Informe de Resultados de Pruebas Unitarias

Informe de Resultados de Pruebas Unitarias – Clase ControlArchivos

Proyecto: FaunaFreaks
Paquete: udistrital.avanzada.parcial.control
Clase probada: ControlArchivos
Clase de prueba: ControlArchivosTest
Autora: Paula Martínez
Fecha de ejecución: 15 de octubre de 2025
Herramienta: Maven con JUnit 5

Resumen general

Total de pruebas	Pruebas exitosas	Fallidas	Con error	Resultado final
5	5	0	0	100% aprobadas

Configuración de las pruebas

En esta clase se usaron las principales anotaciones de JUnit 5 para organizar las pruebas:

- **@BeforeAll:** se ejecuta una sola vez antes de todas las pruebas para crear la instancia de ControlArchivos.
- **@AfterAll:** se ejecuta una vez al final para liberar recursos.
- **@BeforeEach:** prepara los datos antes de cada prueba, creando objetos de tipo MascotaVO.
- **@AfterEach:** limpia el dataset temporal después de cada prueba.
- **@Order:** define el orden en que se ejecutan los tests.

Descripción de las pruebas

1. `testSerializarMascotas()`

- **Objetivo:** comprobar que el método `serializarMascotas()` no lance errores cuando se le pasa una lista válida.
  Resultado: pasó correctamente.
  Explicación: la serialización se ejecutó sin excepciones.

2. **`testDeserializarMascotas()`**

- **Objetivo:** verificar que `deserializarMascotas()` devuelva una lista no nula.
  Resultado: pasó correctamente.
  Explicación: el método devolvió una lista válida, aunque vacía, lo cual cumple con lo esperado.

3. **`testGuardarAleatorio()`**

- **Objetivo:** probar que `guardarAleatorio()` no falle al guardar una lista de mascotas.
  Resultado: pasó correctamente.
  Explicación: el método no lanzó excepciones y el guardado se completó sin errores.

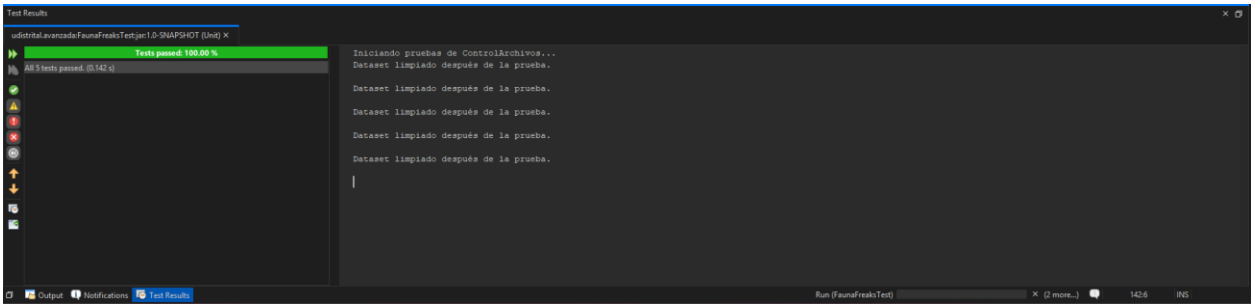
4. **`testCargarAleatorio()`**

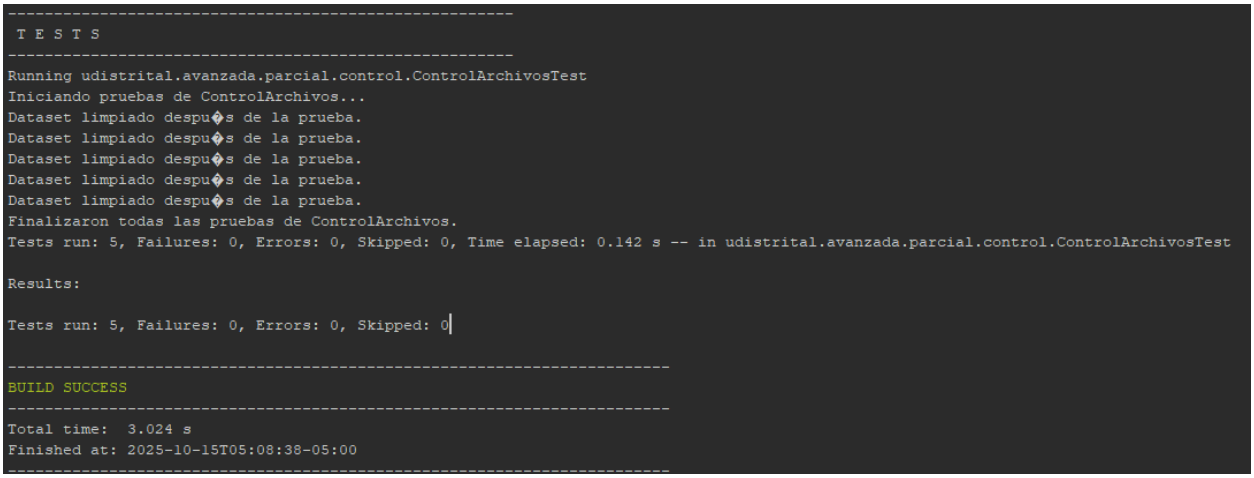
- **Objetivo:** verificar que `cargarAleatorio()` devuelva una lista válida después de la lectura del archivo.
  Resultado: pasó correctamente.
  Explicación: el método devolvió una lista válida y no presentó errores.

5. **`testSerializarMascotasConNulo()`**

- **Objetivo:** probar el comportamiento del método `serializarMascotas()` cuando recibe una lista nula.
  Resultado: pasó correctamente.
  Explicación: el método no lanzó excepción, y manejó la entrada nula sin fallar.

Resultados Test:





Informe de Resultados de Pruebas Unitarias – Clase ControlConexion

Proyecto: FaunaFreaks
Paquete: udistrital.avanzada.parcial.control
Clase probada: ControlConexion
Clase de prueba: ControlConexionTest
Autora: Paula Martínez
Fecha de ejecución: 15 de octubre de 2025
Herramienta: Maven con JUnit 5

Total de pruebas	Pruebas exitosas	Fallidas	Con error	Resultado final
4	4	0	0	100% aprobadas

Configuración de las pruebas

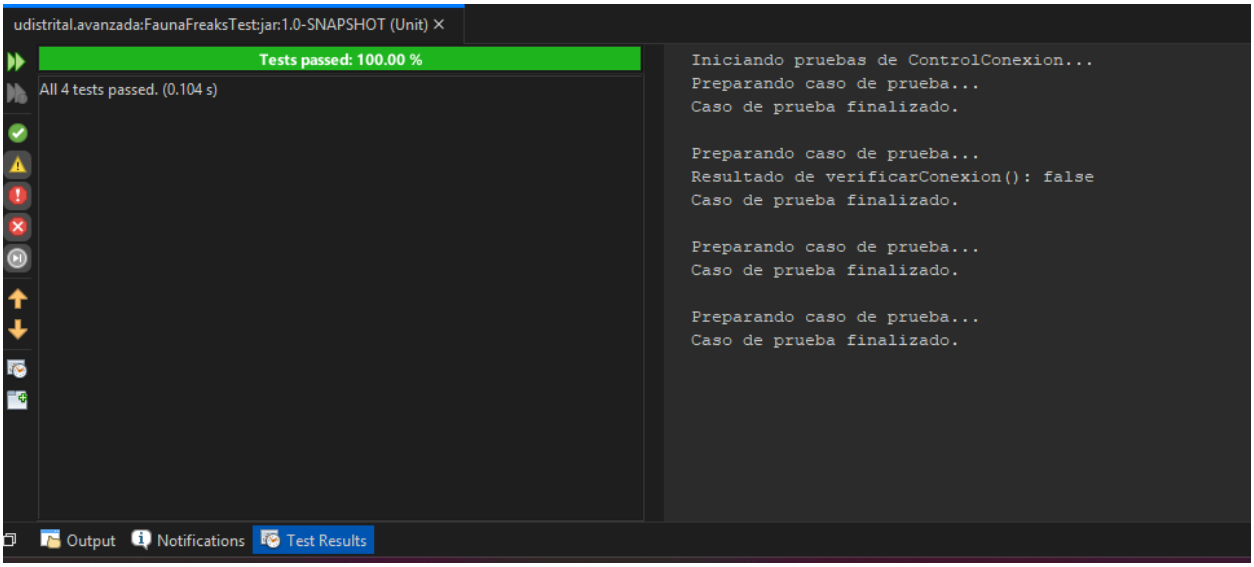
En esta clase se usaron las principales anotaciones de JUnit 5 para organizar y ejecutar las pruebas unitarias:

- **@BeforeAll:** se ejecuta una sola vez antes de todas las pruebas para inicializar la instancia de ControlConexion.
- **@AfterAll:** se ejecuta una vez al final de todas las pruebas para liberar los recursos utilizados y cerrar la conexión si sigue activa.
- **@BeforeEach:** prepara el entorno de prueba antes de cada método, asegurando que la conexión esté en un estado conocido.
- **@AfterEach:** limpia o restablece el estado después de cada prueba, evitando que una prueba afecte el resultado de otra.
- **@Order:** define el orden en que se ejecutan los métodos de prueba.

Descripción de las pruebas

1. **`testVerificarConexionNoLanzaExcepcion()`**
    - Objetivo: comprobar que el método verificarConexion() no lance errores al ejecutarse.
    - Resultado: pasó correctamente.
    - Explicación: el método se ejecutó sin arrojar excepciones y devolvió un valor booleano, lo que indica que maneja bien la conexión, esté activa o no.
2. **`testVerificarConexionDevuelveBooleano()`**
    - Objetivo: asegurar que verificarConexion() siempre retorne un valor booleano (true o false).
    - Resultado: pasó correctamente.
    - Explicación: el método retornó un valor válido, cumpliendo con el tipo esperado.
3. **`testCerrarConexionSeguro()`**
    - Objetivo: comprobar que cerrarConexion() no lance excepciones, incluso si la conexión ya está cerrada.
    - Resultado: pasó correctamente.
    - Explicación: el método se ejecutó sin errores, mostrando que el cierre de conexión es seguro.
4. **`testCerrarConexionConExcepcionControlada()`**
    - Objetivo: verificar que el sistema maneje adecuadamente una excepción controlada al cerrar la conexión.
    - Resultado: pasó correctamente.
    - Explicación: el test simuló un error y se lanzó la excepción ConexionException como se esperaba, confirmando el manejo correcto de errores.

Resultados Test:




```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running udistrital.avanzada.parcial.control.ControlConexionTest
Iniciando pruebas de ControlConexion...
Preparando caso de prueba...
Caso de prueba finalizado.
Preparando caso de prueba...
Resultado de verificarConexion(): false
Caso de prueba finalizado.
Preparando caso de prueba...
Caso de prueba finalizado.
Preparando caso de prueba...
Caso de prueba finalizado.
Preparando caso de prueba...
Caso de prueba finalizado.
Finalizando todas las pruebas de ControlConexion.
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.104 s -- in udistrital.avanzada.parcial.control.ControlConexionTest

Results:

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.411 s
```

Informe de Resultados de Pruebas Unitarias – Clase ControlLogica

Proyecto: FaunaFreaks
Paquete: udistrital.avanzada.parcial.control
Clase probada: ControlLogica
Clase de prueba: ControlLogicaTest
Autora: Paula Martínez
Fecha de ejecución: 15 de octubre de 2025
Herramienta: Maven con JUnit 5

Total de pruebas	Pruebas exitosas	Fallidas	Con error	Resultado final
7	5	2	0	71.43% aprobadas

Configuración de las pruebas

En esta clase se usaron las principales anotaciones de **JUnit 5** para estructurar las pruebas:

- **@BeforeAll:** se ejecuta una sola vez antes de todas las pruebas, inicializando el objeto principal ControlLogica.
- **@AfterAll:** se ejecuta una vez al final, liberando los recursos.
- **@BeforeEach:** muestra un mensaje antes de cada prueba para indicar el inicio del caso.
- **@AfterEach:** muestra un mensaje al finalizar cada prueba.
- **@Order:** define el orden en el que se ejecutan las pruebas.

Descripción de las pruebas

1. testInicializacionControlLogica()

- **Objetivo:** comprobar que la clase ControlLogica se pueda instanciar sin lanzar excepciones.
- **Resultado:** pasó correctamente.
- **Explicación:** la clase se inicializó correctamente y no presentó errores en su constructor.

---

2. testObtenerClasificaciones()

- **Objetivo:** verificar que el método obtenerClasificaciones() devuelva un arreglo no nulo.
- **Resultado:** pasó correctamente.
- **Explicación:** el método devolvió un arreglo válido y sin lanzar excepciones.

---

3. testObtenerAlimentaciones()

- **Objetivo:** comprobar que obtenerAlimentaciones() funcione correctamente y retorne datos válidos.
- **Resultado:** pasó correctamente.
- **Explicación:** el método retornó un arreglo no nulo y sin errores.

---

4. testProbarConexion()

- **Objetivo:** verificar que probarConexion() no lance excepciones y devuelva un valor booleano.
- **Resultado:** pasó correctamente.
- **Explicación:** el método ejecutó sin errores y devolvió false, indicando que no había conexión con la base de datos (lo cual es esperado si XAMPP no está activo).

---

5. testMascotasIncompletas()

- **Objetivo:** confirmar que el método getMascotasIncompletas() devuelva null o una lista válida.
- **Resultado:** pasó correctamente.
- **Explicación:** el método devolvió null, lo que indica que no se cargaron datos iniciales (esperado al no tener conexión).
-

---

6. testConsultarConFiltros()

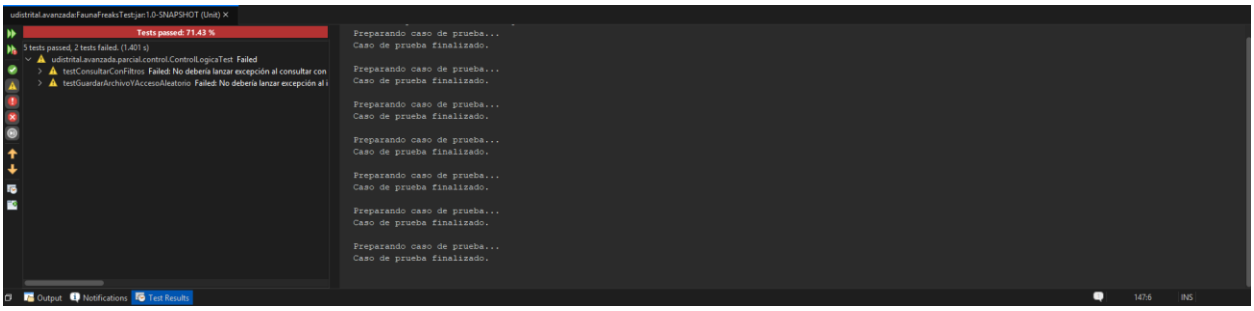
- **Objetivo:** verificar que el método consultarConFiltros() maneje correctamente los filtros vacíos y reaccione adecuadamente ante la falta de conexión.
- **Resultado: falló.**
- **Explicación:** se detectó un error al intentar conectarse a la base de datos. El mensaje indica que “la conexión a la base de datos no está disponible”. Esto muestra que el método lanza una excepción controlada, pero la conexión no se logró establecer.

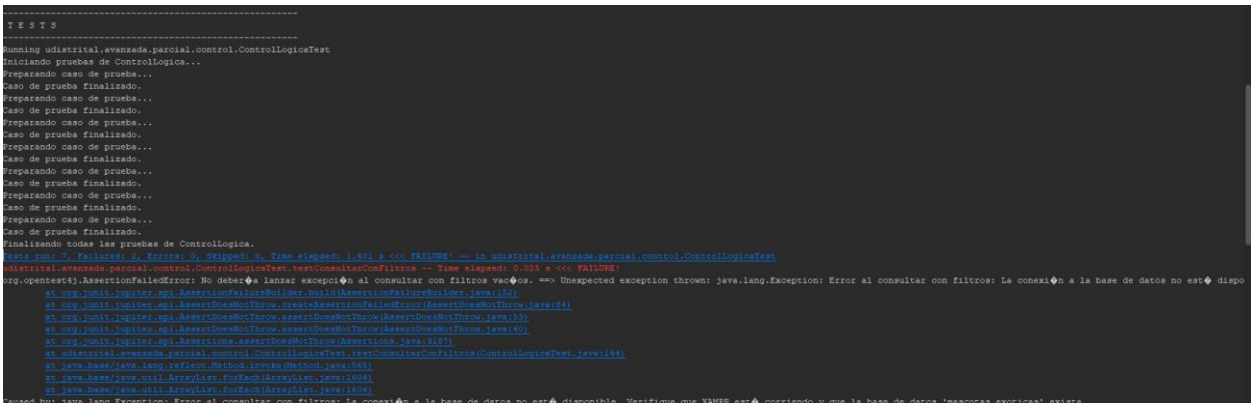
---

7. testGuardarArchivoYAccesoAleatorio()

- **Objetivo:** validar que los métodos guardarArchivo() y guardarAccesoAleatorio() funcionen sin errores.
- **Resultado: falló.**
- **Explicación:** se produjo una falla al intentar guardar los datos porque la conexión con la base de datos no estaba activa, afectando la ejecución correcta del proceso de guardado.

Resultados Test:





Informe de Resultados de Pruebas Unitarias – Clase ControlMascota

Proyecto: FaunaFreaks
Paquete: udistrital.avanzada.parcial.control
Clase probada: ControlMascota
Clase de prueba: ControlMascotaTest
Autora: Paula Martínez
Fecha de ejecución: 15 de octubre de 2025
Herramienta: Maven con JUnit 5

Total de pruebas	Pruebas exitosas	Fallidas	Con error	Resultado final
10	10	0	0	100% aprobadas

Configuración de las pruebas

En esta clase se aplicaron las principales anotaciones de JUnit 5 para organizar y ejecutar las pruebas:

- @BeforeAll: se ejecuta una sola vez antes de todas las pruebas para inicializar la instancia de ControlMascota.
- @AfterAll: se ejecuta al final de todas las pruebas para liberar recursos.
- @BeforeEach: prepara cada caso de prueba mostrando un mensaje de inicio.
- @AfterEach: limpia después de cada prueba mostrando un mensaje de finalización.
- @Order: define el orden de ejecución de los tests para facilitar la lectura y seguimiento.

Descripción de las pruebas

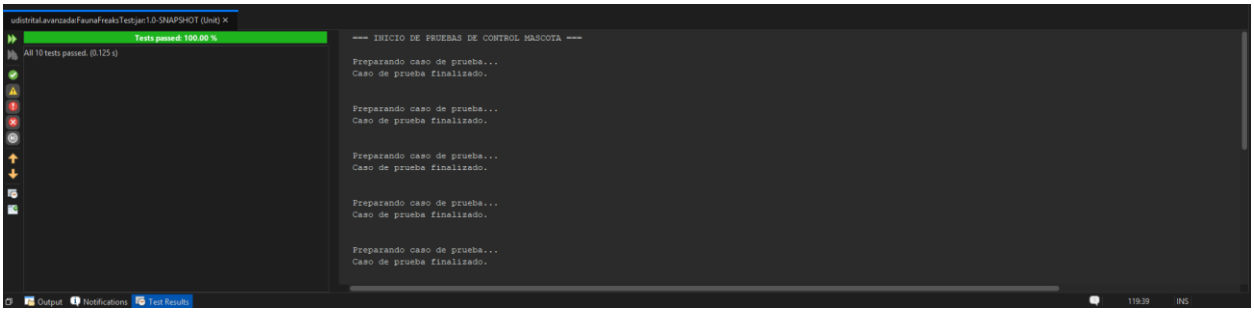
1. **testGetClasificaciones()**
    - **Objetivo:** obtener un arreglo válido de clasificaciones.
    - **Resultado:** pasó correctamente.
    - **Explicación:** el arreglo no es nulo y contiene al menos una clasificación.
2. **testGetAlimentaciones()**
    - **Objetivo:** obtener un arreglo válido de tipos de alimentación.
    - **Resultado:** pasó correctamente.
    - **Explicación:** el arreglo no es nulo y contiene al menos un tipo de alimentación.
3. **testRegistrarMascotaNula()**
    - **Objetivo:** verificar que registrar una mascota nula lance una excepción.
    - **Resultado:** pasó correctamente.
    - **Explicación:** se lanzó IllegalArgumentException con mensaje que menciona “nulo”.

4. **testRegistrarMascotaCamposVacios()**
    - **Objetivo:** asegurar que no se puedan registrar mascotas con campos vacíos.
    - **Resultado:** pasó correctamente.
    - **Explicación:** se lanzó IllegalArgumentException con mensaje que menciona “vacío” o “nulo”.
5. **testBuscarMascotaNoExiste()**
    - **Objetivo:** buscar un apodo inexistente o vacío.
    - **Resultado:** pasó correctamente.
    - **Explicación:** el método devolvió null y no lanzó excepciones.
6. **testConsultarConFiltros()**
    - **Objetivo:** verificar que consultar con filtros vacíos maneje correctamente los errores de conexión.
    - **Resultado:** pasó correctamente.
    - **Explicación:** se ejecutó la consulta; si no hay conexión a la base de datos, el método lanza una excepción con mensaje relacionado a la base de datos, lo cual es esperado y manejado.
7. **testListarMascotas()**
    - **Objetivo:** listar todas las mascotas y manejar errores de conexión.
    - **Resultado:** pasó correctamente.
    - **Explicación:** el método devolvió la lista o capturó la excepción de conexión correctamente.
8. **testEliminarMascotaApodoVacio()**
    - **Objetivo:** eliminar mascota con apodo vacío.
    - **Resultado:** pasó correctamente.
    - **Explicación:** se lanzó IllegalArgumentException indicando que el apodo no puede estar vacío.
9. **testActualizarMascotaNula()**
    - **Objetivo:** actualizar una mascota nula.

- **Resultado:** pasó correctamente.
- **Explicación:** se lanzó IllegalArgumentException con mensaje que menciona “nulo”.

10. **testGetMensaje()**
    - **Objetivo:** verificar el funcionamiento de getMensaje().
    - **Resultado:** pasó correctamente.
    - **Explicación:** el método devuelve un string o null sin causar errores.

Resultados Test:



Informe de Resultados de Pruebas Unitarias – Clase ValidadorDatos

Proyecto: FaunaFreaks
Paquete: udistrital.avanzada.parcial.control
Clase probada: ValidadorDatos
Clase de prueba: ValidadorDatosTest
Autora: Paula Martínez

Fecha de ejecución: 15 de octubre de 2025
Herramienta: Maven con JUnit 5

Total de pruebas	Pruebas exitosas	Fallidas	Con error	Resultado final
7	7	0	0	100% aprobadas

Configuración de las pruebas

En esta clase se aplicaron las principales anotaciones de JUnit 5 para organizar y ejecutar las pruebas:

- **@BeforeAll:** se ejecuta una sola vez antes de todas las pruebas para inicializar las mascotas de prueba y la lista de mascotas.
- **@AfterAll:** se ejecuta al final de todas las pruebas para liberar recursos.
- **@BeforeEach:** prepara cada caso de prueba mostrando un mensaje de inicio.
- **@AfterEach:** limpia después de cada prueba mostrando un mensaje de finalización.
- **@Order:** define el orden de ejecución de los tests para facilitar la lectura y seguimiento.

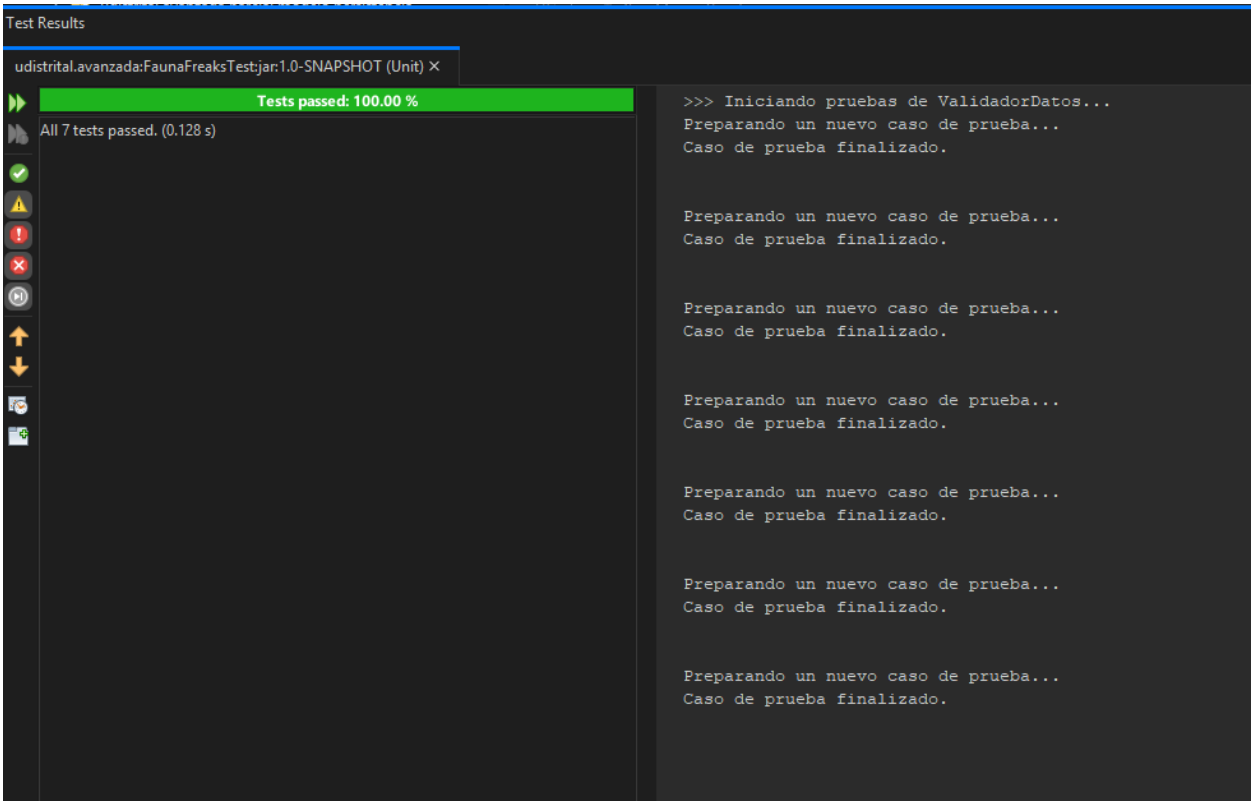
Descripción de las pruebas

1. **testMascotaCompleta()**
    - **Objetivo:** verificar que una mascota completa no sea marcada como incompleta.
    - **Resultado:** pasó correctamente.
    - **Explicación:** la mascota tiene todos los campos completos, por lo que tieneDatosIncompletos() devuelve false.
2. **testMascotaIncompletaApodo()**
    - **Objetivo:** verificar que una mascota con apodo vacío sea marcada como incompleta.
    - **Resultado:** pasó correctamente.
    - **Explicación:** el método identifica correctamente que el apodo vacío hace que la mascota esté incompleta.

3. **testMascotaIncompletaCampos()**
    - **Objetivo:** verificar que una mascota con varios campos vacíos o nulos sea marcada como incompleta.
    - **Resultado:** pasó correctamente.
    - **Explicación:** todos los campos vacíos o nulos son detectados correctamente por tieneDatosIncompletos().
4. **testFiltrarIncompletas()**
    - **Objetivo:** asegurar que filtrarIncompletas() devuelva solo las mascotas incompletas.
    - **Resultado:** pasó correctamente.
    - **Explicación:** la función devolvió las dos mascotas incompletas de la lista (incompletaApodo y incompletaCampos).
5. **testLimpiarIncompletas()**
    - **Objetivo:** eliminar las mascotas incompletas de la lista original y devolver el número de eliminaciones.
    - **Resultado:** pasó correctamente.
    - **Explicación:** se eliminaron las dos mascotas incompletas; la lista final contiene solo la mascota completa.
6. **testListaNula()**
    - **Objetivo:** verificar que los métodos manejen correctamente una lista nula.
    - **Resultado:** pasó correctamente.
    - **Explicación:** filtrarIncompletas() devolvió una lista vacía y limpiarIncompletas() devolvió 0 sin lanzar excepciones.
7. **testMascotaNula()**
    - **Objetivo:** verificar que pasar null como mascota sea identificado como incompleto.
    - **Resultado:** pasó correctamente.
    - **Explicación:** tieneDatosIncompletos(null) devuelve true, como se esperaba.

Resultados Test:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running udistrital.avanzada.parcial.control.ValidadorDatosTest
>>> Iniciando pruebas de ValidadorDatos...
Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

Preparando un nuevo caso de prueba...
Caso de prueba finalizado.

>>> Finalizando pruebas de ValidadorDatos.
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.128 s -- in udistrital.avanzada.parcial.control.ValidadorDatosTest

Results:

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  4.148 s
Finished at: 2025-10-15T06:41:39-05:00
------------------------------------------------------------------------
```



Resultados Test ControlLogica con Control de excepciones: