Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

ALGORITMOS Y ESTRUCTURAS DE DATOS

Matías Oreiro – 239479

M3C

Docente: Sebastián Pesce

Analista en Tecnologías de la Información

12/05/2025

Contenido

Diseño y justificaciones

1. Crear sistema de gestión

Inicializa las listas del sistema (clientes, salas y eventos).

2. Registrar sala

Permite ingresar una nueva sala si la capacidad es válida (> 0) y si no existe una sala con el mismo nombre. Se valida para evitar duplicados y asegurar que los

datos sean correctos para registrarla. Añade las salas al inicio, de manera que queden ordenadas por orden inverso de llegada.

3. Eliminar sala

Elimina una sala si existe. Se recorre la lista buscando coincidencia por nombre usando compareTo y, si no se encuentra, devuelve un error. Esto permite evitar eliminaciones sin querer.

4. Registrar evento

Registra un evento si el aforo necesario para desarrollar el evento es mayor a 0, si el código no está registrado y si hay una sala disponible para la fecha y el aforo requerido. Se apoya en un conjunto de funciones auxiliares que permiten automatizar la asignación de salas y asegurar la disponibilidad sin conflictos. Añade los eventos a la lista de manera ordenada usando compareTo para ordenarlos por código de evento.

5. Registrar cliente

Registra un cliente solo si la cédula tiene 8 dígitos (como se indica en la letra) y si no existe un cliente con esa misma cédula. Se asegura así la unicidad del identificador y se validan datos básicos antes de agregarlos. Se añaden de manera ordenada usando compareTo que compara las cédulas, y como las cédulas están almacenadas como String, primero se convierten a enteros para poder hacer una comparación numérica más coherente.

6. Listar salas, listar eventos y listar clientes

Recorren las listas respectivas y devuelven un texto con los elementos utilizando el toString, quien muestra para Sala: nombre-capacidad, Evento: codigo-descripcion-sala-fecha-aforo, y Cliente: cedula-nombre, y los separa por un #, como indica la consigna.

7. Es sala óptima

Recorre las filas y las columnas de la vista de una sala (matriz de O y X) y verifica si al menos dos columnas cumplen con que el bloque más largo de O es mayor a la cantidad total de X. Es la lógica solicitada para evaluar si una sala es óptima.

8. Sala disponible (auxiliar)

Recorre la lista de salas y busca una sala que tenga la capacidad necesaria y que esté libre en la fecha dada. Utiliza la función auxiliar de fecha disponible, y si esta devuelve true significa que la sala está disponible en esa fecha. Retorna un objeto Sala que significa que la sala está disponible, pero devuelve null si no existe ninguna.

9. Fecha disponible (auxiliar)

Recorre la lista de eventos y verifica si una sala ya está ocupada en una fecha dada. Ésta devuelve un boolean indicando si está disponible o no. Verifica en cada evento que la sala que se pasa por parámetro no coincida con la ingresada en un evento y que no coincida con la fecha que también se pasa por parámetro.

Código de pruebas

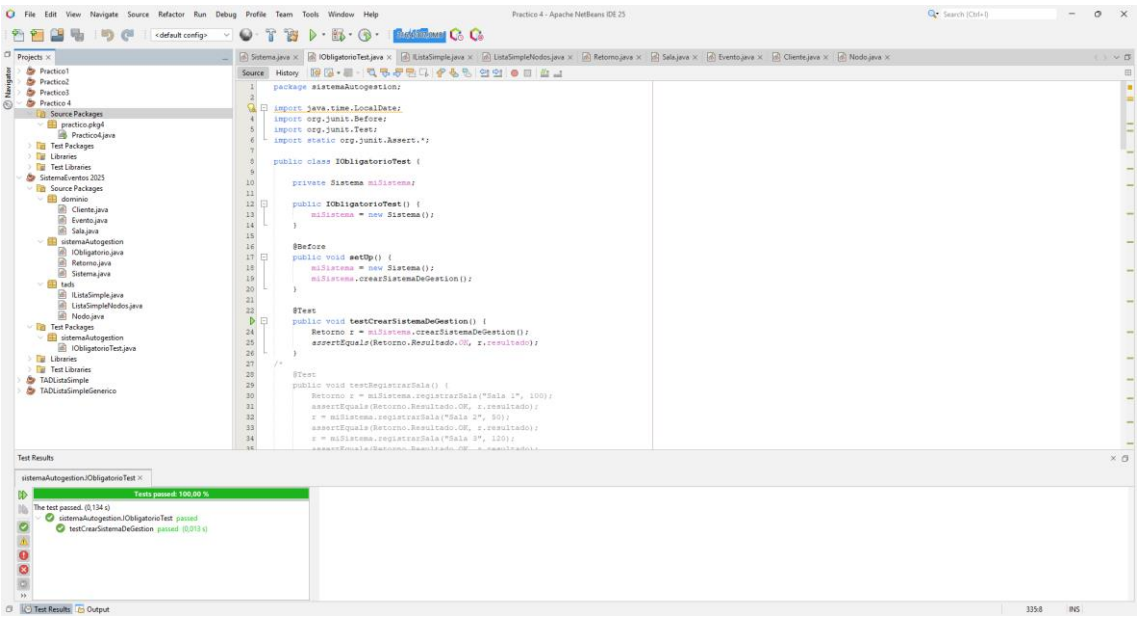
Requisito 1.0 – Inicialización

```
package sistemaAutogestion;

import java.time.LocalDate;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.*;


public class IObligatorioTest {

    private Sistema miSistema;

}

    public IObligatorioTest() {

        miSistema = new Sistema();

    }


    @Before

    public void setUp() {

        miSistema = new Sistema();

        miSistema.crearSistemaDeGestion();
```

}

Requisito 1.1 - Registro de Salas, Eventos y Clientes - OK

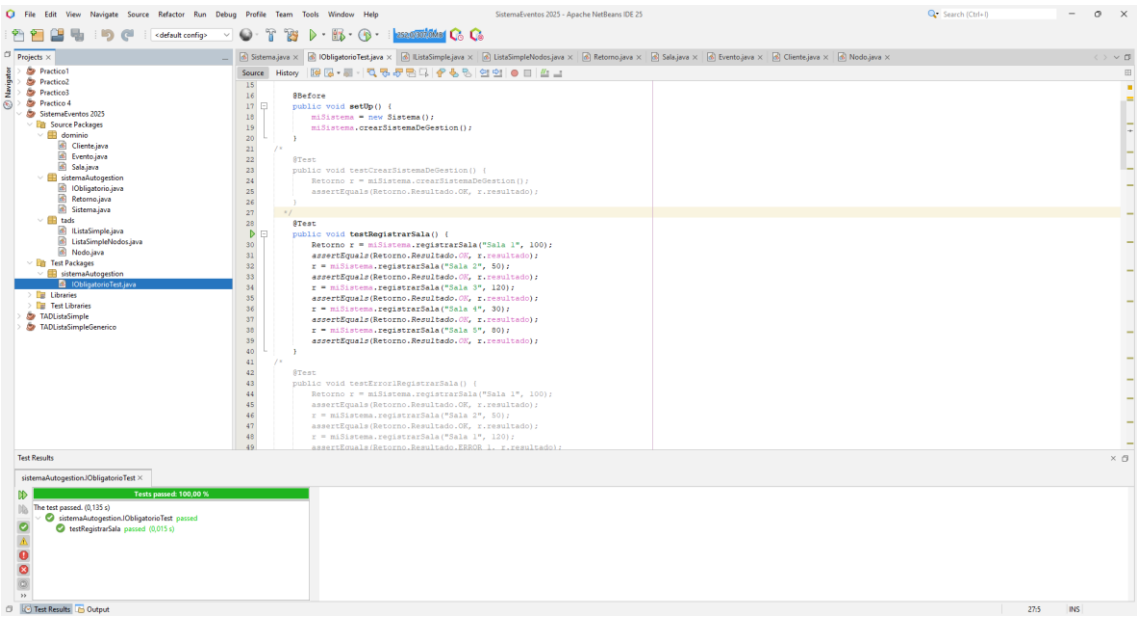
```
@Test

public void testCrearSistemaDeGestion() {

    Retorno r = miSistema.crearSistemaDeGestion();

    assertEquals(Retorno.Resultado.OK, r.resultado);

}
```



Requisito 1.2 – Registrar una sala - OK

```
@Test

public void testRegistrarSala() {

    Retorno r = miSistema.registrarSala("Sala 1", 100);

    assertEquals(Retorno.Resultado.OK, r.resultado);
```

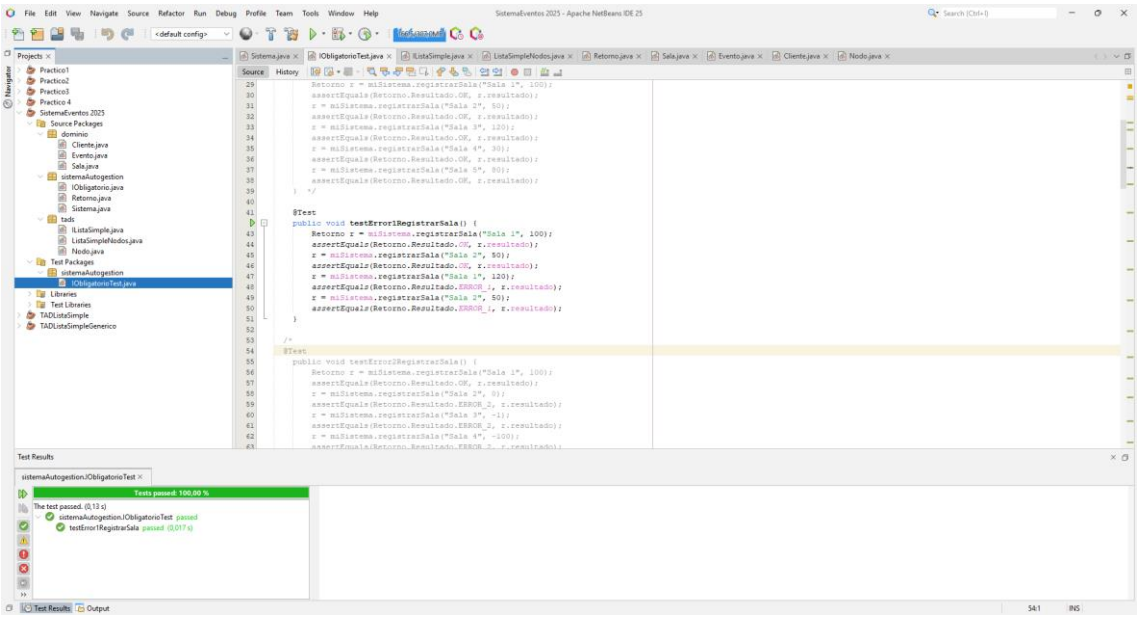
```
        r = miSistema.registrarSala("Sala 2", 50);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 3", 120);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 4", 30);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 5", 80);
        assertEquals(Retorno.Resultado.OK, r.resultado);
    }
```



Requisito 1.2 – Registrar una sala – Error 1, ya existe una sala con el mismo nombre

```
    @Test
    public void testError1RegistrarSala() {
```

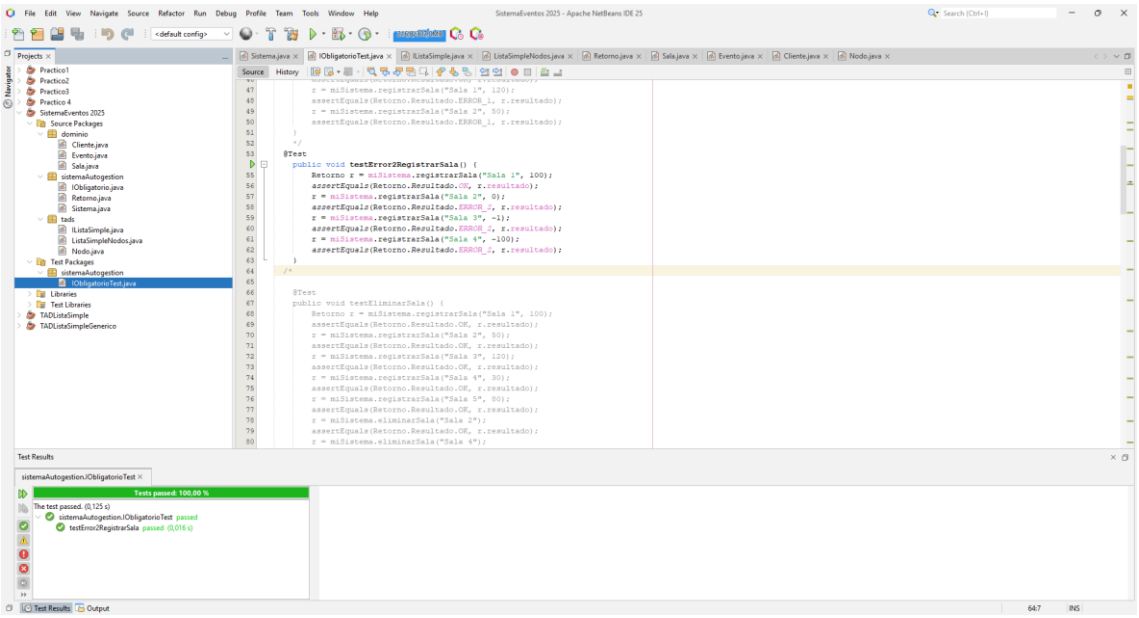
```
        Retorno r = miSistema.registrarSala("Sala 1", 100);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 2", 50);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 1", 120);
        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
        r = miSistema.registrarSala("Sala 2", 50);
        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    }
```



Requisito 1.2 – Registrar una sala – Error 2, la capacidad es menor o igual a 0

```
    @Test
```

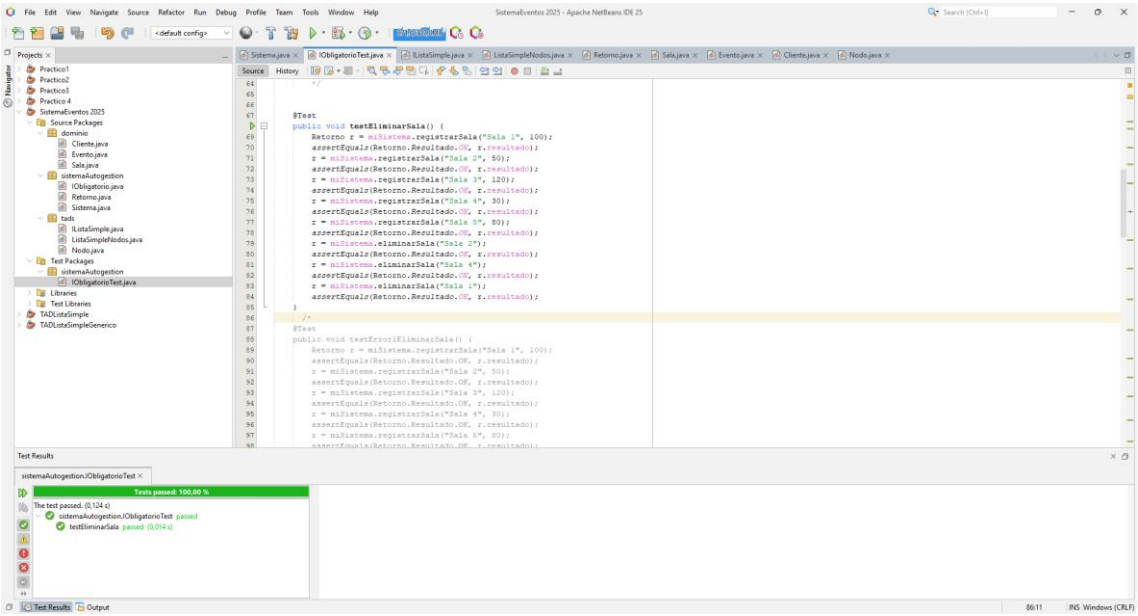
```
public void testError2RegistrarSala() {
    Retorno r = miSistema.registrarSala("Sala 1", 100);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 2", 0);
    assertEquals(Retorno.Resultado.ERROR_2, r.resultado);
    r = miSistema.registrarSala("Sala 3", -1);
    assertEquals(Retorno.Resultado.ERROR_2, r.resultado);
    r = miSistema.registrarSala("Sala 4", -100);
    assertEquals(Retorno.Resultado.ERROR_2, r.resultado);
}
```



Requisito 1.3 – Eliminar una sala - OK

```
@Test
public void testEliminarSala() {
```

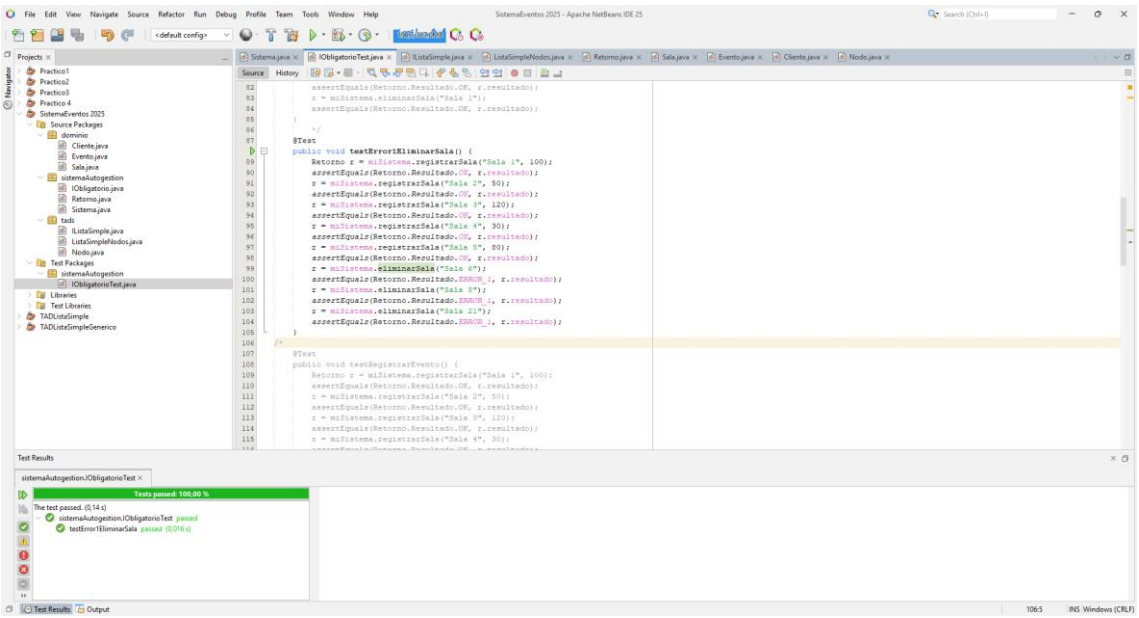
```
    Retorno r = miSistema.registrarSala("Sala 1", 100);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 2", 50);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 3", 120);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 4", 30);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 5", 80);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.eliminarSala("Sala 2");
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.eliminarSala("Sala 4");
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.eliminarSala("Sala 1");
    assertEquals(Retorno.Resultado.OK, r.resultado);
}
```



Requisito 1.3 – Eliminar una sala – Error 1, no se encontró una sala con ese nombre

```
@Test

public void testError1EliminarSala() {

    Retorno r = miSistema.registrarSala("Sala 1", 100);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 2", 50);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 3", 120);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 4", 30);

    assertEquals(Retorno.Resultado.OK, r.resultado);
```

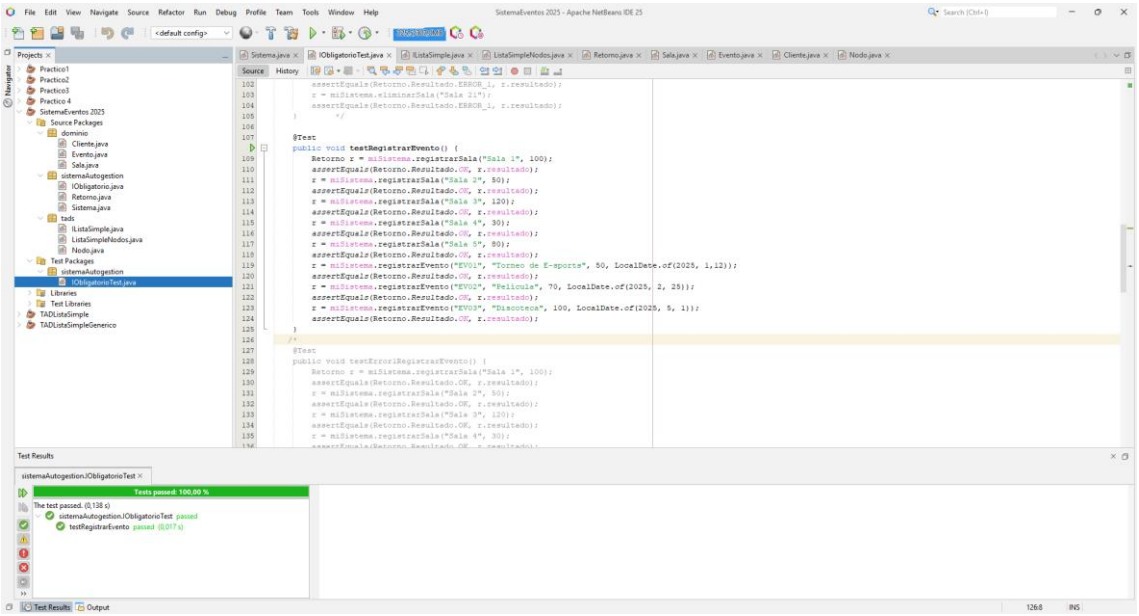
```
        r = miSistema.registrarSala("Sala 5", 80);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.eliminarSala("Sala 6");
        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
        r = miSistema.eliminarSala("Sala 8");
        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
        r = miSistema.eliminarSala("Sala 21");
        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    }
```



Requisito 1.4 – Registrar un evento - OK

```
    @Test
    public void testRegistrarEvento() {
        Retorno r = miSistema.registrarSala("Sala 1", 100);
```

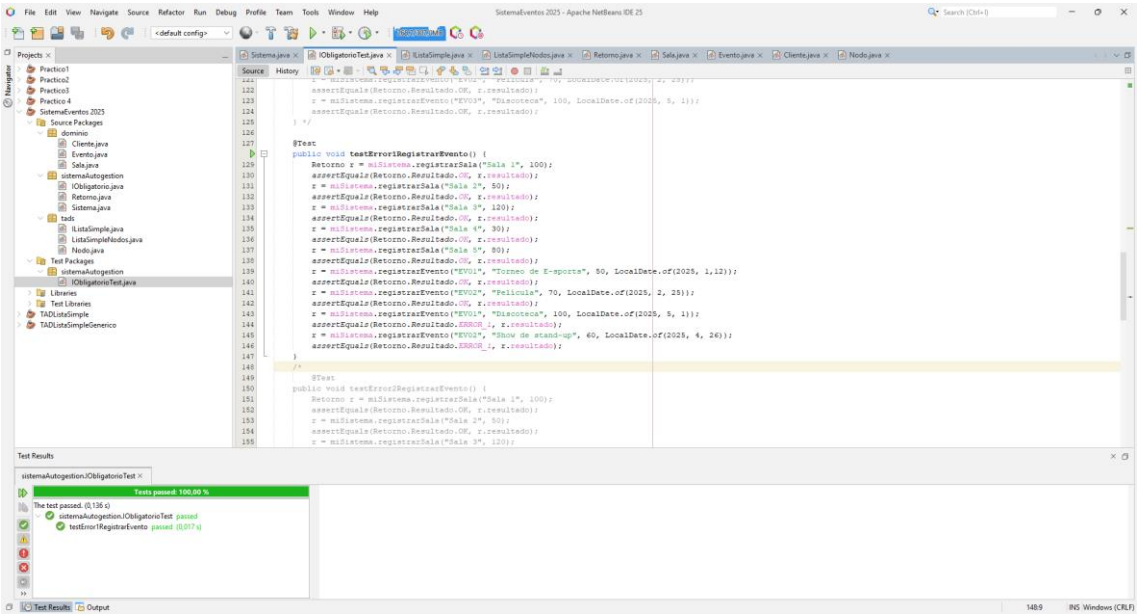
```
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 2", 50);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 3", 120);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 4", 30);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarSala("Sala 5", 80);
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarEvento("EV01", "Torneo de E-sports", 50,
LocalDate.of(2025, 1,12));
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarEvento("EV02", "Película", 70, LocalDate.of(2025, 2,
25));
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarEvento("EV03", "Discoteca", 100, LocalDate.of(2025, 5,
1));
        assertEquals(Retorno.Resultado.OK, r.resultado);
    }
```



Requisito 1.4 – Registrar un evento – Error 1, ya existe un evento con el código registrado

```
@Test

public void testError1RegistrarEvento() {

    Retorno r = miSistema.registrarSala("Sala 1", 100);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 2", 50);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 3", 120);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 4", 30);

    assertEquals(Retorno.Resultado.OK, r.resultado);
```

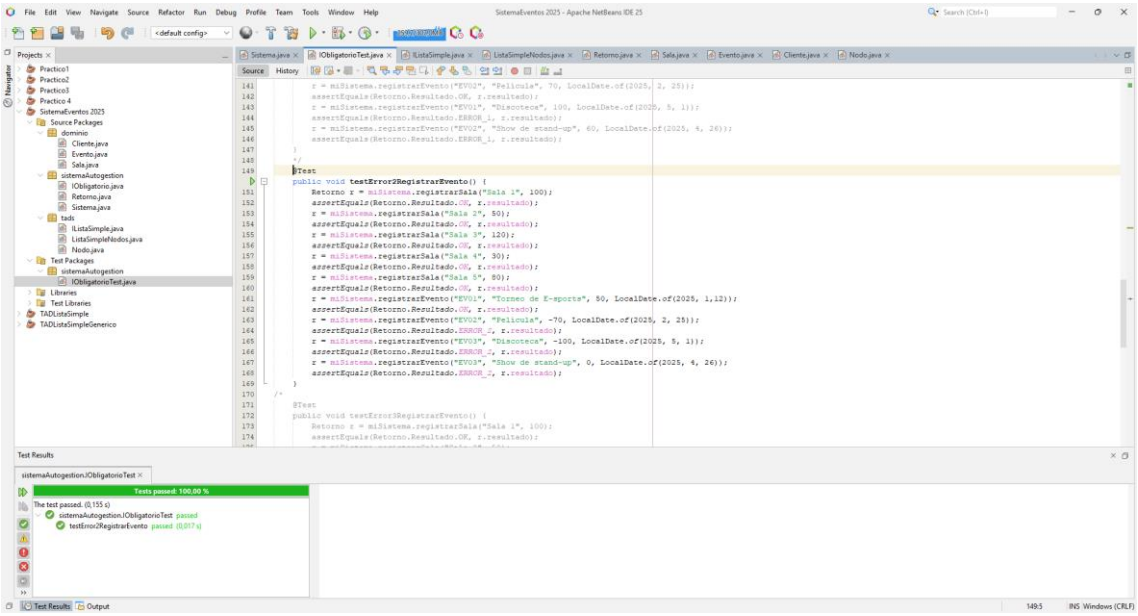
```
        r = miSistema.registrarSala("Sala 5", 80);

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV01", "Torneo de E-sports", 50,
LocalDate.of(2025, 1,12));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV02", "Película", 70, LocalDate.of(2025, 2,
25));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV01", "Discoteca", 100, LocalDate.of(2025, 5,
1));

        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);

        r = miSistema.registrarEvento("EV02", "Show de stand-up", 60,
LocalDate.of(2025, 4, 26));

        assertEquals(Retorno.Resultado.ERROR_1, r.resultado);

    }
```



Requisito 1.4 – Registrar un evento – Error 2, el aforo necesario es menor o igual a 0

```
    @Test
public void testError2RegistrarEvento() {
    Retorno r = miSistema.registrarSala("Sala 1", 100);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 2", 50);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 3", 120);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 4", 30);
    assertEquals(Retorno.Resultado.OK, r.resultado);
```

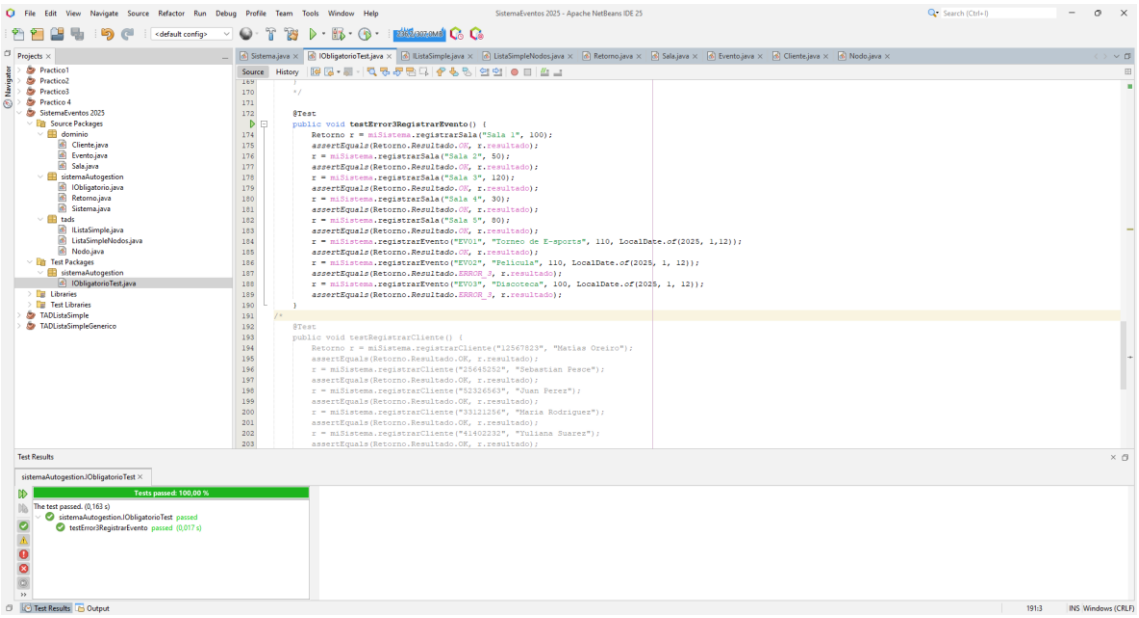
```
        r = miSistema.registrarSala("Sala 5", 80);

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV01", "Torneo de E-sports", 50,
LocalDate.of(2025, 1,12));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV02", "Película", -70, LocalDate.of(2025, 2,
25));

        assertEquals(Retorno.Resultado.ERROR_2, r.resultado);

        r = miSistema.registrarEvento("EV03", "Discoteca", -100, LocalDate.of(2025,
5, 1));

        assertEquals(Retorno.Resultado.ERROR_2, r.resultado);

        r = miSistema.registrarEvento("EV03", "Show de stand-up", 0,
LocalDate.of(2025, 4, 26));

        assertEquals(Retorno.Resultado.ERROR_2, r.resultado);

    }
```



Requisito 1.4 – Registrar un evento – Error 3, no se encontró una sala disponible para esa fecha con aforo suficiente

```
@Test

public void testError3RegistrarEvento() {

    Retorno r = miSistema.registrarSala("Sala 1", 100);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 2", 50);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 3", 120);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 4", 30);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 5", 80);
```

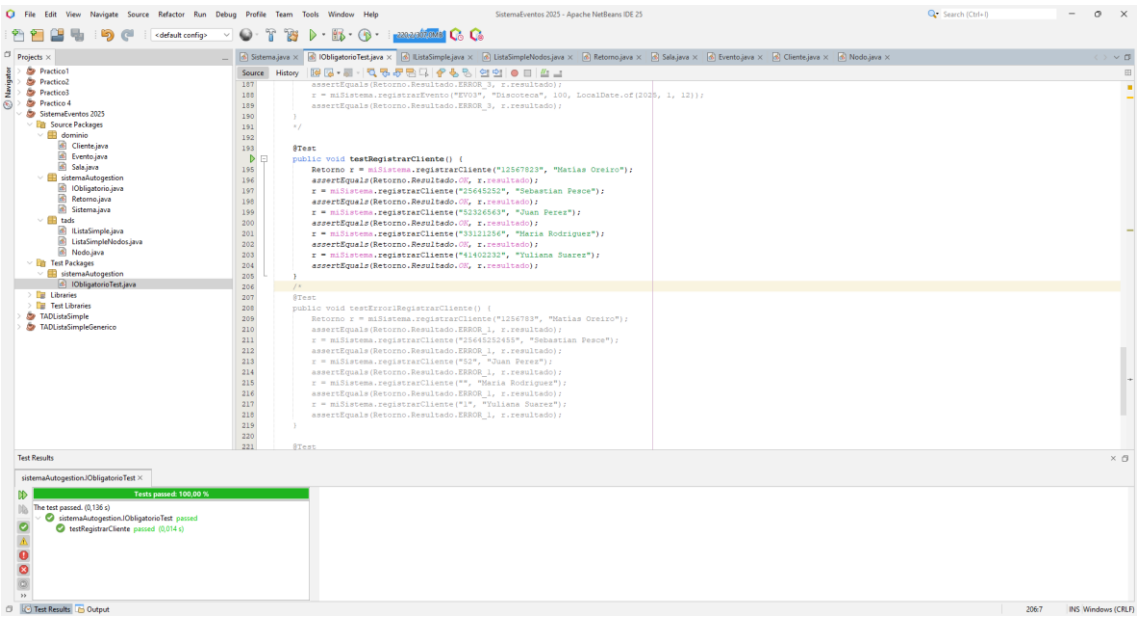
```
        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV01", "Torneo de E-sports", 110, LocalDate.of(2025, 1,12));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("EV02", "Película", 110, LocalDate.of(2025, 1, 12));

        assertEquals(Retorno.Resultado.ERROR_3, r.resultado);

        r = miSistema.registrarEvento("EV03", "Discoteca", 100, LocalDate.of(2025, 1, 12));

        assertEquals(Retorno.Resultado.ERROR_3, r.resultado);

    }
```



Requisito 1.5 – Registrar un cliente - OK

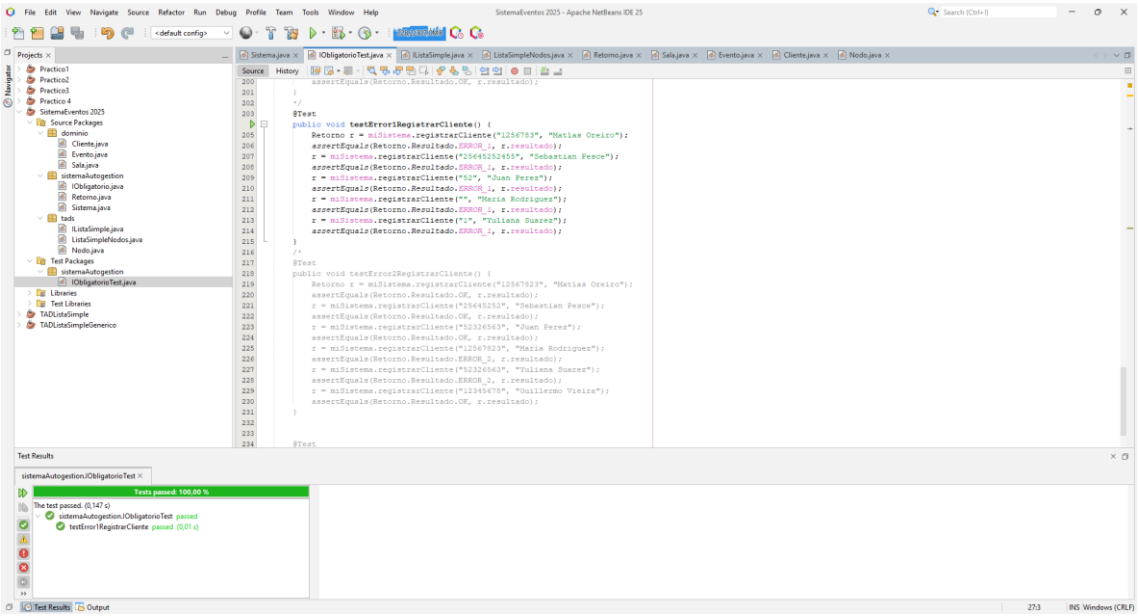
```
    @Test

    public void testRegistrarCliente() {
```

```
        Retorno r = miSistema.registrarCliente("12567823", "Matías Oreiro");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarCliente("25645252", "Sebastian Pesce");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarCliente("52326563", "Juan Perez");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarCliente("33121256", "Maria Rodriguez");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarCliente("41402232", "Yuliana Suarez");
        assertEquals(Retorno.Resultado.OK, r.resultado);
    }
```



Requisito 1.5 – Registrar un cliente – Error 1, la cédula tiene un formato inválido

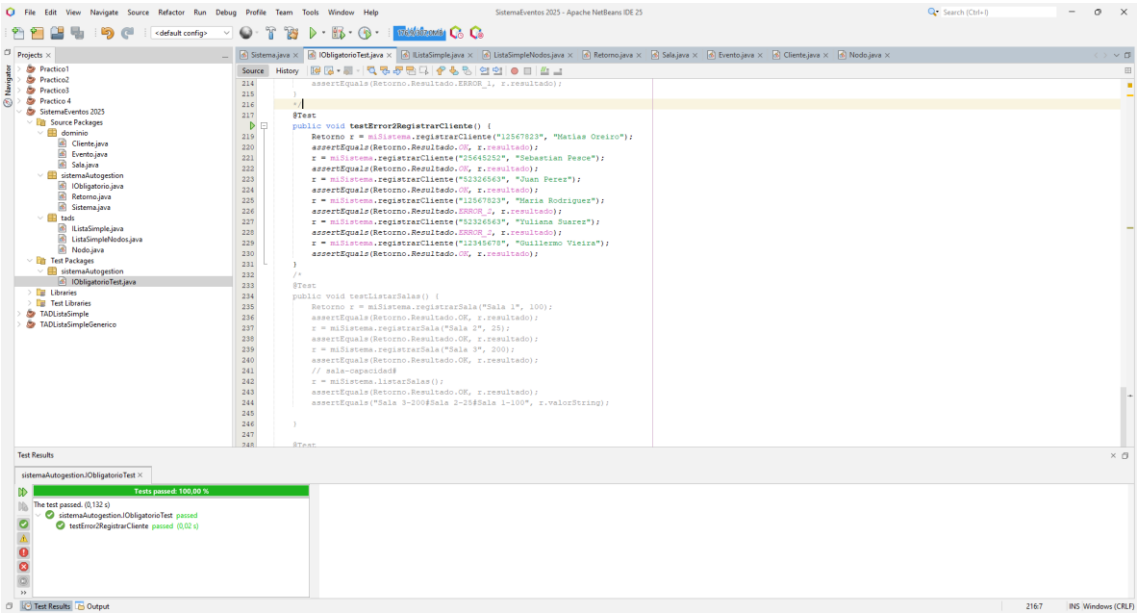
```
@Test
public void testError1RegistrarCliente() {
    Retorno r = miSistema.registrarCliente("1256783", "Matías Oreiro");
    assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    r = miSistema.registrarCliente("25645252455", "Sebastian Pesce");
    assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    r = miSistema.registrarCliente("52", "Juan Perez");
    assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    r = miSistema.registrarCliente("", "Maria Rodriguez");
    assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
    r = miSistema.registrarCliente("1", "Yuliana Suarez");
    assertEquals(Retorno.Resultado.ERROR_1, r.resultado);
}
```



Requisito 1.5 – Registrar un cliente – Error 2, ya existe un cliente registrado con esa cédula

```
@Test

public void testError2RegistrarCliente() {

    Retorno r = miSistema.registrarCliente("12567823", "Matías Oreiro");

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarCliente("25645252", "Sebastian Pesce");

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarCliente("52326563", "Juan Perez");

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarCliente("12567823", "Maria Rodriguez");

    assertEquals(Retorno.Resultado.ERROR_2, r.resultado);
```

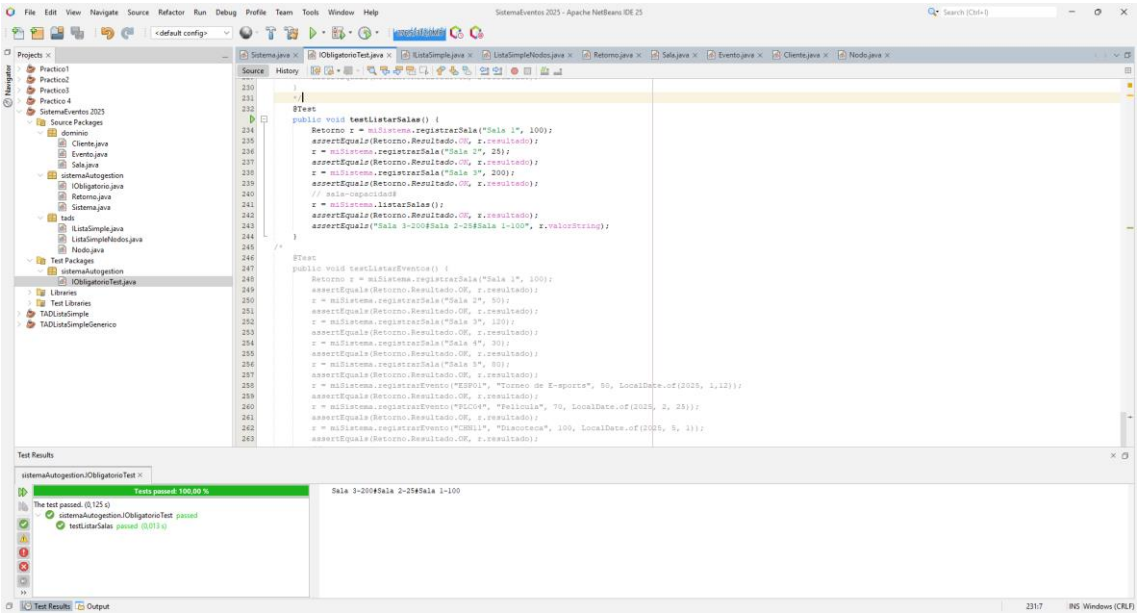
```
    r = miSistema.registrarCliente("52326563", "Yuliana Suarez");
    assertEquals(Retorno.Resultado.ERROR_2, r.resultado);
    r = miSistema.registrarCliente("12345678", "Guillermo Vieira");
    assertEquals(Retorno.Resultado.OK, r.resultado);
}
```



Requisito 2.1 – Listar salas - OK

```
@Test
public void testListarSalas() {
    Retorno r = miSistema.registrarSala("Sala 1", 100);
    assertEquals(Retorno.Resultado.OK, r.resultado);
    r = miSistema.registrarSala("Sala 2", 25);
    assertEquals(Retorno.Resultado.OK, r.resultado);
```

```
    r = miSistema.registrarSala("Sala 3", 200);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    // sala-capacidad#

    r = miSistema.listarSalas();

    assertEquals(Retorno.Resultado.OK, r.resultado);

    assertEquals("Sala 3-200#Sala 2-25#Sala 1-100", r.valorString);

}
```



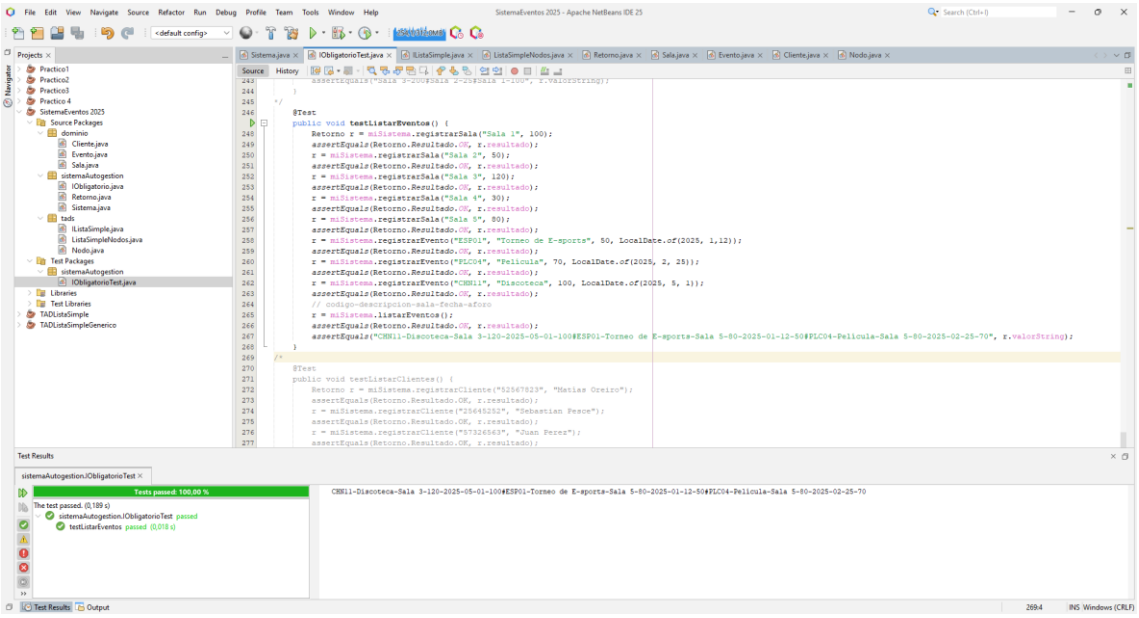
Requisito 2.2 – Listar Eventos – OK

```
@Test

public void testListarEventos() {

    Retorno r = miSistema.registrarSala("Sala 1", 100);

    assertEquals(Retorno.Resultado.OK, r.resultado);

    r = miSistema.registrarSala("Sala 2", 50);
```

```
        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarSala("Sala 3", 120);

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarSala("Sala 4", 30);

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarSala("Sala 5", 80);

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("ESP01", "Torneo de E-sports", 50,
LocalDate.of(2025, 1,12));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("PLC04", "Película", 70, LocalDate.of(2025, 2,
25));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        r = miSistema.registrarEvento("CHN11", "Discoteca", 100, LocalDate.of(2025,
5, 1));

        assertEquals(Retorno.Resultado.OK, r.resultado);

        // codigo-descripcion-sala-fecha-aforo

        r = miSistema.listarEventos();

        assertEquals(Retorno.Resultado.OK, r.resultado);
```

assertEquals("CHN11-Discoteca-Sala 3-120-2025-05-01-100#ESP01-Torneo de E-sports-Sala 5-80-2025-01-12-50#PLC04-Película-Sala 5-80-2025-02-25-70", r.valorString);

}



Requisito 2.3 – Listar clientes – OK

@Test

public void testListarClientes() {

Retorno r = miSistema.registrarCliente("52567823", "Matías Oreiro");

assertEquals(Retorno.Resultado.OK, r.resultado);

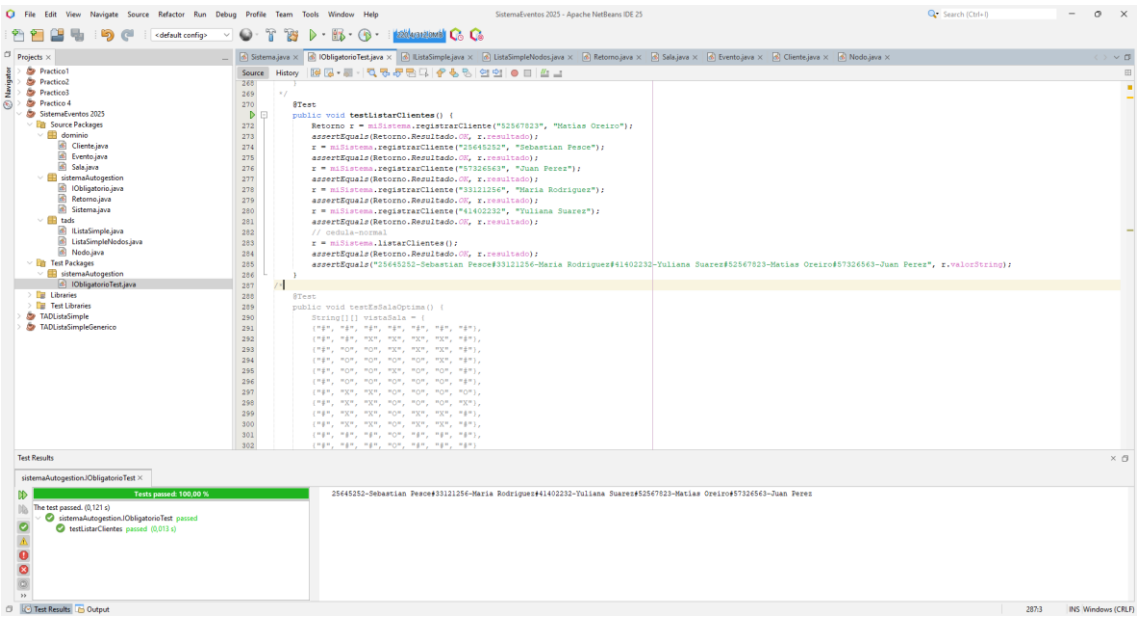
r = miSistema.registrarCliente("25645252", "Sebastian Pesce");

assertEquals(Retorno.Resultado.OK, r.resultado);

r = miSistema.registrarCliente("57326563", "Juan Perez");

assertEquals(Retorno.Resultado.OK, r.resultado);

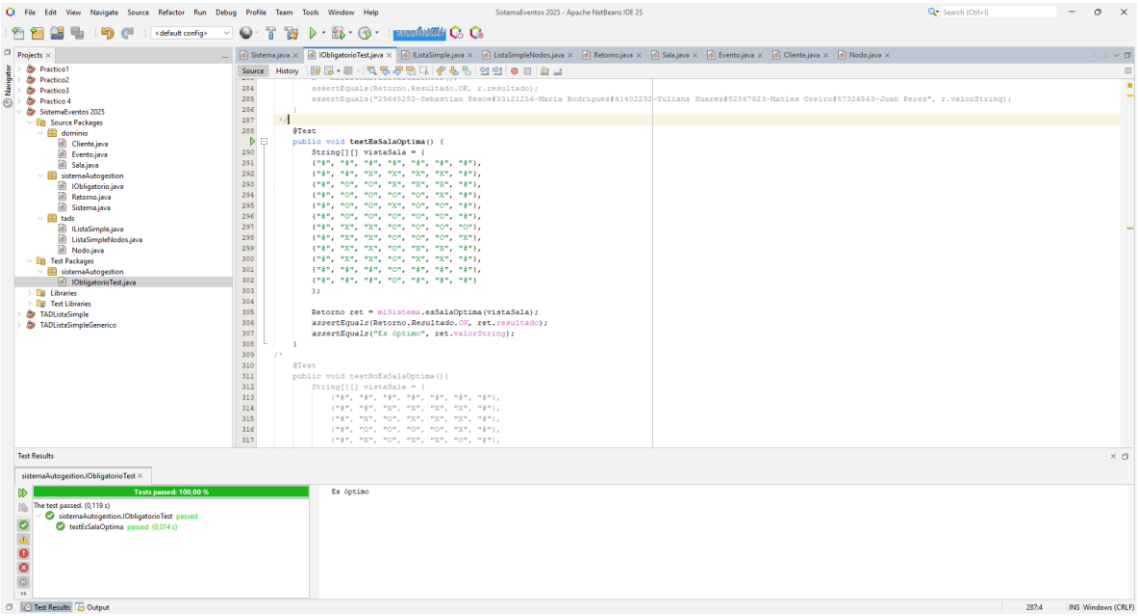
```
        r = miSistema.registrarCliente("33121256", "Maria Rodriguez");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        r = miSistema.registrarCliente("41402232", "Yuliana Suarez");
        assertEquals(Retorno.Resultado.OK, r.resultado);
        // cedula-nombre
        r = miSistema.listarClientes();
        assertEquals(Retorno.Resultado.OK, r.resultado);
        assertEquals("25645252-Sebastian Pesce#33121256-Maria Rodriguez#41402232-Yuliana Suarez#52567823-Matías Oreiro#57326563-Juan Perez", r.valorString);
    }
```



Requisito 2.4 – Es sala óptima – OK, es óptima

```
@Test

public void testEsSalaOptima() {

    String[][] vistaSala = {

    {"#", "#", "#", "#", "#", "#", "#"},

    {"#", "#", "X", "X", "X", "X", "#"},

    {"#", "O", "O", "X", "X", "X", "#"},

    {"#", "O", "O", "O", "O", "X", "#"},

    {"#", "O", "O", "X", "O", "O", "#"},

    {"#", "O", "O", "O", "O", "O", "#"},

    {"#", "X", "X", "O", "O", "O", "O"},

    {"#", "X", "X", "O", "O", "O", "X"},

    {"#", "X", "X", "O", "X", "X", "#"},

    {"#", "X", "X", "O", "X", "X", "#"},

    {"#", "#", "#", "O", "#", "#", "#"},

    {"#", "#", "#", "O", "#", "#", "#"}

      };



      Retorno ret = miSistema.esSalaOptima(vistaSala);
```

```
        assertEquals(Retorno.Resultado.OK, ret.resultado);

        assertEquals("Es óptimo", ret.valorString);

    }
```



Requisito 2.4 – Es sala óptima – OK, no es óptima

```
    @Test

    public void testNoEsSalaOptima(){

        String[][] vistaSala = {

            {"#", "#", "#", "#", "#", "#", "#"},

            {"#", "#", "X", "X", "X", "X", "#"},

            {"#", "X", "O", "X", "X", "X", "#"},

            {"#", "O", "O", "O", "O", "X", "#"},

            {"#", "X", "O", "X", "X", "O", "#"},

            {"#", "X", "X", "O", "X", "O", "#"},
```

```
            {"#", "X", "X", "X", "O", "O", "X"},
            {"#", "X", "X", "O", "O", "O", "X"},
            {"#", "X", "X", "O", "X", "X", "#"},
            {"#", "X", "X", "O", "X", "X", "#"},
            {"#", "#", "#", "X", "#", "#", "#"},
            {"#", "#", "#", "O", "#", "#", "#"}
        };

        Retorno ret = miSistema.esSalaOptima(vistaSala);
          assertEquals(Retorno.Resultado.OK, ret.resultado);
          assertEquals("No es óptimo", ret.valorString);
    }
}
```