

TALLER INGENIERÍA EN SOFTWARE  
Tarea Testing

## 1. Tarea

1. Crear un método de testing en la clase DayTest para verificar que findSpace retorne el valor 10 para una cita de una hora, si en el día ya se cuenta con una cita a las 9am.
2. Crear un test para verificar que el método findSpace retorna el valor de -1, si se hace un intento de agendar una cita en un día que esté lleno.
3. Crear un test negativo (usar la instrucción assertFalse) para el siguiente caso. Intentar inicializar una cita de un ahora y luego poner una cita de dos horas a la misma hora. Que sucede cuando corre el test? Incluir un screenshot en el reporte.
4. Hacer cambios al programa para que acepte citas de media hora. Testear.

## 2. Testing 1

```
@Test
//retorno 10 FindSpace
public void primerTest() {
    System.out.println("Este es el primer test-INICIO");
    day1.makeAppointment(9,a1);
    int diez = day1.findSpace(new Appointment("segunda cita",1));
    System.out.println("Siguiente hora libre ---> "+diez);
    day1.showAppointments();
    assertEquals(10, diez);
}
```

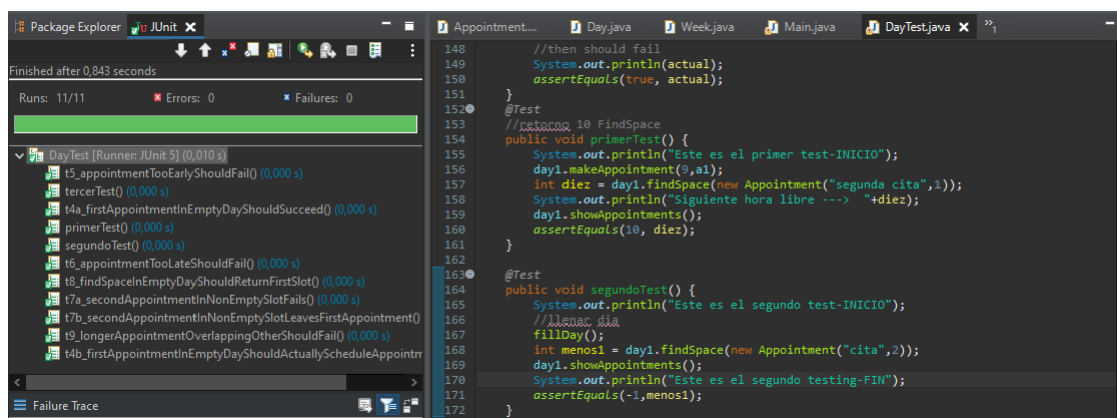
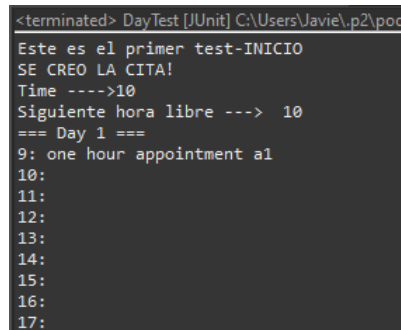


Figura 1: Código 1

En este primerTest tenemos primero un print para identificar cuando estemos ejecutando nuestro test, después llamamos a la función *makeAppointment()* esta lo que hace es agendar una cita, le pasamos los parámetros de hora y la cita(appointment) en este caso tomamos la cita ya inicializada en la función *setUp()* en donde ya se ha creado una cita con la duración de una hora y su descripción correspondiente, entonces esta cita la agendamos a la primera hora 9 osea 9 am. Luego utilizamos la función ya creada *findSpace()* la cual busca el espacio vacío mas cercano para agendar una cita, en caso de encontrarlo retorna el valor de la hora que debería agendarse. En este caso necesitamos que la función nos devuelva el valor de 10, ya que sería el espacio vacío mas cercano despues de la primera cita que inicializamos, por lo que esta función tiene como entrada una cita un *appointment* y le damos, entonces como parámetro le damos una nueva cita, con descripción "segunda citaz una duración de 1. Todo esto será almacenado en una variable llamada diez de valor entero. Para cerciorarnos si se creo la primera cita a las 9, llamamos a la función *showAppontments()*, la cual imprime en consola el día completo con las horas y si tienen o no citas agendadas. Finalmente tenemos el *assertEquals* en donde le entregamos como parámetros el numero 10 que sería nuestra salida esperada y para su comparación la variable *diez* en la cual guardamos el retorno de la función *findSpace*.



```

<terminated> DayTest [JUnit] C:\Users\Javie\p2\pool
Este es el primer test-INICIO
SE CREO LA CITA!
Time ---->10
Siguiete hora libre ----> 10
=== Day 1 ===
9: one hour appointment a1
10:
11:
12:
13:
14:
15:
16:
17:

```

Figura 2: Output test 1

### 3. Testing 2

```

@Test
public void segundoTest() {
    System.out.println("Este es el segundo test-INICIO");
    //llenar dia
    fillDay();
    int menos1 = day1.findSpace(new Appointment("cita",2));
    day1.showAppointments();
    System.out.println("Este es el segundo testing-FIN");
    assertEquals(-1,menos1);
}

```

El el segundo test , tenemos se ocupo la función *fillDay()* el cuál, crea un dia completo con citas a cada hora del día hasta el fin del horario. Despues creamos una variables *menos1* en la cuál vamos a guardar el valor que retorne *findSpace()*, por el comportamiento de esta función al estar el día completo con citas y no encontrar un espacio vacío debería retornar  $-1$ , por lo que en nuestro *asserEquals* pasamos el valor esperado en este caso  $-1$  y la variable que contiene el valor de retorno de la función *findSpace*.

```

<terminated> DayTest [JUnit] C:\Users\Javie\p2\pool\plugin
Este es el segundo test-INICIO
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
SE CREO LA CITA!
Devuelve --> -1
=== Day 1 ===
9: one hour appointment a1
10: a2
11: a3
12: a4
13: a5
14: a6
15: a7
16: a8
17: a9
Este es el segundo testing-FIN

```

Figura 3: Output test 2

## 4. Testing 3

```

@Test
public void tercerTest() {
    System.out.println("Este es el tercer test-INICIO");
    day1.makeAppointment(9,a1);
    Appointment segundacita = new Appointment("segundacita",2);
    day1.showAppointments();
    assertFalse(day1.makeAppointment(9,segundacita));
}

```

Para este test volvemos a usar el *a1* que esta inicializando una cita con duraci3n de 1 hora, ocupamos la funci3n *makeAppointment* en donde la agendamos a las 9, tambien creamos una *segundacita* con largo de dos horas. Entonces en nuestro *assertFalse* le pasamos como variable el retorno de la funcion *makeAppointment(9,segundacita)* osea agendando una segunda cita a la misma hora que la anterior. La cu3l nos dar3a el valor de *false*, la cual fue probada con el c3digo comentado.

Que sucede cuando corre el test? Bueno como vemos el test pasa bien sin ning3n error ni Failures, ya que como el valor de retorno de *makeAppointment* ser3 *false* ya que no se podr3 concretar agendar la cita a esa hora, al ser un *asserFalse* espera un valor de entrada falso si fuera de manera contraria y la entrada fuera *true* entonces tendr3amos un problema.

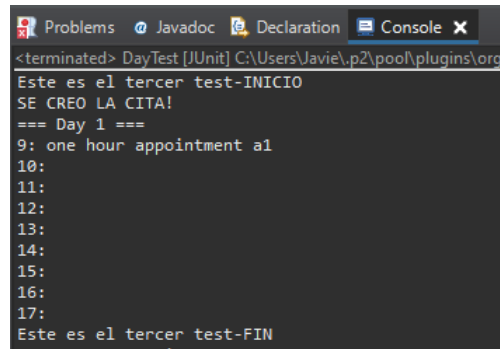
A screenshot of an IDE's console window. The window has tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, showing the output of a test. The text in the console is: '<terminated> DayTest [JUnit] C:\Users\Javie.p2\pool\plugins\org.', 'Este es el tercer test-INICIO', 'SE CREO LA CITA!', '=== Day 1 ===', '9: one hour appointment a1', '10:', '11:', '12:', '13:', '14:', '15:', '16:', '17:', 'Este es el tercer test-FIN'.

Figura 4: Output test 3

## 5. Testing 4

Primero que todo en nuestra clase APPOINTMENT.JAVA, CAMBIAMOS POR TIPO DOUBLE PARA LA VARIABLE DURATION.

```
public class Appointment{
    private String description;
    private double duration;
    public Appointment(String description, double duration)
    {
        this.description = description;
        this.duration = duration;
    }
}
```

DESPUÉS EN NUESTRA CLASE DAY.JAVA, TAMBIEN CAMBIAMOS NUESTROS METODOS Y VARIABLES A DOUBLE, COMO TAMBIÉN EN ALGUNOS CASOS EN DONDE USAMOS INDICES PARA APPOINTMENT[] DEBEMOS TRASFORMAR NUESTROS VALORES DOUBLE EN INT PARA PODER TENER LOS ÍNDICES.

```

public double findSpace(Appointment appointment)
{
    double duration = appointment.getDuration();
    for(int slot = 0; slot < MAX_APPOINTMENTS_PER_DAY; slot++) {
        if(appointments[slot] == null) {
            final double time = START_OF_DAY + slot;
            // Potential start point.
            if(duration == 0.5) {
                // Only a single slot needed.
                System.out.println("Time ---->" + time);
                return time;
            }
        }
        else {
            // How many more slots are needed?
            double further_slots_required = duration - 1;
            for(int nextSlot = slot + 1;
                further_slots_required > 0 &&
                appointments[nextSlot] == null;
                nextSlot++) {
                further_slots_required--;
            }
            if(further_slots_required == 0) {
                // A big enough space has been found.
                return time;
            }
        }
    }
    // Not enough space available.
    //se pone un print para tener más claro
    System.out.println("Devuelve ---> -1");
    return -1;
}
.
.
.

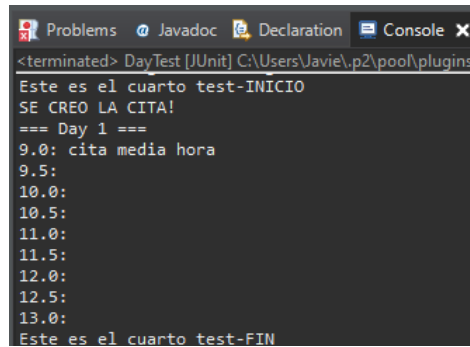
```

Y POR LO TANTO NUESTRO TEST SERIA

```

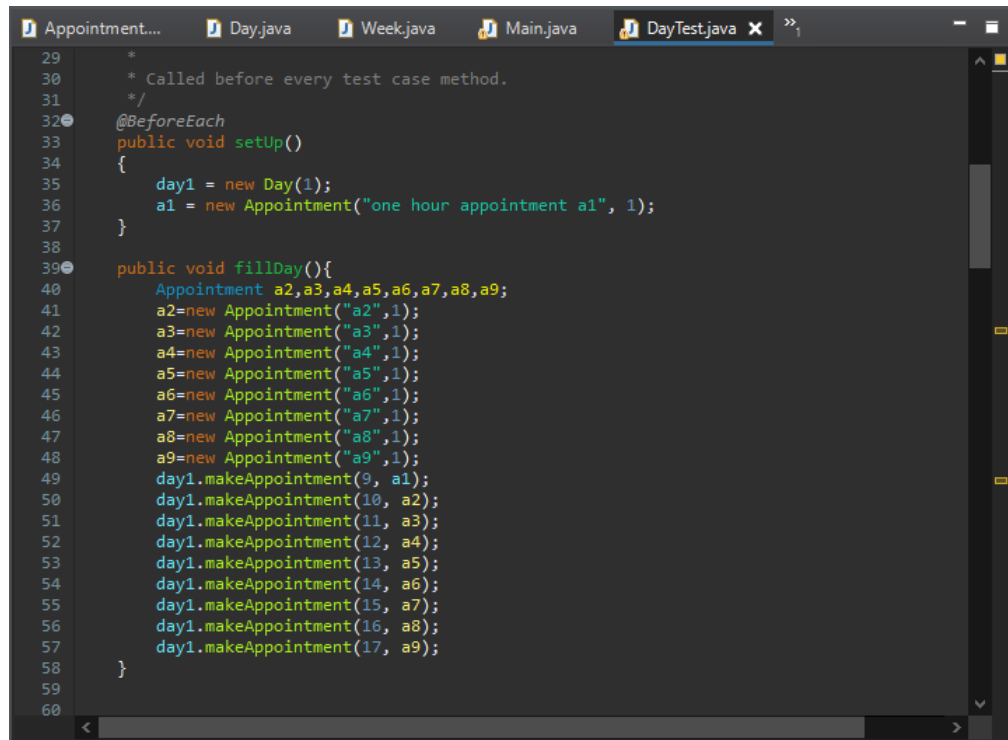
@Test
public void cuartoTest() {
    System.out.println("Este es el cuarto test-INICIO");
    Appointment citamedia = new Appointment("cita media hora",0.5);
    boolean prueba = day1.makeAppointment(9,citamedia);
    assertEquals(true,prueba);
    day1.showAppointments();
    System.out.println("Este es el cuarto test-FIN");
}

```



```
Problems Javadoc Declaration Console
<terminated> DayTest [JUnit] C:\Users\Javie\p2\pool\plugins
Este es el cuarto test-INICIO
SE CREO LA CITA!
=== Day 1 ===
9.0: cita media hora
9.5:
10.0:
10.5:
11.0:
11.5:
12.0:
12.5:
13.0:
Este es el cuarto test-FIN
```

Figura 5: Output test 4



```
Appointment... Day.java Week.java Main.java DayTest.java x »1
29      *
30      * Called before every test case method.
31      */
32  @BeforeEach
33  public void setUp()
34  {
35      day1 = new Day(1);
36      a1 = new Appointment("one hour appointment a1", 1);
37  }
38
39  public void fillDay(){
40      Appointment a2,a3,a4,a5,a6,a7,a8,a9;
41      a2=new Appointment("a2",1);
42      a3=new Appointment("a3",1);
43      a4=new Appointment("a4",1);
44      a5=new Appointment("a5",1);
45      a6=new Appointment("a6",1);
46      a7=new Appointment("a7",1);
47      a8=new Appointment("a8",1);
48      a9=new Appointment("a9",1);
49      day1.makeAppointment(9, a1);
50      day1.makeAppointment(10, a2);
51      day1.makeAppointment(11, a3);
52      day1.makeAppointment(12, a4);
53      day1.makeAppointment(13, a5);
54      day1.makeAppointment(14, a6);
55      day1.makeAppointment(15, a7);
56      day1.makeAppointment(16, a8);
57      day1.makeAppointment(17, a9);
58  }
59
60
```

Figura 6: Función setUp() y fillDay()