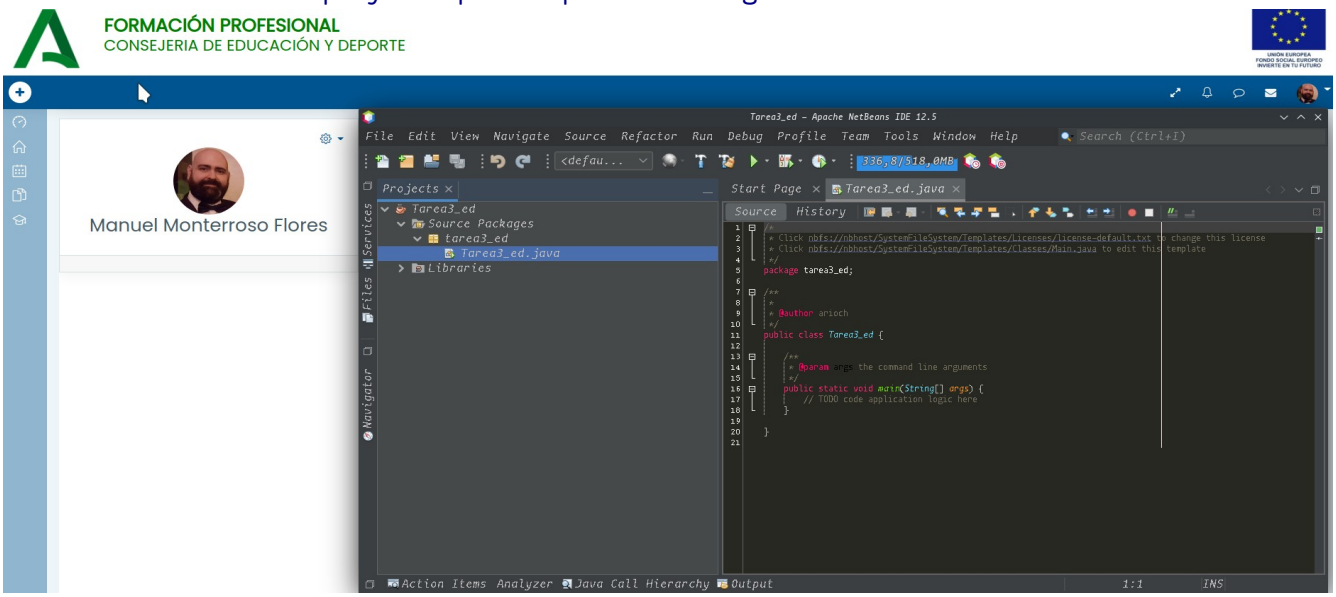
Manuel Monterroso Flores
Tarea 03

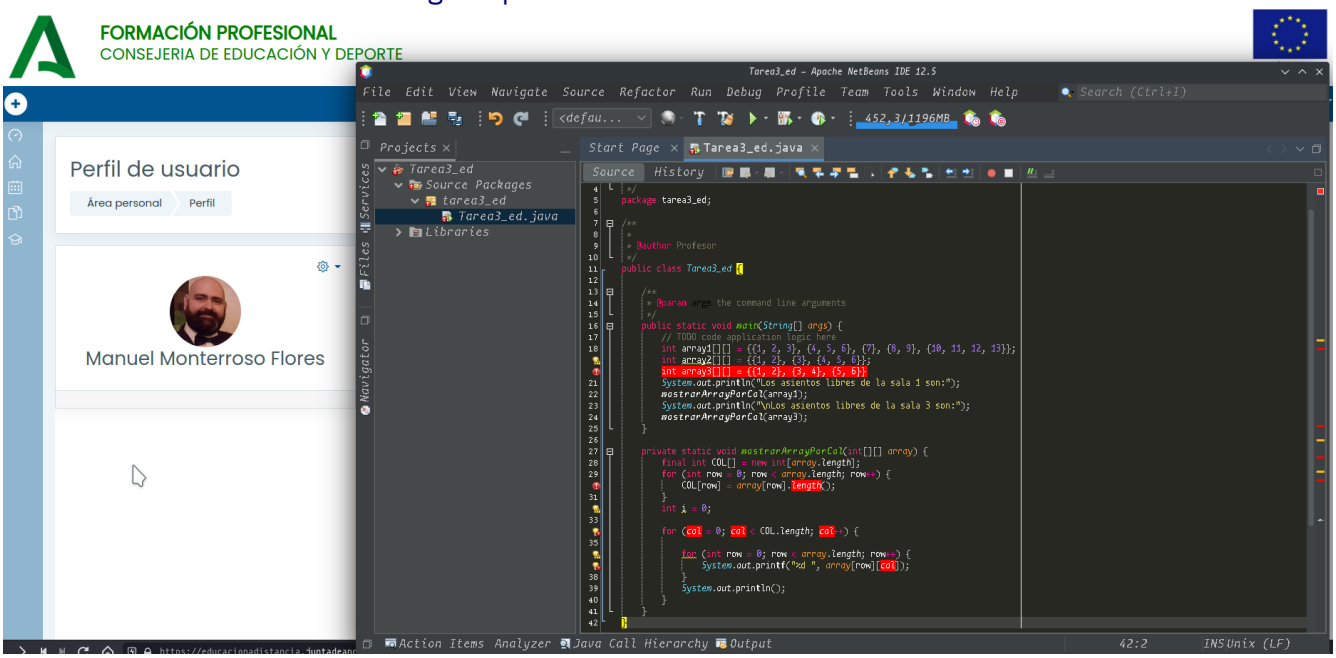
Pragmatic Unit Testing in Java with JUnit

1. Descarga el código, crea un proyecto y soluciona los pequeños errores de compilación que contiene.

- Creación del proyecto para copiar el código.



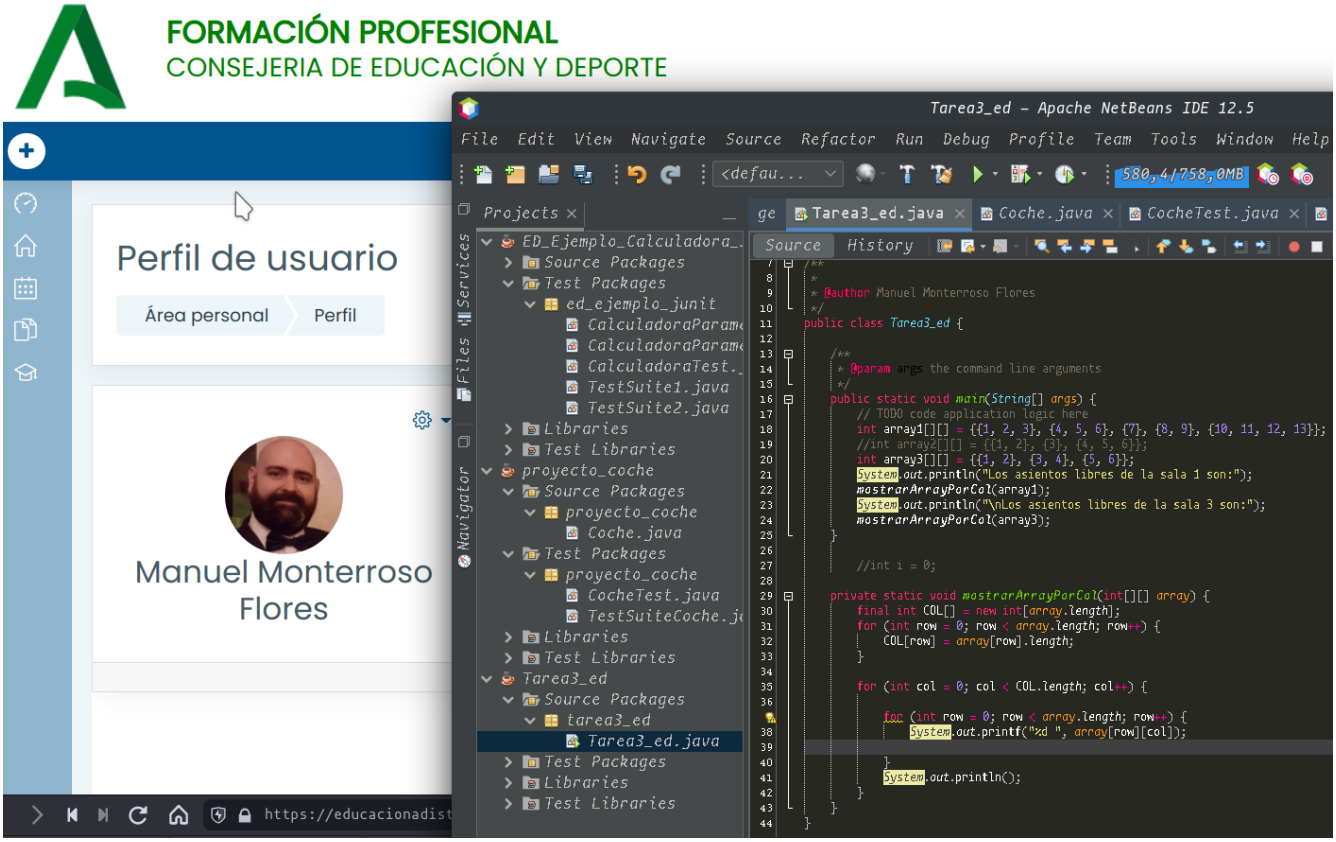
- NetBeans con el código copiado.



Los errores que aparecen son los siguientes:

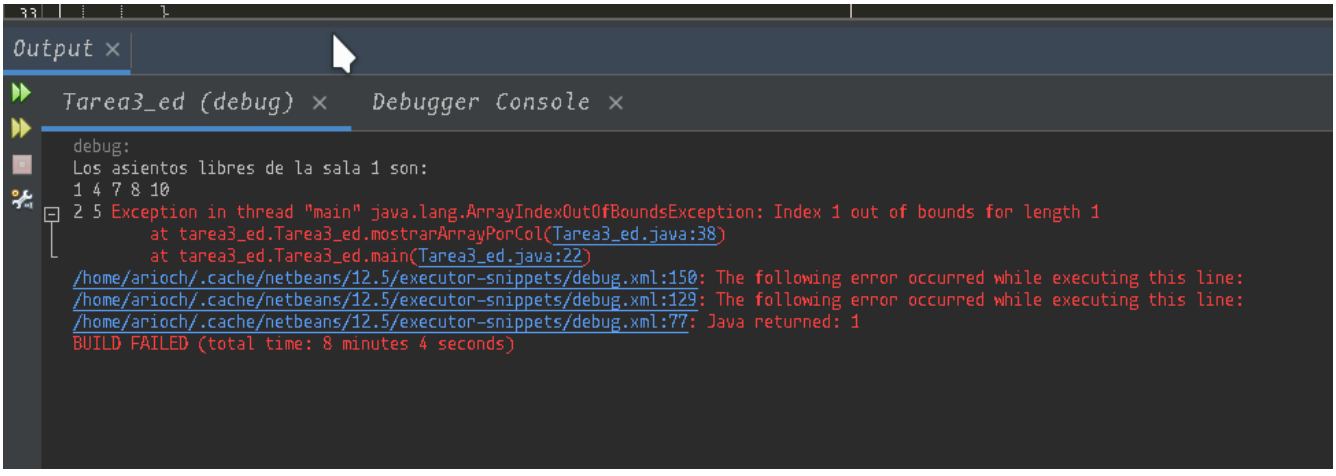
- En la línea 18 falta el “;” que debe de ir al final de la línea.
- En la línea 19 hay un array que no se usa por tanto lo comento para que no ocupe memoria.
- En la línea 30 después del lenght hay que quitar los paréntesis.
- En la línea 32 hay una declaración de variable que no se usa y la comento para que no ocupe memoria.
- En la línea 35 col no tiene definido el tipo de variable, lo defino poniendo “int” ya que es un número entero.

-NetBeans con los errores corregidos.



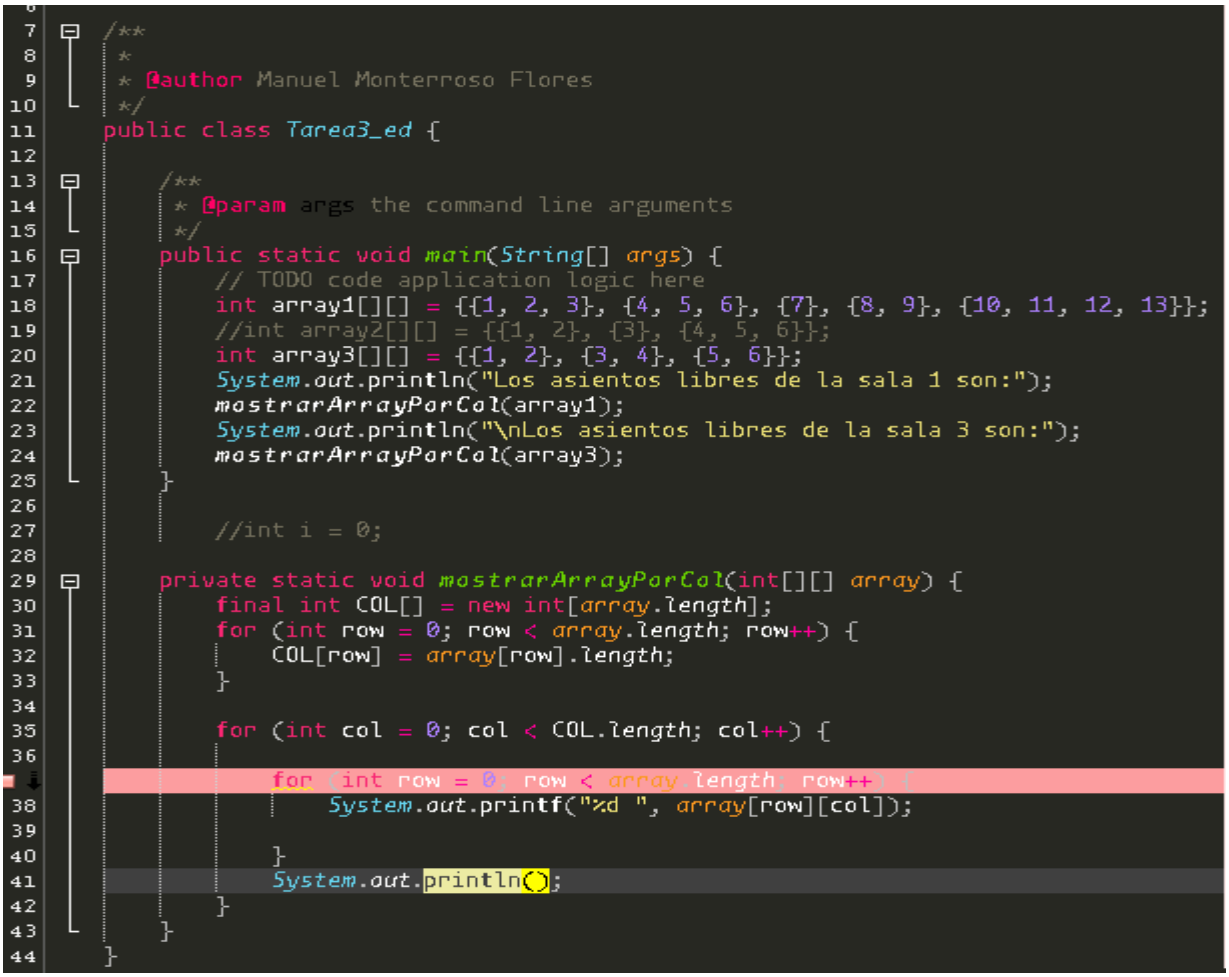
2. Corrige todos los errores lógicos y en un tiempo de ejecución, evitando posibles excepciones, utilizando breakpoints y el depurador de NetBeans (incluyendo sus herramientas y el examinador de variables).

Lo primero será ejecutar el programa y ver los errores que nos salen.



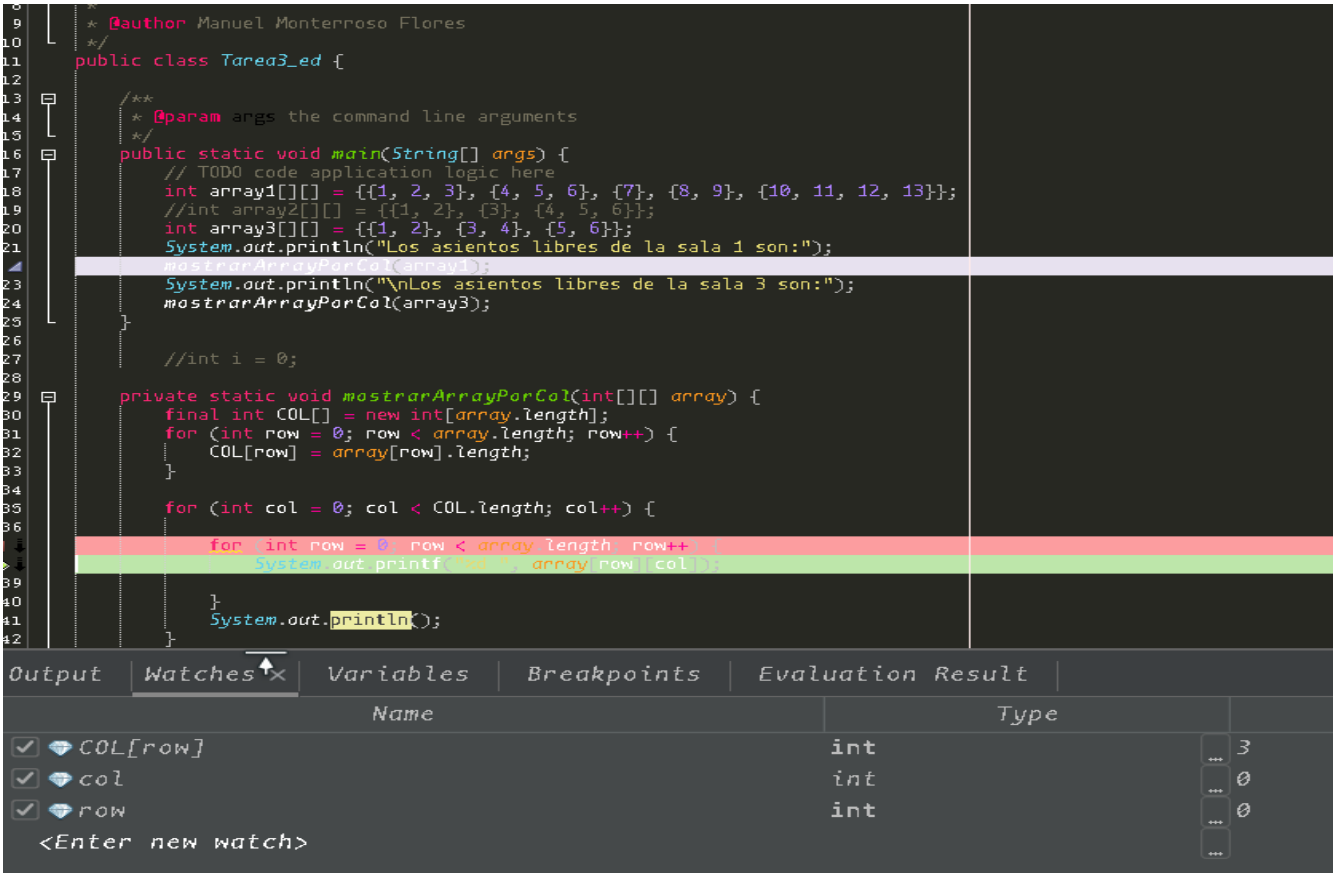
Como vemos indica un problema de array por longitud y señala las líneas 22 y 38 así que decido colocar un punto de breackpoint en la línea 37 y creo whatches sobre COL[row], col y row, para ir comparando las longitudes.

– NetBeans con el breakpoint puesto.



```
/**
 *
 * @author Manuel Monterroso Flores
 */
public class Tarea3_ed {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int array1[][] = {{1, 2, 3}, {4, 5, 6}, {7}, {8, 9}, {10, 11, 12, 13}};
        //int array2[][] = {{1, 2}, {3}, {4, 5, 6}};
        int array3[][] = {{1, 2}, {3, 4}, {5, 6}};
        System.out.println("Los asientos libres de la sala 1 son:");
        mostrarArrayPorCol(array1);
        System.out.println("\nLos asientos libres de la sala 3 son:");
        mostrarArrayPorCol(array3);
    }

        //int i = 0;

    private static void mostrarArrayPorCol(int[][] array) {
        final int COL[] = new int[array.length];
        for (int row = 0; row < array.length; row++) {
            COL[row] = array[row].length;
        }

        for (int col = 0; col < COL.length; col++) {

            for (int row = 0; row < array.length; row++) {
                System.out.printf("%d ", array[row][col]);

            }
            System.out.println();
        }
    }
}
```

- Whatches creados.



- En esta captura podemos apreciar el error antes de que salte, que es que como row tiene el valor 2 intenta sacar la posición 1 de la posición array con longitud COL[row] = 1 y como este array solo tiene un número (7) en la posición 0, pues al intentar acceder a la posición 1 esta no existe y se produce el error.

```
9   * @author Manuel Monterroso Flores
10  */
11  public class Tarea3_ed {
12
13      /**
14       * @param args the command line arguments
15       */
16      public static void main(String[] args) {
17          // TODO code application logic here
18          int array1[][] = {{1, 2, 3}, {4, 5, 6}, {7}, {8, 9}, {10, 11, 12, 13}};
19          //int array2[][] = {{1, 2}, {3}, {4, 5, 6}};
20          int array3[][] = {{1, 2}, {3, 4}, {5, 6}};
21          System.out.println("Los asientos libres de la sala 1 son:");
22          mostrarArrayPorCol(array1);
23          System.out.println("\nLos asientos libres de la sala 3 son:");
24          mostrarArrayPorCol(array3);
25      }
26
27          //int i = 0;
28
29      private static void mostrarArrayPorCol(int[][] array) {
30          final int COL[] = new int[array.length];
31          for (int row = 0; row < array.length; row++) {
32              COL[row] = array[row].length;
33          }
34
35          for (int col = 0; col < COL.length; col++) {
36
37              for (int row = 0; row < array.length; row++) {
38                  System.out.printf("%d ", array[row][col]);
39
40              }
41              System.out.println();
42          }
```

Output | Watches × | Variables | Breakpoints | Evaluation Result

Name	Type	
COL[row]	int	1
col	int	1
row	int	2
<Enter new watch>		

– En esta captura podemos observar cual sería la solución de este programa y es poner dentro del segundo for un if que controle que solo se pueda acceder al array siempre que col sea menor que la longitud del array.

```
for (int col = 0; col < COL.length; col++) {

    for (int row = 0; row < array.length; row++) {
        if(col < COL[row]){
        System.out.printf("%d ", array[row][col]);
        }
    }
    System.out.println();
}
```

– En esta captura vemos la solución junto con la salida del programa.

```
    private static void mostrarArrayPorCol(int[][] array) {
        final int COL[] = new int[array.length];
        for (int row = 0; row < array.length; row++) {
            COL[row] = array[row].length;
        }

        for (int col = 0; col < COL.length; col++) {

            for (int row = 0; row < array.length; row++) {
                if(col < COL[row]){
                System.out.printf("%d ", array[row][col]);
                }
            }
            System.out.println();
        }
```

```
tput ×

Debugger Console ×   Tarea3_ed (run) ×

run:
Los asientos libres de la sala 1 son:
1 4 7 8 10
2 5 9 11
3 6 12
13


Los asientos libres de la sala 3 son:
1 3 5
2 4 6
```

3. Documenta las incidencias detectadas en el ejercicio anterior. Indica el tipo de prueba realizado.

El tipo de prueba que he realizado ha sido del tipo estructural, ya que como en el anterior punto arreglé el código del programa y me saltó con un error de ejecución comprobé el funcionamiento interno del programa en la parte que daba el fallo, que era en la salida del **for**. Luego el método para averiguar donde ocurría el fallo y localizarlo fue a través del debugger de NetBeans y cuando lo encontré realicé el cambio en el código para arreglar el problema y luego realicé una prueba de regresión para ver si con el cambio realizado si funciona el programa y al realizarlo compruebo que el programa funciona correctamente viendo la salida esperada.

4. ¿En qué consiste un Plan de Pruebas?.

Sirve como guía para la realización de las pruebas, y permite verificar que el sistema de información cumple las necesidades establecidas por el usuario, con las debidas garantías de calidad.

El Plan de Pruebas es un producto formal que define los objetivos de la prueba de un sistema, establece y coordina una estrategia de trabajo, y provee del marco adecuado para elaborar una planificación paso a paso de las actividades de prueba. El plan se inicia en el proceso Análisis del Sistema de Información (ASI), definiendo el marco general, y estableciendo los requisitos de prueba de aceptación, relacionados directamente con la especificación de requisitos.
Dicho plan se va completando y detallando a medida que se avanza en los restantes procesos del ciclo de vida del software, Diseño del Sistema de Información (DSI), Construcción del Sistema de Información (CSI) e Implantación y Aceptación del Sistema (IAS).
Se plantean los siguientes niveles de prueba:

- Pruebas unitarias.
- Pruebas de integración.
- Pruebas del sistema.
- Pruebas de implantación.
- Pruebas de aceptación.

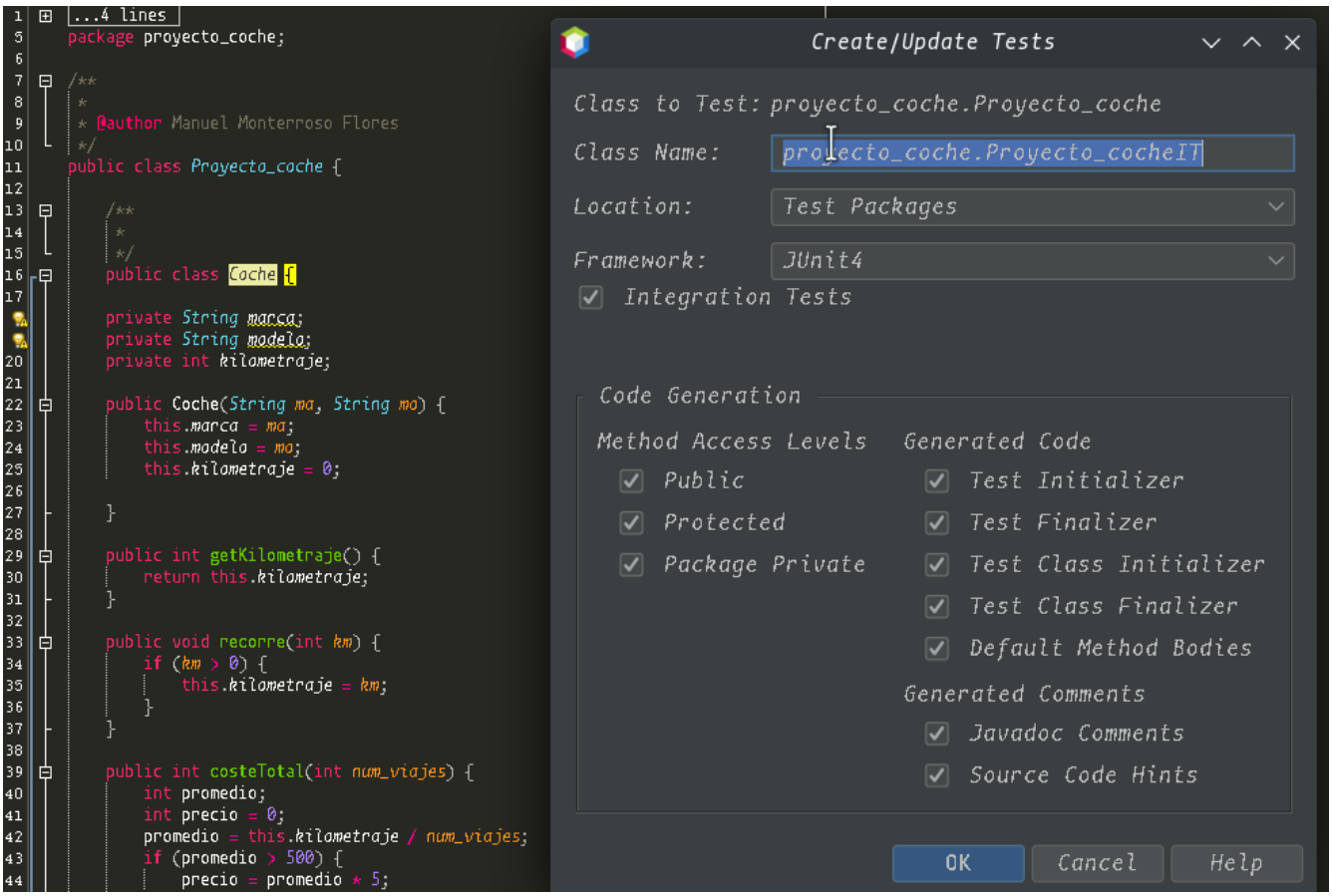
5. A partir del siguiente código, diseña casos de prueba del método “recorre” mediante la técnica de clases de equivalencia.

– Caso de pruebas para km > 0:
  Km por debajo = 0
  Km por encima = 100

– Caso de pruebas para km > 0:
  Km por debajo = –5
  Km por encima = 5

6. Instala Junit en NetBenas (no hace falta ya está instalado de base), e implementa pruebas a los métodos “recorre” y “costeTotal” del ejercicio anterior. Genera las clases Test y Test Suite correspondientes.

– Creando Test con Junit 4.



Aquí con el código de pruebas para recorrer.

```java
@Test
public void testRecorre() {

    System.out.println("prueba del método recorre");
    Coche instance = new Coche("Seat", "Tarraco");

    int valorEsperado = 10;
    int kmActuales = 10;

    instance.recorre(kmActuales);
    assertTrue("Se han introducido " + kmActuales + " kms y se esperaban " + valorEsperado + " kms.", valorEsperado == instance.getKilometraje());
}

@Test
public void testRecorre2() {

    System.out.println("prueba del método recorre con error al introducir un número menor del permitido");
    Coche instance = new Coche("Seat", "Tarraco");

    int valorEsperado = -1;
    int kmActuales = -1;

    instance.recorre(kmActuales);
    assertTrue("Se han introducido " + kmActuales + " kms y se esperaban " + valorEsperado + " kms.", valorEsperado == instance.getKilometraje());
}
```

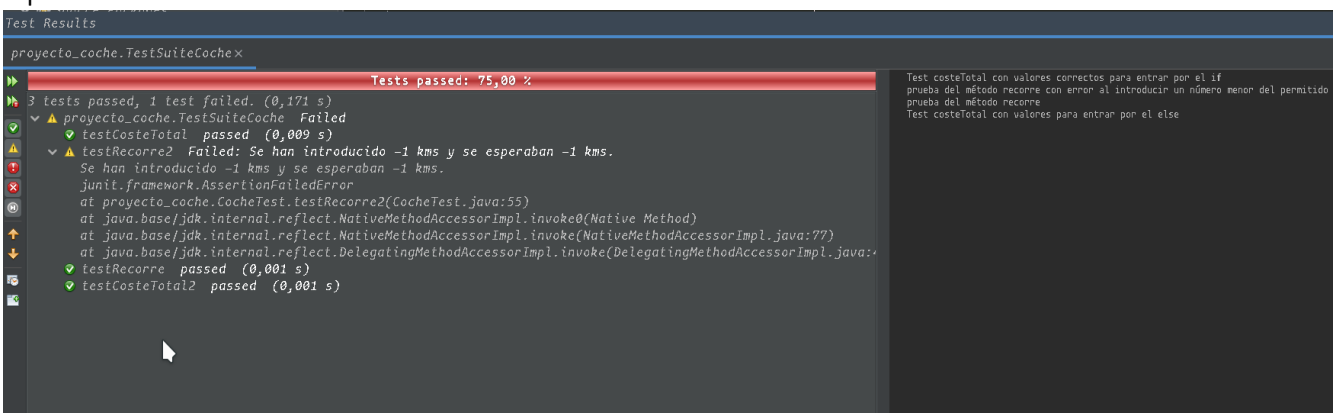
Aquí captura con los test para costeTotal.

```java
@Test
public void testCosteTotal() {
    //test en el que se comprueba con valores correcto que funcione el código y de correcto
    System.out.println("Test costeTotal con valores correctos para entrar por el if");
    Coche instance = new Coche("Seat", "Tarraco");
    int num_viajes = 2;
    int kmActuales = 600;
    instance.recorre(kmActuales);
    int expResult = 600;

    int result = instance.costeTotal(num_viajes);
    assertEquals("El coste total de este núnero de viaje/s " +num_viajes +"ha sido de " + result,expResult, result);
    //fail("The test case is a prototype.");
}

@Test
public void testCosteTotal2() {
    //test en el que se comprueba con valores incorrector que funcione el código y de error
    System.out.println("Test costeTotal con valores para entrar por el else");
    Coche instance = new Coche("Seat", "Tarraco");
    int num_viajes = 2;
    int kmActuales = 1200;
    instance.recorre(kmActuales);
    int expResult = 3000;

    int result = instance.costeTotal(num_viajes);
    assertEquals("El coste total de este núnero de viaje/s " +num_viajes +" ha sido de " + result +".",expResult, result);
    //fail("Viajes o Km introduzos no corresponden.");
}
```

Aquí el código del SuiteTest.

```java
/**
 *
 * @author Manuel Monterroso Flores
 */
@RunWith(Suite.class)
@Suite.SuiteClasses({proyecto_coche.CocheTest.class})
public class TestSuiteCoche {

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }
```

Aquí con el suiteTest dando los resultados de cada test.



7. ¿Qué normas de calidad crees que se deben de aplicar a los procedimientos de desarrollo de software? Razona la respuesta

Los estándares que se han venido utilizando en la fase de prueba de software son:

Estándares BSI

- BS 7925-1, Pruebas de software. Parte 1. Vocabulario.
- BS 7925-2, Pruebas de software. Parte 2. Pruebas de los componentes software.
- Estándares IEEE de pruebas de software.:
- IEEE estándar 829, Documentación de la prueba de software.
- IEEE estándar 1008, Pruebas de unidad
- Otros estándares ISO/IEC 12207, 15289

•Otros estándares sectoriales

Sin embargo, estos estándares no cubren determinadas facetas de la fase de pruebas, como son la organización el proceso y gestión de las pruebas, presentan pocas pruebas funcionales y no funcionales etc. Ante esta problemática, la industria ha desarrollado la norma ISO/IEC 29119.

La norma ISO/IEC 29119 de prueba de software, pretende unificar en una única norma, todos los estándares, de forma que proporcione vocabulario, procesos, documentación y técnicas para cubrir todo el ciclo de vida del software. Desde estrategias de prueba para la organización y políticas de prueba, prueba de proyecto al análisis de casos de prueba, diseño, ejecución e informe. Con este estándar, se podrá realizar cualquier prueba para cualquier proyecto de desarrollo o mantenimiento de software.

Por tanto creo que se debería de usar la norma ISO/IEC 29119 ya que en una misma norma reúne todos los procedimientos para realizar todas las pruebas a nuestro software y por estar más actualizado que los anteriores.

8. ¿Qué medidas de calidad han de realizarse sobre cualquier software que se desarrolle? Razona la respuesta.

Las medidas de calidad que tomaría serían las siguientes:

- Número de errores durante un periodo determinado, esta medida consiste que dada la complejidad de las aplicaciones informáticas, que se desarrollan en la actualidad, es prácticamente imposible, probar todas la combinaciones que se pueden dar dentro de un programa o entre un programa y las aplicaciones que pueden interactuar con él. Por este motivo, en el diseño de los casos de prueba, siempre es necesario asegurar que con ellos se obtiene un nivel aceptable de probabilidad de que se detectarán los errores existentes.
  Las pruebas deben buscar un compromiso entre la cantidad de recursos que se consumirán en el proceso de prueba, y la probabilidad obtenida de que se detecten los errores existentes en un tiempo determinado.

- Fallo en la codificación o diseño de un sistema, que causa que el programa no funcione correctamente, o falle.

- **Tamaño de un producto informático (líneas de código),** ya que si un programa mientras sea más extenso mas difícil es poder realizar modificaciones y de encontrar los posibles problemas por tanto lo aconseja es reducir todo lo posible el código o dividirlo en clases.

- **Estimación de costes y esfuerzos,** ya que si no se controla estos puntos puede hacer que la inversión de la construcción del programa sea muy elevado tanto a nivel económico como para los que estén realizando el programa.