamaño 1
Serializar sin alimento	Lista vacía	Delegación exitosa
Guardar estado	Lista vacía	Delegación exitosa

7. Integración de las pruebas con Apache Ant

Este proyecto está construido con Apache Ant. A diferencia de Maven, Ant no gestiona automáticamente las dependencias; por tanto, las librerías se agregaron manualmente en la carpeta 'lib/' y se configuraron en el classpath del proyecto.

Librerías necesarias (deben añadirse a lib/ y al classpath de Test y Compile en NetBeans):

- junit-4.x.jar
- mockito-core-3.12.4.jar
- byte-buddy-1.10.22.jar
- objenesis-3.2.jar

Pasos de ejecución en Ant/NetBeans:
1. Colocar los .jar en la carpeta lib/ del proyecto.
2. Añadir esos .jar en Properties > Libraries > Compile y > Test Compile.
3. Ejecutar Clean and Build.
4. Ejecutar las pruebas desde NetBeans (Test File o Test Project) y revisar resultados en la ventana de pruebas.

8. Resultados de ejecución y evidencia

Todas las pruebas definidas se ejecutaron correctamente y resultaron pasadas.

9. Conclusión

Las pruebas unitarias confirmaron que ControlMascota cumple con su responsabilidad única dentro del patrón MVC, que delega correctamente las operaciones al DAO y maneja condiciones de error adecuadamente. La configuración con Ant y la inclusión manual de dependencias fue suficiente para ejecutar un suite de pruebas completa.

10. Anexos

• Clase de prueba: ControlMascotaTest.java (incluye los tests: adicionar, modificar, eliminar, listar, serializar, guardarEstado).

```java
public class ControlMascotaTest {

    private ICRUDMascota mascotaDAOMock;
    private ISerializacionService serializacionMock;
    private ControlMascota controlMascota;

    @Before
    public void setUp() {
        mascotaDAOMock = Mockito.mock(ICRUDMascota.class);
        serializacionMock = Mockito.mock(ISerializacionService.class);
        controlMascota = new ControlMascota(mascotaDAOMock, serializacionMock);
    }

    @Test
    public void testAdicionarMascota_CuandoNoExiste_DeberiaAgregar() {
        MascotaVO mascota = new MascotaVO("Luna", "Ave", "Psittacidae", "Hembra", "Loro", "Herbívoro", "Lunita");
        when(mascotaDAOMock.consultarPorApodo("Lunita")).thenReturn(new ArrayList<>());
        when(mascotaDAOMock.adicionarMascota(mascota)).thenReturn(true);

        boolean resultado = controlMascota.adicionarMascota(mascota);

        assertTrue(resultado);
        verify(mascotaDAOMock).adicionarMascota(mascota);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testAdicionarMascota_CuandoYaExiste_DeberiaLanzarExcepcion() {
        MascotaVO mascota = new MascotaVO("Luna", "Ave", "Psittacidae", "Hembra", "Loro", "Herbívoro", "Lunita");
        List<MascotaVO> existentes = new ArrayList<>();
        existentes.add(mascota);
        when(mascotaDAOMock.consultarPorApodo("Lunita")).thenReturn(existentes);

        controlMascota.adicionarMascota(mascota);
    }
```

```java
    @Test
    public void testModificarMascota_CuandoExiste_DeberiaModificar() {
        MascotaVO mascota = new MascotaVO("Luna", "Ave", "Psittacidae", "Hembra", "Loro", "Herbívoro", "Lunita");
        List<MascotaVO> existentes = new ArrayList<>();
        existentes.add(mascota);

        when(mascotaDAOMock.consultarPorApodo("Lunita")).thenReturn(existentes);
        when(mascotaDAOMock.modificarMascota(mascota)).thenReturn(true);

        boolean resultado = controlMascota.modificarMascota(mascota);

        assertTrue(resultado);
        verify(mascotaDAOMock).modificarMascota(mascota);
    }

    @Test
    public void testEliminarMascota_DeberiaLlamarDAO() {
        when(mascotaDAOMock.eliminarMascota("Lunita")).thenReturn(true);

        boolean resultado = controlMascota.eliminarMascota("Lunita");

        assertTrue(resultado);
        verify(mascotaDAOMock).eliminarMascota("Lunita");
    }

    @Test
    public void testListarTodasMascotas_DeberiaRetornarLista() {
        List<MascotaVO> lista = new ArrayList<>();
        lista.add(new MascotaVO("Luna", "Ave", "Psittacidae", "Hembra", "Loro", "Herbívoro", "Lunita"));
        when(mascotaDAOMock.listarTodasMascotas()).thenReturn(lista);

        List<MascotaVO> resultado = controlMascota.listarTodasMascotas();

        assertEquals(1, resultado.size());
        assertEquals("Luna", resultado.get(0).getNombre());
    }
```

```java
    @Test
    public void testSerializarMascotasSinAlimento_DeberiaLlamarServicio() throws Exception {
        when(mascotaDAOMock.listarTodasMascotas()).thenReturn(new ArrayList<>());
        doNothing().when(serializacionMock).serializarSinAlimento(anyList(), anyString());

        boolean resultado = controlMascota.serializarMascotasSinAlimento("archivo.ser");

        assertTrue(resultado);
        verify(serializacionMock).serializarSinAlimento(anyList(), eq("archivo.ser"));
    }

    @Test
    public void testGuardarEstadoMascotas_DeberiaLlamarServicio() throws Exception {
        when(mascotaDAOMock.listarTodasMascotas()).thenReturn(new ArrayList<>());
        doNothing().when(serializacionMock).guardarEstadoRandomAccess(anyList(), anyString());

        boolean resultado = controlMascota.guardarEstadoMascotas("estado.dat");

        assertTrue(resultado);
        verify(serializacionMock).guardarEstadoRandomAccess(anyList(), eq("estado.dat"));
    }
}
```