# Introducción a Java

Práctica de laboratorio 1

Aplicaciones Telemáticas
ETSIT-UPV

F.J. Martínez Zaldívar

## Índice

# 1. Introducción y objetivos

En esta primera práctica se pretenden ejercitar algunas conceptos básicos de Java que desarrollan las herramientas de herencia, paquetes, modificadores, ubicación de ficheros fuente y clases, etc.

Para ello, se formarán grupos de una o dos personas. Si el grupo está formado por dos alumnos, ambos entregarán los mismos resultados de manera individual y replicada (por **duplicado**) haciendo constar claramente los integrantes del grupo.

## 1.1. Requisitos

Para realizar esta práctica es necesario disponer de un ordenador con el *Java Development Kit* o JDK convenientemente instalado, actualizando la variable `PATH` del sistema para que los ejecutables `javac, java, ...` estén disponibles en línea de comandos desde cualquier directorio de trabajo. Para realizar la instalación se deberían seguir las directrices dadas en clase. Existe la posibilidad de que ya exista una instalación de `Java` en el ordenador a utilizar; para verificarlo, ábrase una ventana de comandos y ejecútese el comando `javac` para observar su comportamiento. También se recomienda trabajar con un editor de textos cómodo como alguno de los propuestos en clase (*Sublime Text, Atom, ...*).

# 2. Paquetes, ficheros y su estructura

Se proporcionan varios ficheros fuente, algunos incompletos, y se pretenden cubrir los objetivos que se plantean en la práctica, al ser capaces de rellenar o corregir correctamente estos ficheros, haciendo que la ejecución de cierto test proporcione unos resultados esperados. (Los ficheros pueden ser generados copiando y pegando del propio documento PDF, prestando atención que el proceso de copia no produzca caracteres extraños.) Para ello:

- Se deben identificar los paquetes a los que van a pertenecer ciertas clases descritas en los ficheros fuente Java, teniendo en cuenta los contenidos de los ficheros
- Se deben ubicar convenientemente en ciertos subdirectorios, atendiendo a los requisitos impuestos por los paquetes
- Se deben rellenar o modificar ciertos ficheros fuente
- Se deben compilar todos los ficheros

- Se debe realizar una ejecución, comprobando que los resultados son los esperados.

Aunque el problema concreto planteado pueda resultar algo *artificioso*, valdrá para verificar el correcto conocimiento de los conceptos necesarios. El ejemplo de prueba va a girar en torno a ciertas operaciones que se van a realizar con vectores de 2 o de 3 dimensiones.

## 2.1. Interfaces

Para ello dispondremos de un par de interfaces:

### 2.1.1. Interfaz `EspacioVectorial2D`

EspacioVectorial2D.java

```
package geometria.espacios;

public interface EspacioVectorial2D {

  Vector2D sumar( Vector2D v );

  Vector2D escalar ( double a );

}
```

- `Vector2D sumar( Vector2D v )`: se debe realizar la suma del vector que representa el objeto que implementa la interfaz, con el vector especificado en el parámetro, dejando el resultado en el vector correspondiente al objeto actual y devolviéndolo también.
- `Vector2D escalar ( double a )`: debe escalar el vector que representa el objeto que implementa la interfaz con el escalar `a`, guardando el resultado en el objeto en cuestión y devolviéndolo igualmente.

### 2.1.2. Interfaz `OperacionesVector2D`

OperacionesVector2D.java

```
package geometria.vectores;

interface OperacionesVector2D {

  double norma( );

  Vector2D rotar( double angulo);

  Vector2D restar( Vector2D v );
```

```
}
```

La interfaz `OperacionesVector2D` obliga a la implementación de los siguientes métodos a aquella clase que la implemente:

- `double norma()`: debe devolver la norma o módulo del vector correspondiente al objeto perteneciente a la clase que implementa esta interfaz.
- `Vector2D rotar( double angulo )`: rota el vector correspondiente al objeto perteneciente a la clase que implementa esta interfaz, un ángulo $\theta$ rad en sentido horario. Para ello podemos emplear una matriz de rotación de dos dimensiones:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta + y\sin\theta \\ -x\sin\theta + y\cos\theta \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix}$$

- `Vector2D restar( Vector2D v )`: resta el vector correspondiente al objeto perteneciente a la clase que implementa esta interfaz con el vector del parámetro, dejando el resultado en el propio objeto y devolviéndolo también.

## 2.2. Clases

### 2.2.1. Clase `Vector2D`

Adicionalmente, tenemos la clase `Vector2D` cuya implementación parcial es la siguiente:

Vector2D.java

```
package geometria.vectores;

public class Vector2D
      implements geometria.espacios.EspacioVectorial2D,
                 OperacionesVector2D {

  ...

} // Vector2D
```

Esta clase deberá rellenarse completamente, habida cuenta que debe ser correcta para poderse verificar su correcto funcionamiento en un test que se describirá posteriormente y además debe servir de clase padre o superclase, para la clase `Vector3D` descrita completamente a continuación.

### 2.2.2. Clase `Vector3D.java`

En esta clase se reescribirán algunos métodos heredados de `Vector2D` para que las operaciones en cuestión tengan sentido en tres dimensiones.

Vector3D.java

```
public class Vector3D extends Vector2D  {

  private double z;

  public Vector3D (double x, double y, double z) {
    super( x, y );
    this.z = z;
  }

  // Esta sobrecarga es un constructor de copia
  public Vector3D (Vector3D v) {
    this( v.getX(), v.getY(), v.getZ() );
  }

  public double getZ() {
    return z;
  }

  public Vector3D sumar( Vector3D v ) {
    super.sumar( (Vector2D) v );
    z += v.getZ();
    return this;
  }

  public Vector3D restar( Vector3D v ) {
    super.restar( (Vector2D) v );
    z -= v.getZ();
    return this;
  }


  public Vector3D escalar( double alfa ) {
    super.escalar( alfa );
    z *= alfa;
    return this;
  }

  public double norma() {
    return Math.sqrt( super.norma()*super.norma() + z*z );
  }

  public Vector3D rotar(int dimA, int dimB, double angulo) {
    Vector2D v;
    if ( dimA == 0 && dimB == 1 || dimA == 1 && dimB == 0 ) {
```

```
      v = new Vector2D(x, y).rotar(angulo);
      x = v.getX();
      y = v.getY();
    }
    else if (dimA == 0 && dimB == 2 || dimA == 2 && dimB == 0 ) {
      v = new Vector2D(x, z).rotar(angulo);
      x = v.getX();
      z = v.getY();
    }
    else if (dimA == 1 && dimB == 2 || dimA == 2 && dimB == 1 ) {
      v = new Vector2D(y, z).rotar(angulo);
      y = v.getX();
      z = v.getY();
    }
    return this;
  }

  public String toString() {
    return "(" + x + ", " + y +", " + z + ")";
  }

}  // Vector3D
```

El método `rotar` en 3D requiere alguna explicación adicional, ya que hay que especificar el planos o las dos dimensiones de las tres de las que consta el vector de 3D, en los que se efectúa la rotación. Numeraremos las dimensiones comenzando por la cero: en dos dimensiones, dimensión 0 y dimensión 1; en tres dimensiones, dimensión 0, dimensión 1 y dimensión 2. Por ejemplo, supongamos el siguiente vector tridimensional de partida:

$$\mathbf{v} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}$$

Si deseamos rotar el vector sólo en el plano definido por las dimensiones 1 y 2 un ángulo de $\theta$ rad, podríamos hacerlo de la siguiente manera:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1\cos\theta + v_2\sin\theta \\ -v_1\sin\theta + v_2\cos\theta \end{pmatrix} \Rightarrow \begin{pmatrix} v_0 \\ v_1' \\ v_2' \end{pmatrix}$$

O si por ejemplo, la rotación la queremos hacer en el plano descrito por las componentes 0 y 2:

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_0\cos\theta + v_2\sin\theta \\ v_1 \\ -v_0\sin\theta + v_2\cos\theta \end{pmatrix} \Rightarrow \begin{pmatrix} v'_0 \\ v_1 \\ v'_2 \end{pmatrix}$$

Y así con toda la casuística posible.

Si nos centramos exclusivamente en las componentes de interés $v_{\mathtt{dim}_A} = v_0$ y $v_{\mathtt{dim}_B} = v_2$, como en este último caso, entonces:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} v_{\mathtt{dim}_A} \\ v_{\mathtt{dim}_B} \end{pmatrix} = \begin{pmatrix} v_{\mathtt{dim}_A}\cos\theta + v_{\mathtt{dim}_B}\sin\theta \\ -v_{\mathtt{dim}_A}\sin\theta + v_{\mathtt{dim}_B}\cos\theta \end{pmatrix} \Rightarrow \begin{pmatrix} v'_{\mathtt{dim}_A} \\ v'_{\mathtt{dim}_B} \end{pmatrix}$$

### 2.2.3. Clase principal `Test`

Por último, la clase que contiene el *programa principal* o método `main` será la siguiente:

Test.java

```
package test;

class Test {

  public static void main ( String args[] ) {

    // 2D

    Vector2D v = new Vector2D( -3, 4.5 );
    System.out.println( "v = " + v );

    Vector2D w = new Vector2D( 5, 2.3 );
    System.out.println( "w = " + w );

    System.out.println( "w += v --> w = " + w.sumar(v) );

    System.out.println( "w -= v --> w = " + w.restar(v) );

    System.out.println( "|w| = " + w.norma() );

    System.out.println("rotar w 90 grados:"
                       + w.rotar( 90*Math.PI/180 /* radianes */) );

    System.out.println( "|w| = " + w.norma() );


    //3D
```

```
    Vector3D a = new Vector3D( 1., 2., 3. );
    System.out.println( "a = " + a );

    Vector3D b = new Vector3D( 10., 20., 30. );
    System.out.println( "b = " + b );

    System.out.println( "b += a --> b = " + b.sumar(a) );

    System.out.println( "b -= a --> b = " + b.restar(a) );

    System.out.println( "|b| = " + b.norma() );

    System.out.println( "rotar dimensiones 0 y 1 de b 90 grados --> b = "
                        + b.rotar( 0, 1, 90*Math.PI/180 ) );

    System.out.println( "|b| = " + b.norma() );

    System.out.println( "rotar dimensiones 0 y 2 de b 90 grados --> b = "
                        + b.rotar( 0, 2, 90*Math.PI/180 ) );

    System.out.println( "|b| = " + b.norma() );

    System.out.println( "rotar dimensiones 1 y 2 de b 90 grados --> b = "
                        + b.rotar( 2, 1, 90*Math.PI/180 ) );

    System.out.println( "|b| = " + b.norma() );

  } // main

} // Test
```

La salida correspondiente a la ejecución del anterior test debería ser:

salida por pantalla

```
v = (-3.0, 4.5)
w = (5.0, 2.3)
w += v --> w = (2.0, 6.8)
w -= v --> w = (5.0, 2.3)
|w| = 5.503635162326805
rotar w 90 grados:(2.3000000000000003, -5.0)
|w| = 5.503635162326805
a = (1.0, 2.0, 3.0)
b = (10.0, 20.0, 30.0)
b += a --> b = (11.0, 22.0, 33.0)
b -= a --> b = (10.0, 20.0, 30.0)
|b| = 37.416573867739416
rotar dimensiones 0 y 1 de b 90 grados --> b = (20.0, -9.999999999999998, 30.0)
|b| = 37.416573867739416
rotar dimensiones 0 y 2 de b 90 grados --> b = (30.0, -9.999999999999998, -19.999999999999996)
```

```
|b| = 37.416573867739416
rotar dimensiones 1 y 2 de b 90 grados --> b = (30.0, -19.999999999999996, 9.999999999999996)
|b| = 37.416573867739416
```

Compruébese que efectivamente la ejecución del test proporciona estos resultados.

### 2.3. Formato de los datos numéricos

Estúdiese el método estático `format` de la clase `String` que permite formatear, por ejemplo, un resultado numérico, requiriendo un formato con notación exponencial de tres decimales (por ejemplo: `3.300E+01`). Modifíquense conveniententemente los métodos `toString()` de las clases `Vector2D` y `Vector3D` para conseguir este formateado de la información numérica. Indáguese sobre qué *string de control de formato* sería el apropiado si se desea dejar un *hueco* en el que poner el signo menos si la cantidad es negativa y nada si es positiva. Indíquese de qué forma se puede emplear bien la coma, bien el punto como separador entre unidades y decimales (sugerencia: véase la clase de Java `Locale`).

## 3. Entrega de resultados

Debe mostrarse al profesor la correcta ejecución del software realizado, las secuencias de comandos de compilación y ejecución y por tanto los resultados de ejecutar los tests.

Lo realizado durante la sesión, junto con las respuestas requeridas añadidas como comentarios oportunos al código, deben comprimirse bien en un fichero `zip`(o bien en un `jar`), y depositarse en el espacio compartido de Poliformat, concretamente en un directorio creado por el alumno y denominado `P1`. Recuérdese añadir en el directorio creado un fichero de texto denominado `GRUPO.txt` con el o los nombres de los integrantes del grupo y cualquier consideración que se desee hacer. Recuérdese que si el grupo lo integran dos personas, toda la documentación debe subirse por duplicado.