

## **Tema 4. Pruebas**

Metodología y Desarrollo de Programas

- Verificación y Validación
- Tipos de Pruebas
- Pruebas de caja blanca
- Pruebas de Caja Negra
- El framework JUnit
- JUnit en eclipse
- El framework Junit II
- Otros frameworks de Test
- Test-driven development

- <http://www.zdnet.com/software-testing-and-qa-budgets-keep-rising-with-more-emphasis-on-the-new-stuff-7000034517/>
- Cualquier proceso de construcción moderno y profesional existe una fase de test o pruebas, y el desarrollo de software no es menos.



¿Cómo probáis que vuestro código funciona?

# Verificación y Validación (VV)

- VV es un conjunto de **procedimientos, actividades, técnicas y herramientas** que se utilizan, paralelamente al desarrollo de software, para asegurar que un producto software resuelve el problema inicialmente planteado
- Objetivos:
  - ❖ **Detectar y corregir** los defectos tan pronto como sea posible en el ciclo de vida del software
  - ❖ Disminuir los **riesgos**, las desviaciones sobre los presupuestos y sobre el calendario o programa de tiempos del proyecto
  - ❖ Mejorar la **calidad y fiabilidad** del software
  - ❖ Mejorar la **visibilidad** de la gestión del proceso de desarrollo
  - ❖ Valorar rápidamente los **cambios propuestos** y su consecuencias
- **Las pruebas** son una familia de técnicas de VV

## ➤ Actividades de VV

### ❖ Verificación:

- ✓ ¿Estamos construyendo correctamente el producto?
- ✓ El software debe ser conforme a su especificación
- ✓ Objetivo: Demostrar la consistencia, completitud y corrección de los artefactos software entre las fases del ciclo de desarrollo de un proyecto
- ✓ Técnica más utilizada: **Revisiones SW**

### ❖ Validación:

- ✓ ¿Estamos construyendo el producto sin fallos?
- ✓ El software debe hacer lo que el usuario realmente quiere
- ✓ Objetivo: Determinar la corrección del producto final respecto a las necesidades del usuario
- ✓ Técnica más utilizada: **Pruebas SW**

# Pruebas de software

---

- Las **pruebas de software**, en inglés testing, son **los procesos** que permiten **verificar y revelar** la calidad de un producto software
- Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un programa
- Las pruebas de software se integran dentro de las diferentes fases del **ciclo del software** dentro de la Ingeniería del Software
- Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema
- Hoy en día es crucial verificar y evaluar la calidad de lo construido para minimizar el costo de modificación

## ➤ Tipos de pruebas:

- ❖ Pruebas unitarias
- ❖ Pruebas funcionales
- ❖ Pruebas de Integración
- ❖ Pruebas de validación
- ❖ Pruebas de sistema
- ❖ Caja blanca (sistemas)
- ❖ Caja negra (sistemas)
- ❖ Pruebas de aceptación
- ❖ Pruebas de regresión
- ❖ Pruebas de carga
- ❖ Pruebas de prestaciones
- ❖ Pruebas de recorrido
- ❖ Pruebas de mutación
- ❖ Pruebas concurrentes

## ➤ Pruebas de Caja Blanca:

- ❖ Se centra en el estudio minucioso de la operatividad de una parte del sistema considerando los detalles procedurales (la lógica del sistema)
  - ✓ Pruebas que hagan que se recorran todos los posibles caminos de ejecución, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles , etc.
- ❖ Resuelve: ¿cómo lo hace?

## ➤ Pruebas de Caja Negra:

- ❖ Aquella prueba realizada desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.
- ❖ Resuelve: ¿qué es lo que hace el software?





# Pruebas de software. Pruebas de caja blanca

- Usa la estructura de control del diseño procedimental para obtener los casos de prueba
- Estos casos deben garantizar:
  - ❖ Cobertura de Sentencias
    - ✓ Cada sentencia se ejecuta al menos una vez
  - ❖ Cobertura de Decisiones
    - ✓ Cada sentencia se ejecuta al menos una vez; cada decisión toma todos los resultados posibles al menos una vez
  - ❖ Cobertura de Condiciones
    - ✓ Cada sentencia se ejecuta al menos una vez; cada condición en la decisión toma todos los posibles resultados al menos una vez
  - ❖ Cobertura de Decisión/Condición
    - ✓ Cada sentencia se ejecuta al menos una vez; cada decisión toma todos los resultados posibles al menos una vez; cada condición en la decisión toma todos los posibles resultados al menos una vez
  - ❖ Cobertura de Condiciones Múltiple
    - ✓ Cada sentencia se ejecuta al menos una vez; todas las posibles combinaciones de los resultados de la condición en cada decisión ocurre al menos una vez

- Pueden utilizarse entre otras técnicas:
  - ❖ Prueba del camino básico
    - ✓ Se utiliza una representación del flujo de control en forma de grafo (Grafo del flujo)
  - ❖ Tablas

- Ejemplo (obtenido de [http://www.rogeliodavila.com/tcs/TCS%20Notes%20JAVega/Parte\\_16\\_TestWhite.ppt](http://www.rogeliodavila.com/tcs/TCS%20Notes%20JAVega/Parte_16_TestWhite.ppt))

```
procedure liability (age, sex, married, premium);  
begin  
    premium := 500;  
    if ((age < 25) and (sex = male) and (not married)) then  
        premium := premium + 1500;  
    else (if (married or (sex = female)) then  
        premium := premium - 200;  
        if ((age > 45) and (age < 65)) then  
            premium := premium - 100;  
        end;  
    end;  
end;
```

## Cobertura de Decisiones

```
procedure liability (age, sex, married, premium);  
begin  
  premium := 500;  
  if ((age < 25) and (sex = male) and (not married)) then  
    premium := premium + 1500;  
  else (if (married or (sex = female)) then  
    premium := premium - 200;  
    if ((age > 45) and (age < 65) then  
      premium := premium - 100;  
    end;  
  end;
```

Cobertura de Decisiones	age	sex	married	caso de pruebas
IF-1	<25	Male	FALSO	(1) 23 M F
IF-1	<25	Female	FALSO	(2) 23 F F
IF-2	*	Female	*	(2)
IF-2	>=25	Male	FALSO	(3) 50 M F
IF-3	<=45	Female	*	(2)
IF-3	>45, <65	*	*	(3)

## Cobertura de Condiciones

```
procedure liability (age, sex, married, premium);  
begin  
  premium := 500;  
  if ((age < 25) and (sex = male) and (not married)) then  
    premium := premium + 1500;  
  else (if (married or (sex = female)) then  
        premium := premium - 200;  
        if ((age > 45) and (age < 65) then  
          premium := premium - 100;)  
end;
```

Cobertura de Condiciones	age	sex	married	caso de pruebas
IF-1	<25	Female	FALSO	(1) 23 F F
IF-1	>=25	Male	###	(2) 30 M T
IF-2	*	Male	###	(2)
IF-2	*	Female	FALSO	(1)
IF-3	<=45	*	*	(1)
IF-3	>45	*	*	(3) 70 F F
IF-3	<65	*	*	(2)
IF-3	>=65	*	*	(3)

# Cobertura de Condiciones

Cobertura de Decisiones/ Condiciones	age	sex	married	caso de pruebas
IF-1	<25	Male	FALSE	(1) 23 M F
IF-1	<25	Female	FALSE	(2) 23 F F
IF-1	<25	Female	FALSE	(2)
IF-1	>=25	Male	TRUE	(3) 70 M T
IF-2	*	Female	*	(2)
IF-2	>=25	Male	FALSE	(4) 50 M F
IF-2	*	Male	TRUE	(3)
IF-2	*	Female	FALSE	(2)
IF-3	<=45	*	*	(2)
IF-3	>45, <65	*	*	(4)
IF-3	<=45	*	*	(2)
IF-3	>45	*	*	(4)
IF-3	<65	*	*	(4)
IF-3	>=65	*	*	(3)

## Cobertura de Condiciones

Cobertura de Condiciones Múltiples	age	sex	married	caso de pruebas
IF-1	<25	Male	TRUE	(1) 23 M T
IF-1	<25	Male	FALSE	(2) 23 M F
IF-1	<25	Female	TRUE	(3) 23 F T
IF-1	<25	Female	FALSE	(4) 23 F F
IF-1	>=25	Male	TRUE	(5) 30 M T
IF-1	>=25	Male	FALSE	(6) 7 M F
IF-1	>=25	Female	TRUE	(7) 50 F T
IF-1	>=25	Female	FALSE	(8) 30 F F
IF-2	*	Male	TRUE	(5)
IF-2	*	Male	FALSE	(6)
IF-2	*	Female	TRUE	(7)
IF-2	*	Female	FALSE	(8)
IF-3	<=45, >=65	*	*	imposible
IF-3	<=45, <65	*	*	(8)
IF-3	>45, >=65	*	*	(6)
IF-3	>45, <65	*	*	(7)

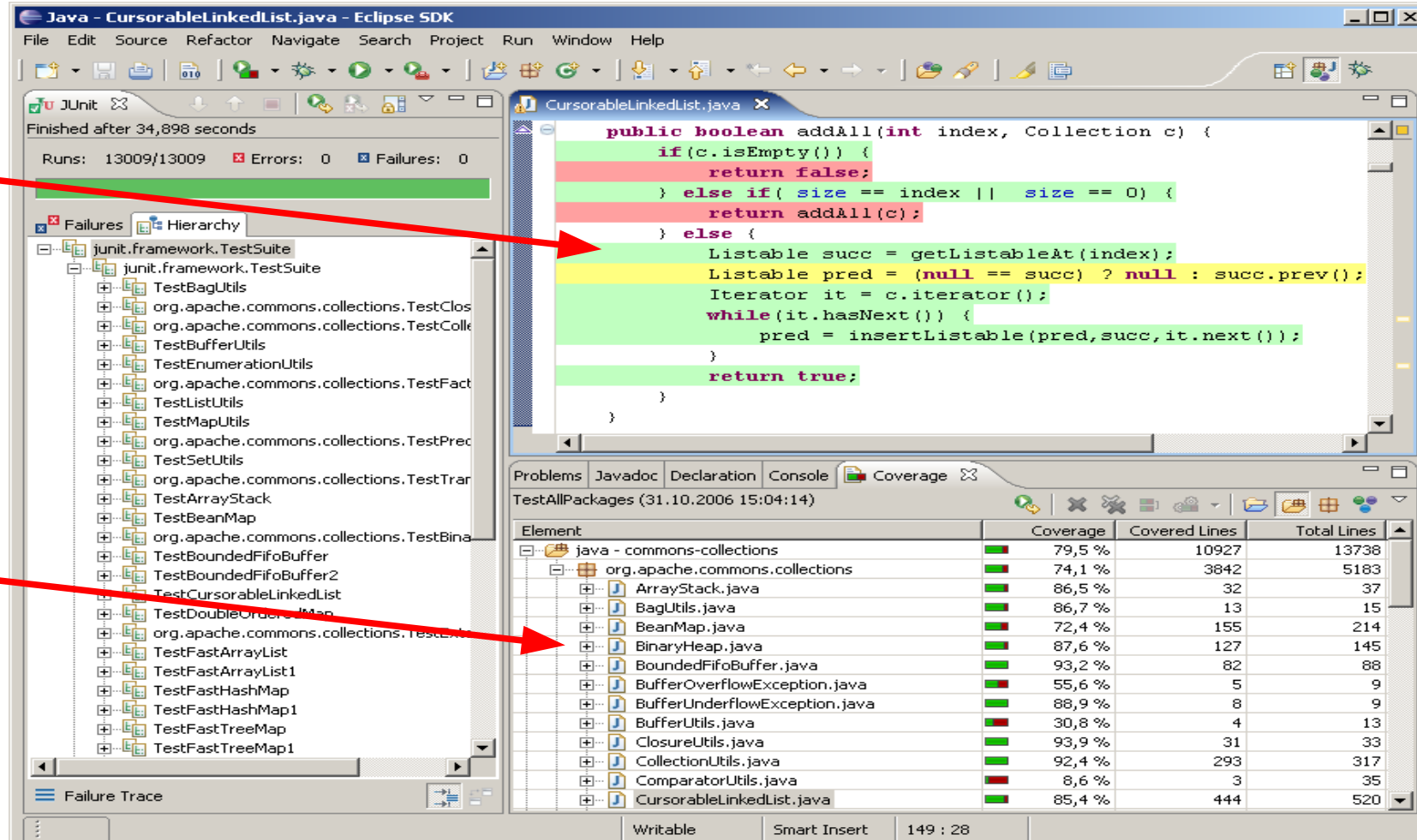
# Pruebas de software. Pruebas de caja blanca

## ➤ Eclipse y Prueba de caja blanca: eclemma

- ❖ Para su instalación añadir el repositorio: <http://update.eclemma.org/> e instalar

Las marcas verdes indican sentencias ejecutadas, las rojas las no ejecutadas y las amarillas las parcialmente ejecutadas

En la ventana inferior "coverage" se incluye el % de cobertura de sentencias.



The screenshot displays the Eclipse IDE interface. The main editor shows the `CursorableLinkedList.java` file with color-coded lines indicating execution status. The bottom panel shows the `Coverage` view for `TestAllPackages (31.10.2006 15:04:14)`.

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520



# Pruebas de software. Pruebas de caja negra

---

- Los casos de prueba de la caja negra pretenden demostrar que:
  - ❖ Las funciones del software son operativas
  - ❖ La entrada se acepta de forma correcta
  - ❖ Se produce una salida correcta
  - ❖ La integridad de la información externa se mantiene
  
- Las pruebas de caja negra suelen realizarse:
  - ❖ A nivel de métodos
  - ❖ A nivel de clases
  - ❖ A nivel de paquetes
  - ❖ A nivel de estados
  
- Pruebas de caja negra con **Junit**. [www.junit.org](http://www.junit.org)

```
public class Carta {  
    private int valor;  
    private int palo;  
    private String rutaImagen;  
    private String rutaReverso;  
    private boolean levantada;  
    .....  
    ....  
}
```

← *¿Cómo probar esta clase a nivel unitario??*

- *Programador prueba aleatoriamente casos imaginarios?*
- *Vuelve a realizar un caso que ha fallado?*
- *Cuándo hace un cambio, prueba los fallos anteriores?*
- *Comprueba si clase esta correcta con respecto a las otras?*
- *.....*

- 5 cualidades que debe tener un buen test (FIRST):
  - ❖ **Fast:** Los test deben ejecutarse rápido
  - ❖ **Independant:** Un test nunca debe depender de otros para pasar. Los puedes ejecutar en cualquier orden y deben funcionar igual
  - ❖ **Repeteable:** El test se debe ejecutar en cualquier entorno
  - ❖ **Self-validating:** Un test debe devolver: "correcto" o "fallo" y no debe requerir ningún tipo de intervención manual posterior que determine si el test paso o no
  - ❖ **Timely:** Con esto se refiere a que los test deben ser escritos en el momento adecuado, es decir, antes de que el código vaya a producción

## ➤ Ventajas:

- ❖ Aislar cada parte del programa y mostrar que las partes individuales son correctas
- ❖ Fomentan el cambio: Programador cambie el código para mejorar su estructura (refactorización), pues se asegura que los nuevos cambios no han introducido errores
- ❖ Simplifica la integración: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad
- ❖ Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo
- ❖ Separación de la interface y la implementación
- ❖ Los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos

# El framework JUnit

---

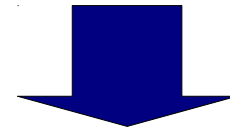
- JUnit es un “framework” para automatizar las pruebas de programas Java
- Open Source, disponible en <http://www.junit.org>
- Adecuado para el Desarrollo dirigido por las pruebas (*Test-driven development*)
- *Consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba y ejecutarlos automáticamente*
- Los casos de prueba son realmente programas Java. Quedan archivados y pueden ser reejecutados tantas veces como sea necesario

## ➤ Cómo pruebo el método equals de Carta??

- ❖ Poniendo un caso de uso correcto e incorrecto que verifiquen su funcionamiento → **Importante:** Automático y repetible
- ❖ Mediante el uso de instrucciones **if**.

```
//Prueba1
Carta c1= new Carta(10,2,"","",true);
Carta c2= new Carta(10,2,"","",true);
boolean expected=true;
boolean result=c1.equals(c2));
if (expected==result)
    System.out.println("OK");
else
    System.out.println("Fallo");
```

- Muy artesano
- Arduo
- Salida



jUnit

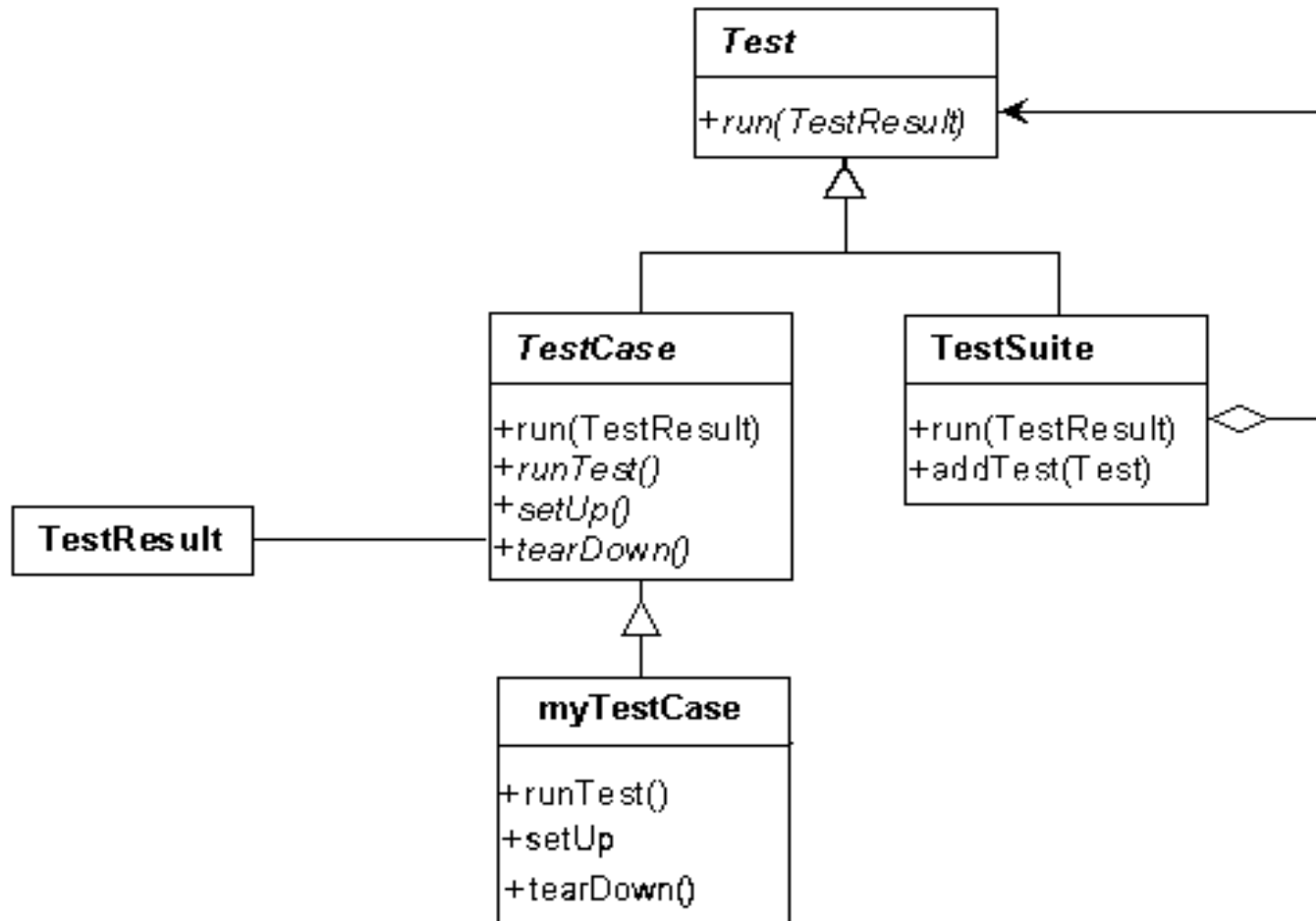
- El ejemplo anterior (*result* frente a *expected*) es una idea fundamental de JUnit
  - ❖ Siempre se va a establecer el resultado y se va a comparar con el resultado obtenido
- JUnit:
  - ❖ Permite separar los casos de prueba frente al código fuente
  - ❖ Permite ejecutarlos automáticamente
  - ❖ Nos va a permitir realizar conjuntos de test (*suites*)

- JUnit tiene un conjunto de métodos para comprobar lo esperado con lo obtenido

```
public class Tests extends TestCase {  
    public Tests () { }  
    public void testSumas () {  
        assertEquals(4, 2+2);  
    }  
    public void testDivisiones () {  
        assertTrue(1==2/2);  
    }  
}
```



# El framework JUnit



## El framework JUnit

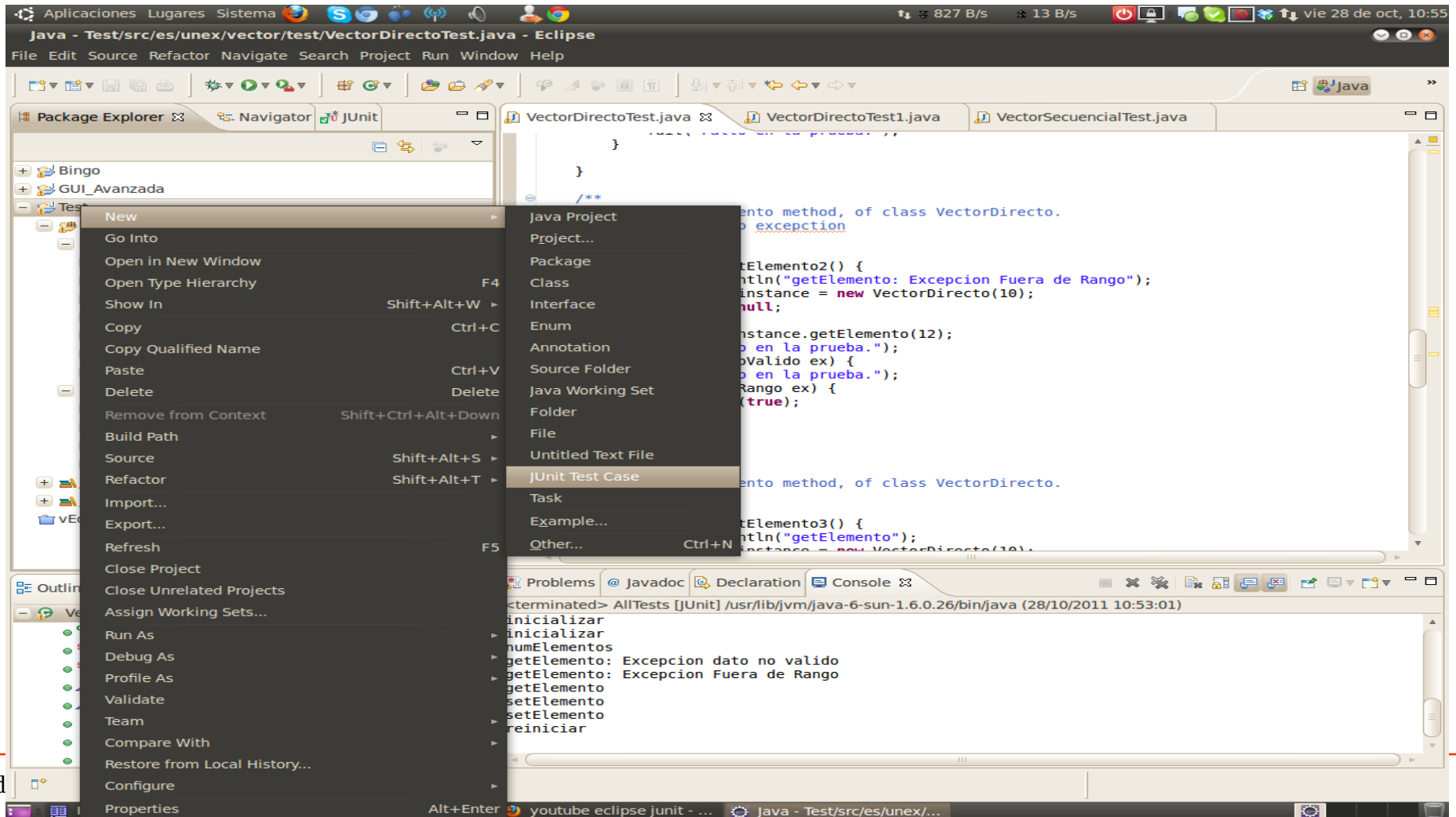
- JUnit incluye una serie de métodos para probar que las cosas son como esperamos.

<code>assertEquals (X esperado, X obtenido)</code>	compara un resultado esperado con el resultado obtenido, determinando que la prueba pasa si son iguales, y que la prueba falla si son diferentes.
<code>assertFalse (boolean resultado)</code>	verifica que el resultado es FALSE
<code>assertTrue (boolean resultado)</code>	verifica que el resultado es TRUE
<code>assertNull (Object resultado)</code>	verifica que el resultado es "null"
<code>assertNotNull (Object resultado)</code>	verifica que el resultado no es "null" fail sirve para detectar que estamos en un sitio del programa donde NO deberíamos estar

# jUnit en Eclipse

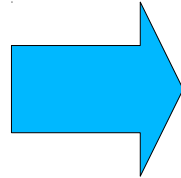
➤ Eclipse facilita el trabajo con jUnit → genera el esqueleto y sólo es necesario rellenarlo correctamente

❖ Proyecto → Botón Derecho → new → jUnit Test Case



- Por cada método de la clase ha generado un método TestXXX con su esqueleto. Cuidado:
  - ❖ Por defecto devuelve fail: hay que quitarlo
  - ❖ Hay que adaptar el código.

```
@Test  
public void testSetElemento() {  
    fail("Not yet implemented");  
}
```

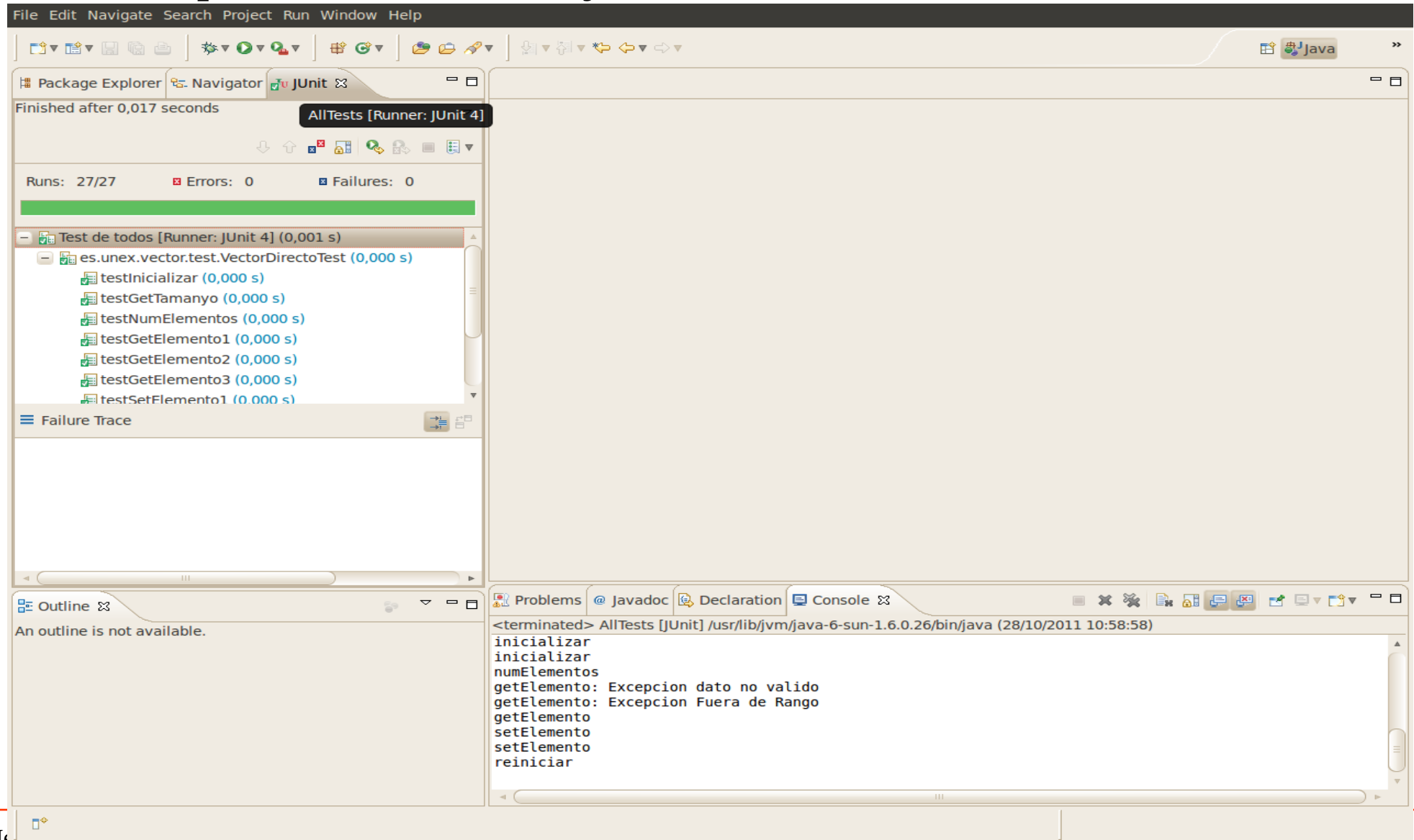


```
@Test  
public void testSetElemento() {  
    System.out.println("setElemento");  
    VectorDirecto instance = new VectorDirecto(10);  
    try {  
        instance.setElemento(12, new Persona());  
        fail("Fallo en la prueba.");  
    } catch (FueraRango ex) {  
        assertTrue(true);  
    }  
}
```

Generado

Correcto

➤ Para probar los test → Ejecutar el fichero de Test



## ➤ Como verificar que se producen excepciones

```
@Test
public void MethodTest()
{
    try
    {
        Vector v= new Vector (10);
        For (int i=0;i<20;i++)
            v.add(new Persona());
        Assert.Fail("An exception should have been thrown");
    } catch (VectorLleno ae)  {
        System.out.println(ae.getMessage());
    } catch (Exception e)  {
        System.out.println(e.getMessage());
    }
}
```

## ➤ Otros métodos de Test

### ❖ SetUp():

- ✓ JUnit lo llamará antes de lanzar cada uno de los casos de prueba, testXXX.
- ✓ Puede ser útil cuando todos los casos de prueba requieren la misma inicialización de variables privadas y no basta con hacerlo en el constructor.

### ❖ TearDown():

- ✓ JUnit lo llamará después de lanzar cada uno de los casos de prueba, testXXX.
- ✓ Puede ser útil para cerrar elementos abiertos durante la prueba que pudieran quedar en un estado lamentable. Por ejemplo: ficheros, conexiones Internet, conexiones a bases de datos

### ❖ Same

- ✓ A veces no basta probar que dos objetos son iguales con assertEquals() sino que nos interesa comprobar que es exactamente el mismo. Por ejemplo, si estamos probando algoritmos de almacenamiento y recuperación.
- ✓ Para saber si dos objetos son el mismo o no:
  - ✓ assertEquals(X esperado, X real);
  - ✓ assertEquals(X esperado, X real);

## ➤ TestSuite (JUnit 3.x)

❖ Agrupar casos de prueba:

```
import junit.framework.TestSuite;
import junit.framework.Test;
import junit.framework.TestCase;
public class VariosTest extend TestCase{
    public VariosTest( String nombre ) {
        super( nombre );
    }
    public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite( CartaTest.class );
        suite.addTest( JugadorTest.class);
        return suite;
    }
}
```



### ➤ TestSuite (JUnit 4.x)

❖ Agrupar casos de prueba:

```
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;
```

```
@RunWith(Suite.class)  
@Suite.SuiteClasses({  
    TestFeatureLogin.class,  
    TestFeatureLogout.class,  
    TestFeatureNavigate.class,  
    TestFeatureUpdate.class  
})
```

```
public class FeatureTestSuite {  
    // la clase se queda vacía,  
    // se usa sólo como propietario de la notación anterior  
}
```

## ➤ Test en Java:

- ❖ DbUnit: Extensión de JUnit para realizar test unitarios que con un conjunto de datos. <http://www.dbunit.org/>
- ❖ XMLUnit: Para pruebas de XML. <http://xmlunit.sourceforge.net/>
- ❖ HTMLUnit: Para pruebas HTML
- ❖ Jmeter: es una herramienta Java dentro del proyecto Jakarta, que permite realizar Pruebas de Rendimiento y Pruebas Funcionales sobre Aplicaciones Web. <http://jakarta.apache.org/jmeter/>
- ❖ Mock: Simula Sustituyen a clases complejas, dispositivos, etc. como por ejemplo: servlets, páginas jsp, bases de datos...
  - ✓ EasyMock: <http://easymock.org/>
  - ✓ Mockito: <http://mockito.org/>

## ➤ TDD

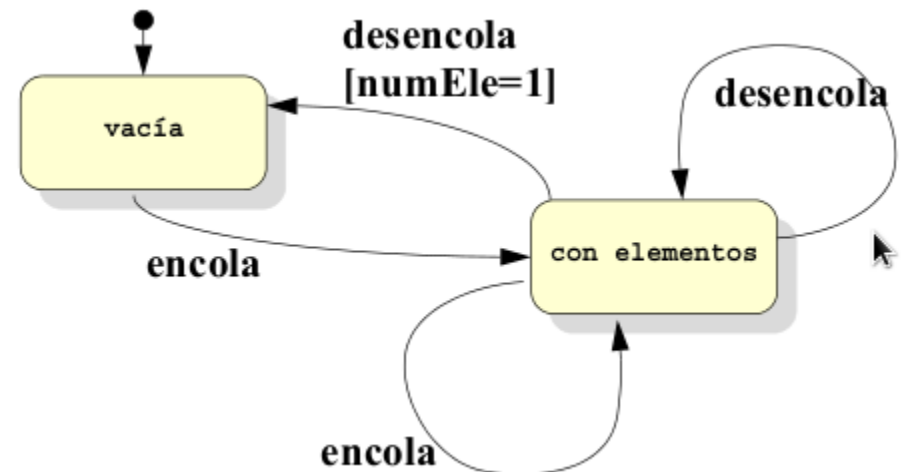
- ❖ Es una metodología en la cual en **Primer Lugar** se escriben las pruebas y se verifica que las pruebas fallen, luego se implementa el código que haga que la prueba pase satisfactoriamente
- ❖ El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione (Del inglés: Clean code that works)
- ❖ La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que los requisitos se hayan implementado correctamente

## ➤ Ciclo de desarrollo conducido por pruebas

- ❖ Antes de comenzar el ciclo se debe definir una lista de requisitos.
- ❖ Elegir un requisito
- ❖ Escribir una prueba: Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar
- ❖ Verificar que la prueba falla: Si la prueba no falla es porque el requerimiento ya estaba implementado o porque la prueba es errónea
- ❖ Escribir la implementación: Escribir el código más sencillo que haga que la prueba funcione
- ❖ Ejecutar las pruebas automatizadas: Verificar si todo el conjunto de pruebas funciona correctamente
- ❖ Eliminación de duplicación: El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado
- ❖ Actualización de la lista de requisitos: Se actualiza la lista de requisitos tachando el requisito implementado

## Ejemplo jUnit

- Implementación de la batería de Test de la Clase HashSet de Java.
- Implementación de la batería de Test de la Clase Queue de Java. Concretamente
  - ❖ Add / offer
  - ❖ Peek / Element
  - ❖ Poll / Remove
  - ❖ IsEmpty
  - ❖ Clear
- Implementar la batería de estado de la clase Queue/HashSet



## ➤ HashSet

```
public class TestHashSet {

    private String e1 = "elemento 1";
    private String e2 = "elemento 2";
    private String e3 = "elemento 3";
    private String e4 = "elemento 4";
    private String e5 = "elemento 5";

    public TestHashSet() {
        // constructor
        // útil si hay que inicializar variables privadas
    }

    public void testAdd() {
        Set<String> set = new HashSet<String>();
        assertEquals(0, set.size());
        assertTrue(set.add(e1));
        assertEquals(1, set.size());
        assertTrue(set.add(e2));
        assertEquals(2, set.size());
        assertFalse(set.add(e1));
        assertEquals(2, set.size());
    }
}
```

```
@Test
public void testRemove() {
    Set<String> set = new HashSet<String>();
    set.add(e1);set.add(e2);set.add(e3);
    assertEquals(3, set.size());
    assertTrue(set.remove(e2));
    assertEquals(2, set.size());
    assertFalse(set.remove(e2));
    assertEquals(2, set.size());
    assertTrue(set.remove(e1));
    assertEquals(1, set.size());
    assertTrue(set.remove(e3));
    assertEquals(0, set.size());
}
```

```
@Test
public void testClear() {
    Set<String> set = new HashSet<String>();
    set.add(e1);set.add(e2);set.add(e3);
    set.clear();
    assertEquals(0, set.size());
}
```

```
@Test
public void testIsEmpty() {
    Set<String> set = new HashSet<String>();
    assertTrue(set.isEmpty());
    set.add(e1);set.add(e2); set.add(e3);
    assertFalse(set.isEmpty());
    set.remove(e3);
    assertFalse(set.isEmpty());
    set.clear();
    assertTrue(set.isEmpty());
}
```

## ➤ HashSet

```
@Test
public void testContains() {
    Set<String> set = new HashSet<String>();
    set.add(e1); set.add(e2); set.add(e3);
    assertTrue(set.contains(e1));
    assertTrue(set.contains(e2));
    assertTrue(set.contains(e3));
    assertFalse(set.contains(e4));
    set.remove(e2);
    assertTrue(set.contains(e1));
    assertFalse(set.contains(e2));
    assertTrue(set.contains(e3));
    assertFalse(set.contains(e4));
}

@Test
public void testEquals() {
    Set<String> set1 = new HashSet<String>();
    set1.add(e1); set1.add(e2); set1.add(e3);
    Set<String> set2 = new HashSet<String>();
    assertFalse(set1.equals(set2));
    set2.add(e1);
    assertFalse(set1.equals(set2));
    set2.add(e2);
    assertFalse(set1.equals(set2));
    set2.add(e3);
    assertTrue(set1.equals(set2));
    assertTrue(set2.equals(set1));
    set2.add(e4);
    assertFalse(set1.equals(set2));
    assertFalse(set2.equals(set1));
}
```

```
@Test
public void testIteration() {
    Set<String> set1 = new HashSet<String>();
    set1.add(e1);
    set1.add(e2);
    set1.add(e3);
    Set<String> set2 = new HashSet<String>();
    for (String e : set1)
        set2.add(e);
    assertEquals(set1, set2);
}
```

```
@Test(expected = ConcurrentModificationException.class)
public void testConcurrentModification() {
    Set<String> set = new HashSet<String>();
    set.add(e1);
    set.add(e2);
    set.add(e3);
    for (String e : set)
        set.remove(e);
    assertTrue(set.isEmpty());
}
}
```

### ➤ Ejemplo de Queue a nivel de métodos

- ❖ A queue is empty on construction
- ❖ A queue has size 0 on construction
- ❖ After  $n$  enqueues to an empty queue,  $n > 0$ , the queue is not empty and its size is  $n$
- ❖ If one enqueues  $x$  then dequeues, the value dequeued is  $x$ .
- ❖ If one enqueues  $x$  then peeks, the value returned is  $x$ , but the size stays the same
- ❖ If the size is  $n$ , then after  $n$  dequeues, the stack is empty and has a size 0
- ❖ If one enqueues the values 1 through 50, in order, into an empty queue, then if 50 dequeues are done the values dequeued are 1 through 50.
- ❖ Dequeueing from an empty queue does throw a `NoSuchElementException`
- ❖ Peeking into an empty queue does throw a `NoSuchElementException`
- ❖ For bounded queues only, pushing onto a full stack does throw an `IllegalStateException`



## ➤ Ejemplo de Queue a nivel de métodos

```
@Test
public void testPeek() {
    Queue q = new LinkedList();
    Object o = q.peek();
    assertNull(o);
    q.add(e1);
    o = q.peek();
    assertNotNull(o);
}

@Test
public void testElement() {
    Queue q = new LinkedList();
    try {
        Object o = q.element();
        Fail();
    } catch (NoSuchElementException e) {
    }
    q.add(e1);
    try {
        Object o = q.element();
    } catch (NoSuchElementException e) {
        fail();
    }
}
```

- Una clase EmpleadoBR con las reglas de negocio aplicables a los empleados de una tienda.
  - ❖ float calculaSalarioBruto( TipoEmpleado tipo, float ventasMes, float horasExtra)
    - ✓ El salario base será 1000 euros si el empleado es de tipo TipoEmpleado.vendedor, y de 1500 euros si es de tipo TipoVendedor.encargado. A esta cantidad se le sumará una prima de 100 euros si ventasMes es mayor o igual que 1000 euros, y de 200 euros si fuese al menos de 1500 euros. Por último, cada hora extra se pagará a 20 euros. Si tipo es null, o ventasMes o horasExtra toman valores negativos el método lanzará una excepción de tipo BRException.
  - ❖ float calculaSalarioNeto(float salarioBruto)
    - ✓ Si el salario bruto es menor de 1000 euros, no se aplicará ninguna retención. Para salarios a partir de 1000 euros, y menores de 1500 euros se les aplicará un 16%, y a los salarios a partir de 1500 euros se les aplicará un 18%. El método nos devolverá salarioBruto \* (1-retencion), o BRExcepcion si el salario es menor que cero.
- A partir de dichas especificaciones, diseñar un conjunto de casos de prueba

Obtenido de: <http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html>

## ➤ Casos

Método a probar	Entrada	Salida esperada
calculaSalarioNeto	2000	1640
calculaSalarioNeto	1500	1230
calculaSalarioNeto	1499.99	1259.9916
calculaSalarioNeto	1250	1050
calculaSalarioNeto	1000	840
calculaSalarioNeto	999.99	999.99
calculaSalarioNeto	500	500
calculaSalarioNeto	0	0
calculaSalarioNeto	-1	BRException
calculaSalarioBruto	vendedor, 2000 euros, 8h	1360
calculaSalarioBruto	vendedor, 1500 euros, 3h	1260
calculaSalarioBruto	vendedor, 1499.99 euros, 0h	1100
calculaSalarioBruto	encargado, 1250 euros, 8h	1760
calculaSalarioBruto	encargado, 1000 euros, 0h	1600
calculaSalarioBruto	encargado, 999.99 euros, 3h	1560
calculaSalarioBruto	encargado, 500 euros, 0h	1500
calculaSalarioBruto	encargado, 0 euros, 8h	1660
calculaSalarioBruto	vendedor, -1 euros, 8h	BRException
calculaSalarioBruto	vendedor, 1500 euros, -1h	BRException
calculaSalarioBruto	null, 1500 euros, 8h	BRException

/sesion04-apuntes.html

## ➤ Ejemplo Junit (faltarían más casos)

```
public class EmpleadoBRTTest {
    @Test
    public void testCalculaSalarioBruto1() {
        float resultadoReal = EmpleadoBR.calculaSalarioBruto(TipoEmpleado.vendedor, 2000.0f, 8.0f);
        float resultadoEsperado = 1360.0f;
        assertEquals(resultadoEsperado, resultadoReal, 0.01);
    }
    @Test
    public void testCalculaSalarioBruto2() {
        float resultadoReal = EmpleadoBR.calculaSalarioBruto(TipoEmpleado.vendedor, 1500.0f, 3.0f);
        float resultadoEsperado = 1260.0f;
        AssertEquals(resultadoEsperado, resultadoReal, 0.01);
    }
    @Test
    public void testCalculaSalarioNeto1() {
        float resultadoReal = EmpleadoBR.calculaSalarioNeto(2000.0f);
        float resultadoEsperado = 1640.0f;
        assertEquals(resultadoEsperado, resultadoReal, 0.01);
    }
    @Test
    public void testCalculaSalarioNeto2() {
        float resultadoReal = EmpleadoBR.calculaSalarioNeto(1500.0f);
        float resultadoEsperado = 1230.0f;
        assertEquals(resultadoEsperado, resultadoReal, 0.01);
    }
}
```

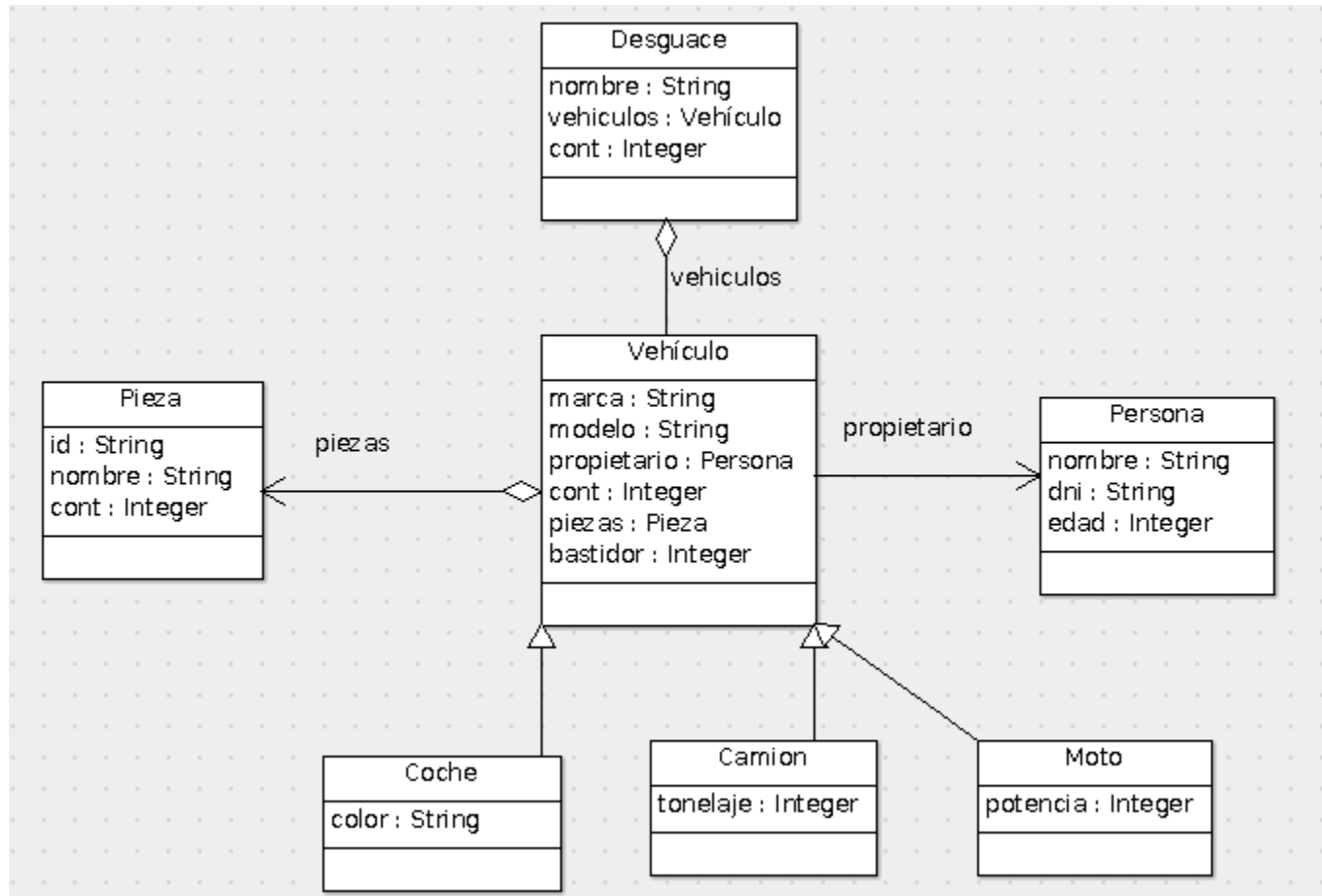
Obtenido de: <http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html>

## ➤ Junit con excepciones(faltarían más casos)

```
@Test
public void testCalculaSalarioNeto9() {
    try {
        EmpleadoBR.calculaSalarioNeto(-1.0f);
        fail("Se esperaba excepcion BRException");
    } catch (BRException e) {
    }
}

public void testCalculaSalarioBruto1() {
    float resultadoReal;
    try {
        resultadoReal = EmpleadoBR.calculaSalarioBruto(TipoEmpleado.vendedor, 2000.0f, 8.0f);
        float resultadoEsperado = 1360.0f;
        assertEquals(resultadoEsperado, resultadoReal, 0.01);
    } catch (BRException e) {
        fail("Lanzada excepcion no esperada BRException");
    }
}
```

- Implementación de la batería de Test de Vehículo con un Array de Piezas donde el tamaño del array es 3 por defecto



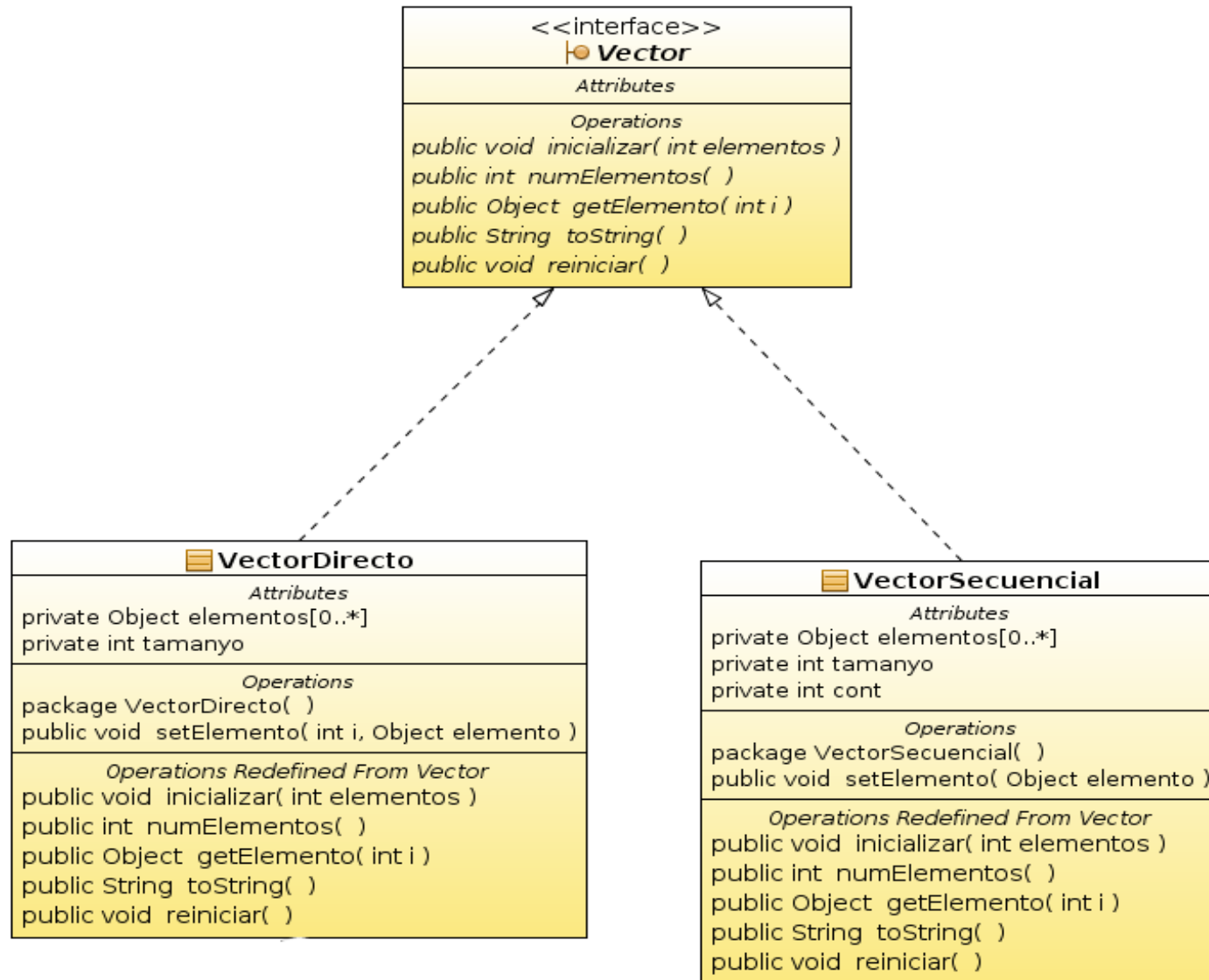
# Ejemplo junit

## ➤ TestVehiculo.java

```
public class TestVehiculo {
    private Vehiculo v1, v2, v3, v4, v_v1, v_v2, v_v3;
    private Persona p;
    @Before
    public void setUp() {
        v_v1 = new Vehiculo();
        p = new Persona("Luis", "4", 4);
        v_v2 = new Vehiculo("Renault", "Laguna", p, 123);
        v_v3 = new Vehiculo(v_v2);
    }
    @After
    public void tearDown() {
        v_v1 = null; v_v2 = null; v_v3 = null; p = null;
    }
    /***** Test vehiculo *****/
    @Test
    public void testVehiculo() {
        assertNotNull(v_v1);
        assertNotNull(v_v2);
        assertNotNull(v_v3);
        assertEquals(v_v2, v_v3);
    }
    @Test
    public void testGetMarca() {
        String Marca = v_v2.getMarca();
        assertEquals("Renault", Marca);
    }
    @Test
    public void testSetMarca() {
        v_v2.setMarca("Fiat");
        assertEquals("Fiat", v_v2.getMarca());
    }
}
```

```
@Test
public void testGetModelo() {
    String modelo = v_v2.getModelo();
    assertEquals("Laguna", modelo);
}
@Test
public void testSetModelo() {
    v_v2.setModelo("Laguna");
    assertEquals("Laguna", v_v2.getModelo());
}
@Test
public void testGetAddPieza() {
    Pieza p1 = new Pieza("1", "Freno", 2);
    Pieza p4 = new Pieza("4", "Bujia", 1);
    Pieza p5 = new Pieza("4", "Bujia", 1); // duplicado id
    Pieza p7 = new Pieza("5", "Otra", 1);
    Pieza p8 = new Pieza("6", "Otra", 1); // No hay tamaño
    assertTrue(v_v2.addPiezaV(p1));
    assertTrue(v_v2.addPiezaV(p4));
    assertFalse(v_v2.addPiezaV(p5));
    assertTrue(v_v2.addPiezaV(p7));
    assertFalse(v_v2.addPiezaV(p8));
}
@Test
public void testGetPieza() {
    Pieza p1 = new Pieza("1", "Freno", 2);
    Pieza p2 = new Pieza("2", "Faro", 5);
    Pieza p3 = new Pieza("3", "Cambio", 1);
    addPiezaV(p1);
    addPiezaV(p2);
    assertTrue(v_v2.addPiezaV(p3));
    assertNotNull(v_v2.getPiezaV(0));
    assertEquals(p1, v_v2.getPiezaV(0));
    assertNull(v_v2.getPiezaV(10));
    assertNull(v_v2.getPiezaV(-5));
    assertEquals(3, v_v2.getCont());
}
}
```

## ➤ Bateria de Test de VectorDirecto





## ➤ Ejemplo de VectorDirecto (1/4)

```
public class VectorDirectoTest extends TestCase {
    @Test
    public void testInicializar() {
        System.out.println("inicializar");
        int elementos = 10;
        VectorDirecto instance = new VectorDirecto();
        instance.Inicializar(elementos);
        assertEquals(elementos, instance.Tamanyo());
    }

    @Test
    public void testGetTamanyo() {
        System.out.println("inicializar");
        int elementos = 10;
        VectorDirecto instance = new VectorDirecto();
        instance.Inicializar(elementos);
        assertEquals(elementos, instance.Tamanyo());
    }

    @Test
    public void testNumElementos() {
        System.out.println("numElementos");
        VectorDirecto instance = new VectorDirecto();
        try {
            instance.setElemento(1, new Fregadero());
        } catch (FueraRango ex) {
            fail("Fallo en la prueba.");
        }
        int expResult = 1;
        int result = instance.numElementos();
        assertEquals(expResult, result);
    }
}
```

## ➤ Ejemplo de VectorDirecto (2/4)

```
@Test
public void testGetElemento1() {
    System.out.println("getElemento: Excepcion dato no valido");
    int i = 0;
    VectorDirecto instance = new VectorDirecto(10);
    Object expResult = null;
    Object result=null;
    try {
        result = instance.getElemento(1);
        fail("Fallo en la prueba.");
    } catch (DatoNoValido ex) {
        assertTrue(true);
    } catch (FueraRango ex) {
        fail("Fallo en la prueba.");
    }
}

@Test
public void testGetElemento2() {
    System.out.println("getElemento: Excepcion Fuera de Rango");
    VectorDirecto instance = new VectorDirecto(10);
    Object result=null;
    try {
        result = instance.getElemento(12);
        fail("Fallo en la prueba.");
    } catch (DatoNoValido ex) {
        fail("Fallo en la prueba.");
    } catch (FueraRango ex) {
        assertTrue(true);
    }
}
```

## ➤ Ejemplo de VectorDirecto (3/4)

```
@Test
public void testGetElemento3() {
    System.out.println("getElemento");
    VectorDirecto instance = new VectorDirecto(10);
    Object result=null;
    try {
        instance.setElemento(1, new Persona());
        result = instance.getElemento(1);
        assertNotNull(result);
    } catch (DatoNoValido ex) {
        fail("Fallo en la prueba.");
    } catch (FueraRango ex) {
        fail("Fallo en la prueba.");
    }
}

@Test
public void testSetElemento1() {
    System.out.println("setElemento");
    VectorDirecto instance = new VectorDirecto(10);
    try {
        instance.setElemento(12, new Fregadero());
        fail("Fallo en la prueba.");
    } catch (FueraRango ex) {
        assertTrue(true);
    }
}
```

## ➤ Ejemplo de VectorDirecto (4/4)

```
@Test
public void testSetElemento2() {
    System.out.println("setElemento");
    VectorDirecto instance = new VectorDirecto(10);
    try {
        Persona f= new Persona();
        instance.setElemento(1, f);
        try {
            assertEquals(instance.getElemento(1),f);
        } catch (DatoNoValido ex) {
            fail("Fallo en la prueba.");
        }
    } catch (FueraRango ex) {
        fail("Fallo en la prueba.");
    }
}

@Test
public void testReiniciar() {
    System.out.println("reiniciar");
    VectorDirecto instance = new VectorDirecto(10);
    try {
        Persona f = new Persona("Luis", 30,"30");
        instance.setElemento(1, f);
        instance.Reiniciar();
        assertTrue(instance.numElementos() == 0);

    } catch (FueraRango ex) {
        fail("Fallo en la prueba.");
    }
}
```

# Bibliografía

---

- **Piensa en Java. 4ª Edición.** Bruce Eckel. Pearson Prentice Hall.
- Libros de JUnit
  - ❖ JUnit Pocket Guide. Beck, Kent. O'REILLY & ASSOCIATES. 84 páginas
  - ❖ Pruebas de software y junit. Pearson.
- URL:
  - ❖ Apuntes de Ingeniería del Software I de la Univesidad de Cantabria:  
[personales.unican.es/ruizfr/is1/doc/teo/05/is1-t05-trans.pdf](http://personales.unican.es/ruizfr/is1/doc/teo/05/is1-t05-trans.pdf)
  - ❖ <http://www.junit.org>
  - ❖ Testing with JUnit en [java.sun.com/developer/Books](http://java.sun.com/developer/Books)
  - ❖ Libro: Diseño Ágil con TDD. <http://www.dirigidoportests.com/el-libro>
  - ❖ Libro: JUnit in Action (algunos capítulos online)  
<http://manning.com/tahchiev/>
- **MP3. Dos ficheros:**
  - ❖ [http://www.javahispano.org/podcasts/064\\_JavahispanoPodcast\\_Test1.mp3](http://www.javahispano.org/podcasts/064_JavahispanoPodcast_Test1.mp3)
  - ❖ [http://www.javahispano.org/podcasts/065\\_JavahispanoPodcast\\_Test2.mp3](http://www.javahispano.org/podcasts/065_JavahispanoPodcast_Test2.mp3)