Práctica 6. **El Hospital**

1 OBJETIVO **3**

2 ENUNCIADO **4**

3 PRIMERA PARTE **5**

3.1 Esbozar las clases principales 5

3.2 Tests del método RequiereAislamiento() 6

3.3 Tests del método ApellidosComaNombre() 12

4 SEGUNDA PARTE **14**

4.1 Tests del método RellenarDesdeLinea() 15

4.2 Tests del método Ingresar() 16

1 OBJETIVO

Un caso de prueba pretende certificar si el sujeto bajo tests hace lo que debe. La prueba debe arrojar únicamente dos posibles resultados: SI o NO, (verde o rojo, OK o NOK).

En esta práctica se implementarán varias pruebas de “caja negra”. Las pruebas de “caja negra” centran su interés en lo que hace el método no en cómo lo hace. Para ello el método o función es alimentado con un conjunto de valores de entrada, la prueba certifica que los resultados devueltos por el método son los que deben. Es indiferente el código ejecutado para lograr los resultados. Cada una de los pares “entrada” - ”salida esperada” se llama caso de prueba o *“test case”*.

Esta práctica pretende mostrar cómo el Desarrollo Guiado por Pruebas permite:

- ❏ *Hacer emerger el diseño*. El programador ejercita su código. Al poner a prueba su implementación el programador sufre las consecuencias de su diseño aflorando oportunidades de mejora.
- ❏ *Documentar el código*. Los tests escritos facilitarán a otros programadores entender el propósito del código implementado. Es, además, una documentación viva, permanentemente actualizada.
- ❏ *Refactorizar el código*. El programador cuenta con una red de seguridad que le permite limpiar el código haciéndolo más mantenible o extensible asegurándose que los cambios realizados no han alterado el comportamiento esperado.
- ❏ *Detectar regresiones o errores*. Si el programador introduce un error, será informado con una alerta al romperse algún test.

Para ello la práctica propone un ejercicio en el que se deberá codificar una batería de pruebas para testar el código escrito. En la primera parte del ejercicio el alumno seguirá un guión para, paso a paso, entender cómo funcionan las pruebas de código. En la segunda parte el alumno deberá, de forma más autónoma y menos guiada, completar el ejercicio.

2 ENUNCIADO

Las residencias y otros centros sociosanitarios envían diariamente al servicio de admisión hospitalaria, los pacientes que serán ingresados en un fichero de texto. Cada paciente es informado con el nombre, apellido, fecha de nacimiento y síntomas.

Los posibles síntomas son:

- ❑ **Tos seca**
- ❑ **Cansancio**
- ❑ **Fiebre**
- ❑ Náuseas
- ❑ Diarrea

Los tres primeros síntomas pueden ser producidos por el COVID-19. Si el paciente tiene dos o más síntomas COVID-19 y tiene más de 70 años requiere aislamiento.

El fichero de texto enviado por la residencias indica en cada línea un paciente con el siguiente formato:

```
nombre1|apellido1|fechanacimiento|sintoma1,sintoma2,sintoma3
```

Escriba una aplicación de consola que lea el fichero y muestre en pantalla si el paciente requiere aislamiento de la siguiente manera: "`apellido1, nombre AISLAMIENTO`"

Por ejemplo, el fichero de texto:

```
Mikel|Martinez|01/01/2000|Tos seca,Nauseas

Fermin|Navarro|01/01/1940|Cansancio,Fiebre,Nauseas
```

Hará mostrar en la consola:

```
Navarro, Fermin AISLAMIENTO
```

3 PRIMERA PARTE

3.1 Esbozar las clases principales

Para dar solución a este problema haciendo uso de la Programación Orientada a Objetos encapsule en una clase los atributos del paciente así como la lógica que razona con dichos atributos para calcular si el paciente requiere aislamiento y para mostrar su apellido y nombre separado por comas:

```
public class Paciente {

    public String Nombre;

    public String Apellido1;

    public LocalDate FechaNacimiento;

    public String[] Sintomas;

    public Paciente() {

        this.Nombre = "";

        this.Apellido1  = "";

        this.FechaNacimiento = null;

        this.Sintomas = new String[0];

    }

    public Boolean RequiereAislamiento(){

        return false

    }

    public String ApellidosComaNombre(){

        return "";

    }

}
```

Por ahora no es necesario que implemente los métodos RequireAislamiento() y ApellidosComaNombre(), limítese a devolver un valor, el que desee.

La clase ServicioAdmision hará uso de la entidad “Paciente”:

```
public class ServicioAdmision {

    public String Ingresar(String fichero){

        ArrayList<Paciente> pacientes = new ArrayList<Paciente>();

        pacientes.add(new Paciente("Mikel","Martinez",LocalDate.of(2001,1,1),new
String[]{"Fiebre"}));

        pacientes.add(new Paciente("Mikel","Martinez",LocalDate.of(1940,1,1),new
String[]{"Tos seca","Fiebre"}));

        StringBuilder respuesta = new StringBuilder();

        for(Paciente paciente : pacientes){

            if(paciente.RequiereAislamiento()){

                respuesta.append(paciente.ApellidosComaNombre() + " AISLAMIENTO\n");

            }

        }

        return respuesta.toString();

    }

}
```

Por ahora no es necesario implementar la lógica que lee el fichero de texto.

La clase ejecutora se limitará a invocar el método ingresar del objeto ServicioAdmision e imprimirá su resultado en la consola:

```
public static void main(String[] args) {

        System.out.println(new ServicioAdmision().Ingresar("pacientes.txt"));

    }
```

Escriba el código, compile, ejecute y observe el resultado. Ya tiene esbozada la solución.

3.2 Tests del método RequiereAislamiento()

Estos son los casos de prueba (tests case) que debe codificar:

Entrada		Salida
Fec Nac	**Síntomas**	**Requiere Aislamiento**
01/01/1940	{“Tos Seca”, “Cansancio”}	true
01/01/1940	{“Tos Seca”, “Diarrea”, “Náuseas”}	false
01/01/1940	{“Fiebre”, “Cansancio”, “Diarrea”}	true
01/01/1940	{“Cansancio”, “Diarrea”}	false
01/01/2001	{“Tos Seca”, “Cansancio”}	false
01/01/2001	{“Tos Seca”, “Cansancio”, “Fiebre”}	true
null	{“Tos Seca”, “Cansancio”, “Fiebre”}	false
01/01/1940	{}	false
null	{}	false

Los 6 primeros casos de prueba se denominan “*happy path*”, muestran el comportamiento que se espera del código en situaciones normales. Los 3 últimos casos de prueba se denominan “*bad path*” y muestran el comportamiento esperado del código es situaciones extremas.

Escriba una nueva clase ejecutora llamada “TestsAislamiento”. Esta clase instanciará los dos primeros pacientes de pruebas para los que deberá verificar que el cálculo del requerimiento de aislamiento es correcto.

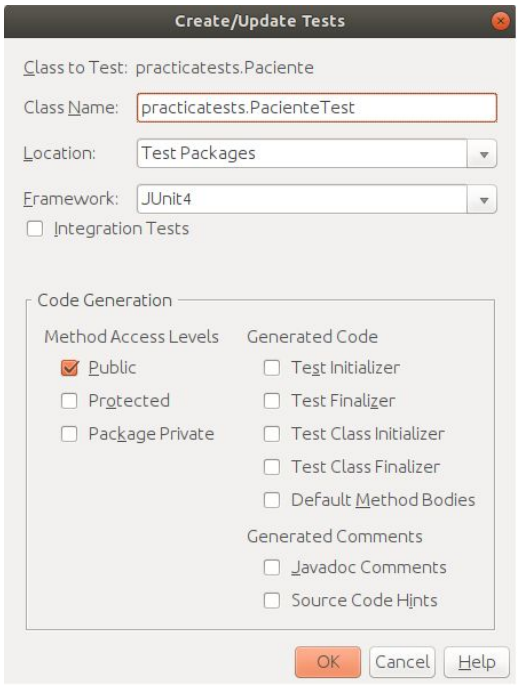
```
public static void main(String[] args) {

        // TEST1

        // Dado un paciente de más de 70 años que tiene dos sintomas COVID19

        Paciente paciente1 = new Paciente("Mikel", "Martinez", LocalDate.of(1940,1,1), new
String[]{"Tos seca", "Cansancio"});

        // Cuando chequeo si requiere o no aislamiento

        Boolean requiereAislamientoPaciente1 = paciente1.RequiereAislamiento();

        // Entonces debe devolver cierto

        if(requiereAislamientoPaciente1) System.out.println("OK");

        else System.out.println("NOK");
```

```
        // TEST2
        // Dado un paciente de más de 70 años que tiene solo un sintoma COVID19
        Paciente paciente2 = new Paciente("Mikel", "Martinez", LocalDate.of(1940,1,1), new
String[]{"Tos seca", "Diarrea", "Nauseas"});
        // Cuando chequeo si requiere o no aislamiento
        Boolean requiereAislamientoPaciente2 = paciente2.RequiereAislamiento();
        // Entonces debe devolver cierto
        if(!requiereAislamientoPaciente2) System.out.println("OK");
        else System.out.println("NOK");
    }
```

Observe como el método RequiereAislamiento() no pasa las pruebas.

Codificar los casos de prueba en una aplicación de consola resulta farragoso. Por suerte contamos con la librería JUnit que facilita mucho la codificación de las pruebas. Además netbeans ofrece un interface gráfico para visualizar los resultados de las pruebas de forma mucho más agradable.

Seleccione la clase Paciente.java del proyecto. Abra el menú contextual (botón derecho) y elija Tools -> Create/Update Tests. Seleccione JUnit 4 y desactive todas la opciones excepto “Public” en “method access level”. Observe como netbeans es capaz de autogenerar los “cimientos” del código de pruebas en una nueva clase en el paquete de Tests.



Para cada caso de prueba debe escribir un método etiquetado con *@Test.* Utilice *assertTrue* o *assertFalse* para implementar la verificación del resultado. Para ver la ventana de pruebas de netbeans elija Windows -> IDE tools → Test Result. Para lanzar las pruebas seleccione el archivo de pruebas y elija “Test File” en el menú contextual o simplemente pulse CTRL + F6.

A modo de ejemplo se muestra la codificación de los dos primeros casos de prueba:

```
@Test
public void Un_paciente_mayor_con_dos_sintomas_covid_requiere_aislamiento() {
    // Dado un paciente de más de 70 años que tiene dos sintomas COVID19
    Paciente paciente = new Paciente();
    paciente.FechaNacimiento = LocalDate.of(1940,1,1);
    paciente.Sintomas = new String[]{"Tos seca", "Cansancio"};
    // Cuando chequeo si requiere o no aislamiento
    Boolean requiereAislamiento = paciente.RequiereAislamiento();
    // Entonces debe devolver cierto
    assertTrue(requiereAislamiento);
}
```

```
@Test
public void Un_paciente_mayor_con_un_sintoma_covid_no_requiere_aislamiento() {
    // Dado un paciente de más de 70 años que tiene un sintoma COVID19
    Paciente paciente = new Paciente();
    paciente.FechaNacimiento = LocalDate.of(1940,1,1);
    paciente.Sintomas = new String[]{"Tos seca", "Diarrea", "Nauseas"};
    // Cuando chequeo si requiere o no aislamiento
    Boolean requiereAislamiento = paciente.RequiereAislamiento();
    // Entonces debe devolver cierto
    assertFalse(requiereAislamiento);
}
```

> Las tres A's del Unit Testing:
> 1. Arrange (organizar). Se establecen los valores de los parámetros de entrada y se instancia el objeto o sujeto a testar (SUT).
> 2. Act (actuar). Se desencadena el código objeto del test ejecutando un método del SUT.
> 3. Assert (afirmar). Se comprueba que el resultado obtenido es el esperado.

Observe cómo la fase "Arrange" establece los valores de los parámetros de entrada e instancia el SUT (sujeto bajo test), en este caso el objeto paciente. La fase "Act" desencadena el código que se desea probar, en este caso RequiereAislamiento(). Por último la fase "Assert" ejecuta las verificaciones necesarias comprobando que lo obtenido es lo esperado. Advierta el uso de notación "Gerkhin" (Dado-Cuando-Entonces) para ayudar a documentar el test.

Escriba el resto de casos de prueba.

Ya esta preparado para escribir el código del método RequiereAislamiento().

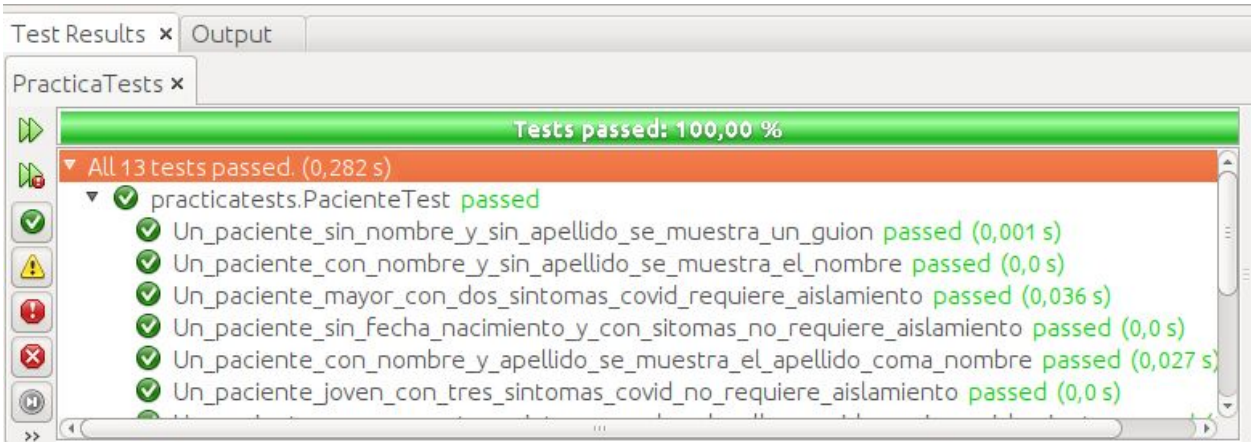
```
public Boolean RequiereAislamiento(){
        LocalDate ahora = LocalDate.now();
        Period periodo = Period.between(this.FechaNacimiento, ahora);
```

```
        int edad = periodo.getYears();

        if(edad < 70) return false;

        int numeroSintomasCovid = 0;

        for(String sintoma : Sintomas){

           if(sintoma.equals("Tos seca") || sintoma.equals("Fiebre") ||
sintoma.equals("Cansancio"))

                numeroSintomasCovid++;

        }

        if(numeroSintomasCovid>= 2) return true;

        return false;

    }
```

Ejecute ahora la pruebas. El código ha pasado casi todos los casos de prueba. Los tests que están fallando se deben a que el programador ha “olvidado” contemplar la fecha de nacimiento con posible valor nulo.

> Los tests detectan bugs. Ayudan al programador a escribir código que contemple todos los casos, también los casos extremos.

Añada una línea de código para chequear si la fecha de nacimiento es nula. Vuelva a lanzar los test. Ahora pasan todos.



> Los tests documentan el código. Leyendo los nombre de los métodos de los tests se entiende el propósito del código.

El código del método RequiereAislamiento() puede ser mejorado: escriba dos métodos privados, uno para calcular la edad y otro para calcular el número de síntomas covid. Modifique RequiereAislamiento() para hacer uso de ambos métodos privados. Lance las pruebas y observe que el código sigue funcionando.

```
public Boolean RequiereAislamiento(){

        if(this.FechaNacimiento==null) return false;

        return Edad()>=70 && SintomasCOVID()>=2;

    }
```

> Los tests actúan como una red de seguridad para el programador cuando se desea refactorizar un segmento de código.

3.3 Tests del método ApellidosComaNombre()

Estos son las casos de prueba que debe codificar:

Entrada		Salida
Nombre	**Apellido1**	**Apellido, Nombre**
“Mikel”	“Martinez”	“Martinez, Mikel”
“”	“Martinez”	“Martinez”
“Mikel”	“”	“Mikel”
“”	“”	“_”

Primero codifique los casos de prueba haciendo uso de JUnit.

```
@Test
```

```
public void Un_paciente_con_nombre_y_apellido_se_muestra_con_el_apellido_coma_nombre() {

        // Dado un paciente con nombre y apellido

        Paciente paciente = new Paciente();

        paciente.Nombre = "Mikel";

        paciente.Apellido1 = "Martinez";

        // Cuando calculo su nombre descriptivo

        String nombreDescriptivo = paciente.ApellidosComaNombre();

        // Entonces debe ser apellido, nombre

        assertEquals(nombreDescriptivo, "Martinez, Mikel");

    }
```

Escriba el resto de casos de prueba.

Modifique el código del método ApellidosComaNombre(). Lance las pruebas para certificar que el código hace lo que debe.

4 SEGUNDA PARTE

Ahora debe escribir el código encargado de leer el fichero de texto, convertir cada línea del fichero en un objeto “Paciente” y procesar cada paciente mostrando en pantalla si requiere aislamiento.

Modifique el método Ingresar() de la clase ServicioAdmision para leer el fichero y “parsear” sus líneas en objetos “Paciente”.

```
public String Ingresar(String fichero){

        ArrayList<Paciente> pacientes = new ArrayList<Paciente>();

        try {

           BufferedReader reader = new BufferedReader(new FileReader(new File(fichero)));

           String linea;

           while((linea=reader.readLine())!=null){

               // Parsear la línea a un objeto Paciente:

               String[] partes = linea.split("\\|",-1);

               String[] sintomas = partes[3].split(",");

               String nombre = partes[0];

               String apellido =partes[1];

               LocalDate fecha =
LocalDate.parse(partes[2],DateTimeFormatter.ofPattern("dd/MM/yyyy"));

               pacientes.add(new Paciente(nombre,apellido,fecha,sintomas));

            }

        }

        catch(Exception e){

        }

        StringBuilder respuesta = new StringBuilder();

        for(Paciente paciente : pacientes){

            if(paciente.RequiereAislamiento()){
```

```
                respuesta.append(paciente.ApellidosComaNombre() + " AISLAMIENTO\n");

            }

        }

        return respuesta.toString();

    }
```

4.1 Tests del método RellenarDesdeLinea()

Para probar que el “parseo” de la línea del fichero funciona correctamente deberá codificar los siguientes casos de prueba.

Entrada	Salida
Mikel|Martinez|01/01/2001|Tos seca,Nauseas	objeto Paciente con dos síntomas
Fermin|Navarro|01/01/1940|Cansancio,Fiebre,Nauseas	objeto Paciente con tres síntomas
Aitor|Menta|01/01/2001|	objeto Paciente sin síntomas
Aitor|Menta||Tos seca	objeto Paciente sin nacimiento
|||	objeto Paciente vacío

Sin embargo el código propuesto le dificulta mucho escribir las pruebas. Para facilitar la testabilidad del código conviene extraer el código de “parseo” del método “Ingresar”:

```
String[] partes = linea.split("\\|",-1);

String[] sintomas = partes[3].split(",");

String nombre = partes[0];

String apellido =partes[1];

LocalDate fecha = LocalDate.parse(partes[2],DateTimeFormatter.ofPattern("dd/MM/yyyy"));

pacientes.add(new Paciente(nombre,apellido,fecha,sintomas));
```

Para ello cree un nuevo método en la clase “Paciente” llamado RellenarDesdeLinea(String linea). El método debe recibir como parámetro de entrada cada una de las líneas del fichero de

texto y poblar los atributos del objeto de acuerdo al convenio. Por ahora sencillamente deje el método vacío.

```
public void RellenarDesdeLinea(String linea){

        // poblar los atributos del objeto con los valores de la linea

    }
```

Codifica todos los casos de pruebas indicados.

Escriba el método RellenarDesdeLinea(). Lance los tests y compruebe que todos los casos de prueba pasan. Haga uso del método RellenarDesdeLinea() en el método Ingresar de ServicioAdmision. Observe como el código ha ganado en claridad.

> Los tests exigen al programador escribir código testable lo cual ayuda a un mejor diseño.

4.2 Tests del método Ingresar()

Escriba dos ficheros de pruebas: un fichero con 3 líneas de pacientes y otro fichero vacío, sin ninguna línea. Calcule cuál debe ser la respuesta esperada para cada fichero. Codifique los tests de estos casos de prueba. Tendrá que crear un nuevo fichero *ServicioAdmisionTest*. Asegure que todos los tests pasan.

Estos casos de pruebas corresponden a tests de integración ya que para su realización es necesario “ensamblar” elementos de infraestructura como ficheros de texto.

Han sido necesarios únicamente dos tests de integración para probar que toda la aplicación funciona, gracias a los más de 10 tests unitarios escritos. Por ejemplo, no es necesario escribir la línea “Mikel|||” en un fichero para saber que se procesará adecuadamente gracias a los tests unitarios escritos. El test de integración certifica únicamente que se procesan todas las líneas del fichero, el resto de funcionalidad de la solución ha sido probada de forma unitaria.