Programación en Lenguaje Java

Tema 11. Pruebas

Michael González Harbour

Mario Aldea Rivas

Departamento de Matemáticas,
Estadística y Computación

Programación en Java

1. Introducción a los lenguajes de programación
2. Datos y expresiones
3. Estructuras algorítmicas
4. Datos Compuestos
5. Entrada/salida
6. Clases, referencias y objetos
7. Modularidad y abstracción
8. Herencia y polimorfismo
9. Tratamiento de errores
10. Entrada/salida con ficheros
11. ***Pruebas***

- Pruebas del software. Caja negra: particiones de equivalencia. Herramienta JUnit.

11.1 Pruebas del software

Probar un módulo software (método, clase, paquete, etc.) consiste en:

- ***Ejecutar el módulo*** bajo diferentes situaciones
- Para ***comprobar que presenta el comportamiento esperado***
  - Se comporta como dice su especificación (sus comentarios de documentación)

Fundamental disponer de un ***entorno automatizado de pruebas***

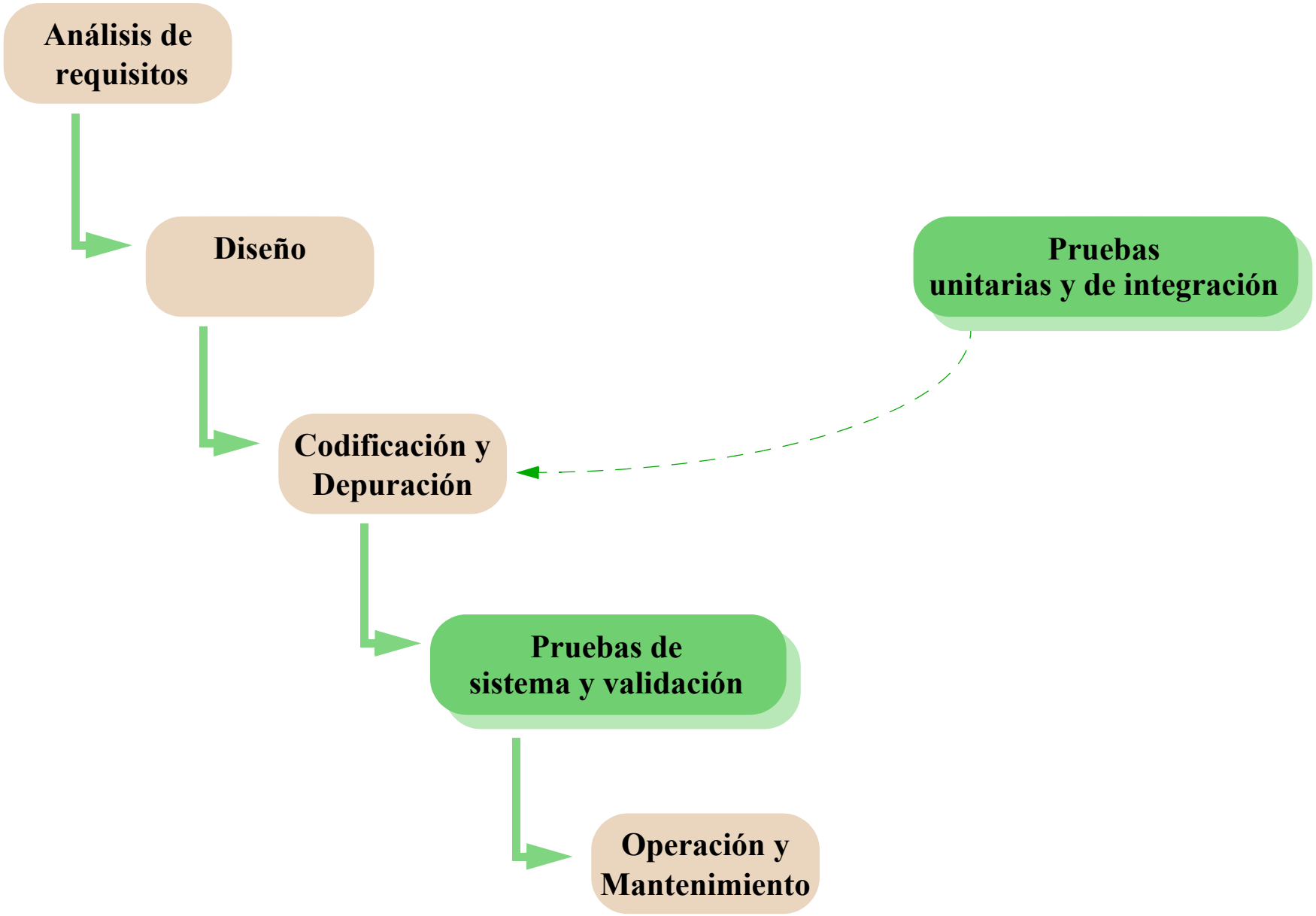
- Ejecución automática de los casos de prueba
- Generación de informes

Lo tratado en este tema no pretende ser más que una breve introducción al capítulo de "Pruebas de Sistemas Software" de la asignatura "Ingeniería del Software II" (3er curso)

Las pruebas se realizan en cuatro ***etapas***:

Visto en este tema

1. ***Pruebas unitarias*** (prueba independiente de cada clase)
   - se prueban sus métodos para distintos ***casos de prueba***
   - se intenta cubrir todo el comportamiento de la clase
2. Prueba de integración o de subsistemas
   - se prueban agrupaciones de clases relacionadas
3. Prueba de sistema
   - se prueba el sistema como un todo
4. Prueba de validación
   - prueba del sistema en el entorno real de trabajo
   - con intervención del usuario final


Análisis de requisitos
Diseño
Codificación y Depuración
Pruebas de sistema y validación
Operación y Mantenimiento
Pruebas unitarias y de integración

Pruebas unitarias (de clases)

Caso de prueba: ejecución de un método bajo unas condiciones particulares (valores de los parámetros + estado de la clase)

- El resultado permite determinar si, para este caso particular, la clase se ha comportado acorde a su especificación

En general el número de posibles casos de prueba es muy alto (muchas veces infinito)

Objetivo de la prueba de clases:

- ***Encontrar el mayor número de defectos posible***
- Utilizando un número "pequeño" de casos de prueba

Nunca olvidar que:

- Las pruebas permiten probar la existencia de defectos, no la ausencia de éstos

Estrategias para la elección de casos de prueba

Dos enfoques:

Visto en este tema

- Pruebas funcionales o de "***caja negra***"
  - los casos de prueba se basan en la especificación del módulo
  - no se requiere conocimiento de su estructura interna
  - no es necesario disponer del código fuente
- Pruebas estructurales o de "***caja blanca***" o "caja de cristal"
  - los casos de prueba se seleccionan en función del conocimiento que se tiene de la estructura del método
  - es necesario disponer del código fuente

11.2 Caja negra: particiones de equivalencia

En general es imposible probar un método para todas las combinaciones posibles de entradas y de estados de la clase

En lugar de eso las entradas/estados se dividen en ***particiones de equivalencia***: conjunto de entradas/estados que, según la especificación, se espera que tengan un comportamiento equivalente

Los casos de prueba se eligen en función de las particiones:

- Se elige un caso de prueba para cada combinación válida de particiones
- Se elige un valor central por partición: caso típico
- Se eligen valores correspondientes a las fronteras con otras particiones: casos atípicos

Ej.: elección de casos de prueba

Prueba de una clase `BuscaEnArray` con un único método (`busca`):

```
public class BuscaEnArray {

   /**
    * Busca la primera ocurrencia de un elemento en un array
    * @param ele elemento buscado
    * @param a array en el que se busca
    * @return la posición en la que se encuentra el elemento o -1
    * en el caso de que el elemento no esté en el array
    * @throws SecuenciaVacia si el array no tiene ningún elemento
    */
    public static int busca(int ele, int[] a) throws SecuenciaVacia {
      ...
    }
}
```

Particiones de equivalencia

- "el elemento está en el array" / "el elemento no está en el array"
- "array vacío" / "array con elementos"

Criterios de prueba generales para secuencias:

- utilizar secuencias de distintos tamaños (en especial de tamaños 0 y 1)
- acceder a los elementos primero, central y último

Casos de prueba basados en las particiones y los criterios:

Particiones/criterios	Casos de prueba			Resultado esperado
	ID	a	ele	
Array vacío	1	vacío	20	Excepción
Frontera entre vacío y no vacío: encontrado y no	2	[17]	17	0
	3	[17]	18	-1
Secuencia de más de un elemento: encontrado (primero, central y último) y no encontrado	4	[17, 29, 21, 23]	17	0
	5	[17, 18, 21, 23, 29, 41, 38]	23	3
	6	[41, 18, 9, 31, 30, 16, 45]	45	6
	7	[21, 23, 29, 33, 38]	25	-1

11.3 Herramienta JUnit

JUnit es una herramienta de código abierto que permite ejecutar conjuntos de pruebas de forma rápida y sistemática

- integrada en el entorno Eclipse
- para ***prueba de clases***
- http://www.junit.org

Pensada para realizar pruebas repetidamente cada vez que se realiza un pequeño cambio en el código

- *"Code a little, test a little, code a little, test a little".*

Permite crear una ***Clase probadora*** para cada una de nuestras clases

- Formada por un conjunto de ***métodos de prueba***
- Cada ***método de prueba*** implementa uno o más ***casos de prueba***

Métodos de prueba

Son ejecutados automáticamente por la herramienta JUnit

- no debemos asumir ningún orden de ejecución

Debemos escribir en ellos nuestros casos de prueba

Para cada método se diferencian ***tres comportamientos***:

- ***correcto***: si finaliza sin lanzar ninguna excepción
- ***fallo***: si lanza la excepción `AssertionFailedError`
- ***error***: si lanza cualquier otra excepción

Para detectar los ***fallos*** usaremos el método:

```
void assertTrue(String msj, Boolean cond)
```

- lanza `AssertionFailedError` con `msj` como mensaje asociado cuando `cond` es false

Ejemplo: clase probadora de `BuscaEnArray`

Implementa los casos de prueba identificados en la tabla de la transparencia página 9

Hemos elegido organizarla en tres métodos de prueba

- `testArrayVacio`: caso de prueba 1
- `testArrayUnEle`: casos de prueba 2 y 3
- `testArrayMasDeUnEle`: casos de prueba 4, 5, 6 y 7

```
/**
 * Clase probadora de la clase BuscaEnArray
 */
public class BuscaEnArrayTest {


   @Test
   public void testArrayVacio() {
      // Array vacío (1)
      int[] a = {};
      try {
         int pos = BuscaEnArray.busca(20, a);
         assertTrue("No excepción array vacío", false);
      } catch (BuscaEnArray.SecuenciaVacia e) {
         // El comportamiento correcto es que se lance la excepción
         // Simplemente la cojo para que no salga fuera del método y
         // JUnit lo interprete como un error
      }
   }
```

```
@Test
public void testArrayUnEle() {
   int[] a = {17};
   int pos;

   // elemento encontrado (2)
   pos = BuscaEnArray.busca(17, a);
   assertTrue("Error posición:" + pos, pos == 0);

   // elemento no encontrado (3)
   pos = BuscaEnArray.busca(18, a);
   assertTrue("Error posición:" + pos, pos == -1);
}
```

```
    @Test
    public void testArrayMasDeUnEle() {
        int pos;

        // encontrado primero (4)
        int[] a1 = {17, 29, 21, 23};
        pos = BuscaEnArray.busca(17, a1);
        assertTrue("Error posición:" + pos, pos == 0);

        // encontrado central (5)
        int[] a2 = {17, 18, 21, 23, 29, 41, 38};
        pos = BuscaEnArray.busca(23, a2);
        assertTrue("Error posición:" + pos, pos == 3);

        // encontrado último (6)
        int[] a3 = {41, 18, 9, 31, 30, 16, 45};
        pos = BuscaEnArray.busca(45, a3);
        assertTrue("Error posición:" + pos, pos == 6);

        // no encontrado (7)
        int[] a4 = {21, 23, 29, 33, 38};
        pos = BuscaEnArray.busca(25, a4);
        assertTrue("Error posición:" + pos, pos == -1);
    }

} // fin clase BuscaEnArrayTest
```

Uso de JUnit desde el entorno Eclipse

Configuración del proyecto para usar JUnit:

1. Botón derecho del ratón sobre el proyecto y elegir: `Build Path => Configure Build Path`
2. En la ficha `Libraries` elegir `Add Library`, seleccionar `JUnit` y pulsar `Next`, elegir la versión `JUnit 4`

Creación de una clase probadora en JUnit:

1. Pulsar con el botón derecho sobre la clase a probar y elegir `New => Other => Java => JUnit => JUnit Test Case`
2. Seleccionar los métodos a probar y pulsar `Finish`
3. Se crea la clase probadora con un conjunto de métodos de prueba que tienen `@Test` antes del método
4. Podemos añadir más métodos de prueba (también deberán tener `@Test`)

Resultados y Depuración de métodos de prueba

Obtención de los resultados:

- Ejecutando la clase probadora ("*run*")

Resumen de resultados con:

- "*Failures*" ✗ : fallo detectado con `assertTrue`
- "*Errors*" ✗ : excepción inesperada

Es posible depurar un método individualmente:

1. situar al menos un punto de ruptura en el método a depurar
2. pulsar con el botón derecho sobre el método de prueba y elegir debug