

Programación

JAVA INTRODUCCION

1º DESARROLLO DE
APLICACIONES
MULTIPLATAFORMA (D.A.M.)
2021-2022

Java es un lenguaje de programación comercializado por primera vez en 1995 por Sun Microsystems.

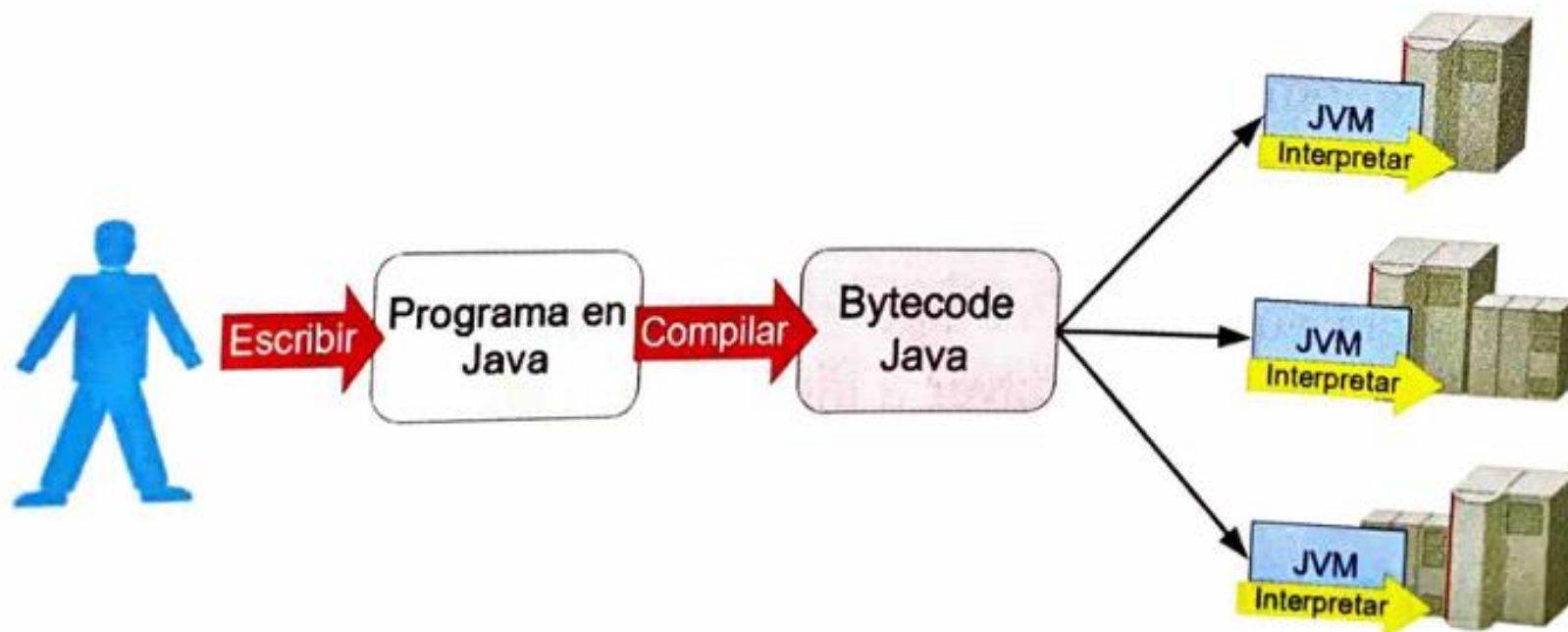
Java fue desarrollado por James Gosling, de Sun Microsystems (constituido en 1983 y posteriormente adquirido en el 2010 por la compañía Oracle). Su sintaxis deriva de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.



Java se concibió como un lenguaje para internet y, por lo tanto, necesita ser multiplataforma, para que un mismo programa se compile una única vez y pueda ser ejecutado en multitud de ordenadores y sistemas operativos completamente diferentes. El compilador de Java no genera un código máquina dependiente de ninguna plataforma; en su lugar, genera un código binario especial llamado bytecode de Java. Este no es ejecutable directamente por ningún ordenador, ya que es independiente de cualquier plataforma y ha sido ideado como un código intermedio por los implementadores de Java. Para poder ejecutarlo, la solución está en disponer de un interprete en cada equipo , que traduce el bytecode de Java al código máquina nativo de cada plataforma.

Al programa que interpreta el bytecode se le conoce como Máquina Virtual de Java o JVM, por sus siglas en inglés.

De esta forma, se consigue que Java sea un lenguaje multiplataforma: un mismo programa, una vez compilado, se puede ejecutar en cualquier ordenador que tenga instalada la maquina virtual de Java, que no es mas que un interprete e bytecode.



Tiempo de compilación: es el espacio de tiempo en el que se traduce el código fuente al bytecode.

Tiempo de ejecución: es el tiempo durante el cual, el bytecode se interpreta (por la JVM) y se ejecuta por la plataforma correspondiente.



Conceptos básicos de Programación Orientada a Objetos:

Objeto:

Se trata de un ente abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras.

Los objetos integran, a diferencia de los métodos procedurales, tanto los procedimientos como las variables y datos referentes al objeto.

A los objetos se les otorga ciertas características en la vida real. Cada parte del programa que se desea realizar es tratado como objeto, siendo así estas partes independientes las unas de las otras. Los objetos se componen de 3 partes fundamentales: métodos, eventos y atributos.

Métodos:

Son aquellas funciones que permite efectuar el objeto y que nos rinden algún tipo de servicio durante el transcurso del programa.

Determinan a su vez como va a responder el objeto cuando recibe un mensaje.

Eventos:

Son aquellas acciones mediante las cuales el objeto reconoce que se está interactuando con él.

De esta forma el objeto se activa y responde al evento según lo programado en su código.

Atributos:

Características que aplican al objeto solo en el caso en que el sea visible en pantalla por el usuario; entonces sus atributos son el aspecto que refleja, tanto en color, tamaño, posición, si está o no habilitado.

Ejemplo de objeto: un coche.

Se considera un coche un objeto, totalmente diferenciado del alquitrán donde está aparcado.

Sus atributos son su color, marca, modelo, número de matrícula, número de puertas,...

Sus eventos todos aquellas acciones por las cuales si el coche tuviera vida propia reconocería que le estamos dando un uso, como abrir la puerta, girar el volante, embragar, abrir el capot,

Los métodos son todo aquello que nos ofrece el coche como hacer sonar una bocina cuando tocamos el claxon (evento), llevarnos por la carretera, reducir la velocidad al pisar el freno (evento), ...

Mensajes:

Aunque los objetos se han de diferenciar claramente en una aplicación, estos se han de poder comunicar para poder trabajar en conjunto y construir así aplicaciones.

Esto se hace posible a través de lo que se denomina paso de mensajes. Cuando un objeto quiere comunicarse con otro lo que hace es enviarle un mensaje con los datos que desea transmitir.

En el simil del coche, al apretar el claxon, el objeto claxon envía un mensaje a la bocina indicándole que haga sonar cierto sonido.

La potencia de este sistema radica en que el objeto emisor no necesita saber la forma en que el objeto receptor va a realizar la acción. Simplemente este la ejecuta y el emisor se desentiende del como; de hecho ni le importa, solo tiene conocimiento de que se está realizando.

Para que todo esto sea posible es necesario una buena programación de los eventos y de los métodos de cada objeto.

El conjunto de mensajes a los que un objeto puede responder se denomina protocolo del objeto.

Instancia:

Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro, menos la clase Object que es la madre de todas.

Clases: Descripción de objeto. Consta de una serie de métodos y datos que resumen las características de este objeto. Definir clases permite trabajar con código reutilizable. Puesto que desde una clase se puede crear una instancia y así reutilizar el código escrito para esta si tener que volver a escribir el código para la instancia. La instancia toma el patrón de la clase padre. Sin embargo, las variables son independientes.

Herencia:

Mecanismo para compartir automáticamente métodos y datos entre clases, subclases y objetos.

Permite crear nuevas clases introduciendo las variaciones con respecto a su clase padre.

Herencia simple: una subclase puede heredar datos y métodos de una clase simple así como añadir o sustraer ciertos comportamientos.

Herencia múltiple: posibilidad de adquirir métodos y datos de varias clases simultáneamente.

Encapsulación:

Define el comportamiento de una clase u objeto que tiene dentro de él todo tipo de métodos y datos pero que solo es accesible mediante el paso de mensajes. y los datos a través de los métodos del objeto/clase.

Polimorfismo:

Los objetos responden a los mensajes que se les envían. Un mismo mensaje puede ser interpretado o dar paso a distintas acciones según que objeto es el destinatario.

Con este sistema el emisor se desentiende de los detalles de la ejecución (aunque el programador ha de saber en todo momento cuales son las consecuencias de ese mensaje).

Tiempo real durante el cual el objeto existe en memoria:

Los objetos se crean a medida que estos son requeridos (en vez de todos a la vez, con la consiguiente pérdida de memoria) y se eliminan de la misma forma.

Persistencia: La **persistencia de datos** es un medio mediante el cual una aplicación puede recuperar información desde un sistema de almacenamiento no volátil y hacer **que** esta persista. La **persistencia de datos** es vital en las aplicaciones empresariales debido al acceso necesario a las bases de **datos** relacionales.


Abstracción: La **abstracción** consiste en seleccionar datos de un conjunto más grande para mostrar solo los detalles relevantes del objeto. Ayuda a reducir la complejidad y el esfuerzo de programación. En **Java**, la **abstracción** se logra usando clases e interfaces abstractas. Es uno de los conceptos más importantes de POO.


1 - Instalación de Java


Para poder hacer este curso debemos instalar el compilador de Java y la máquina virtual de Java. Estas herramientas las podemos descargar de:

[Java SE Development Kit.](#)

Si disponemos de un sistema operativo Windows debemos descargar:



 [Products](#) [Industries](#) [Resources](#) [Support](#) [Events](#) [Developer](#)

 [View Accounts](#)

JDK 17 will receive updates under these terms, until at least September 2024.

Java SE Development Kit 17.0.1 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

[Linux](#) [macOS](#) [Windows](#)

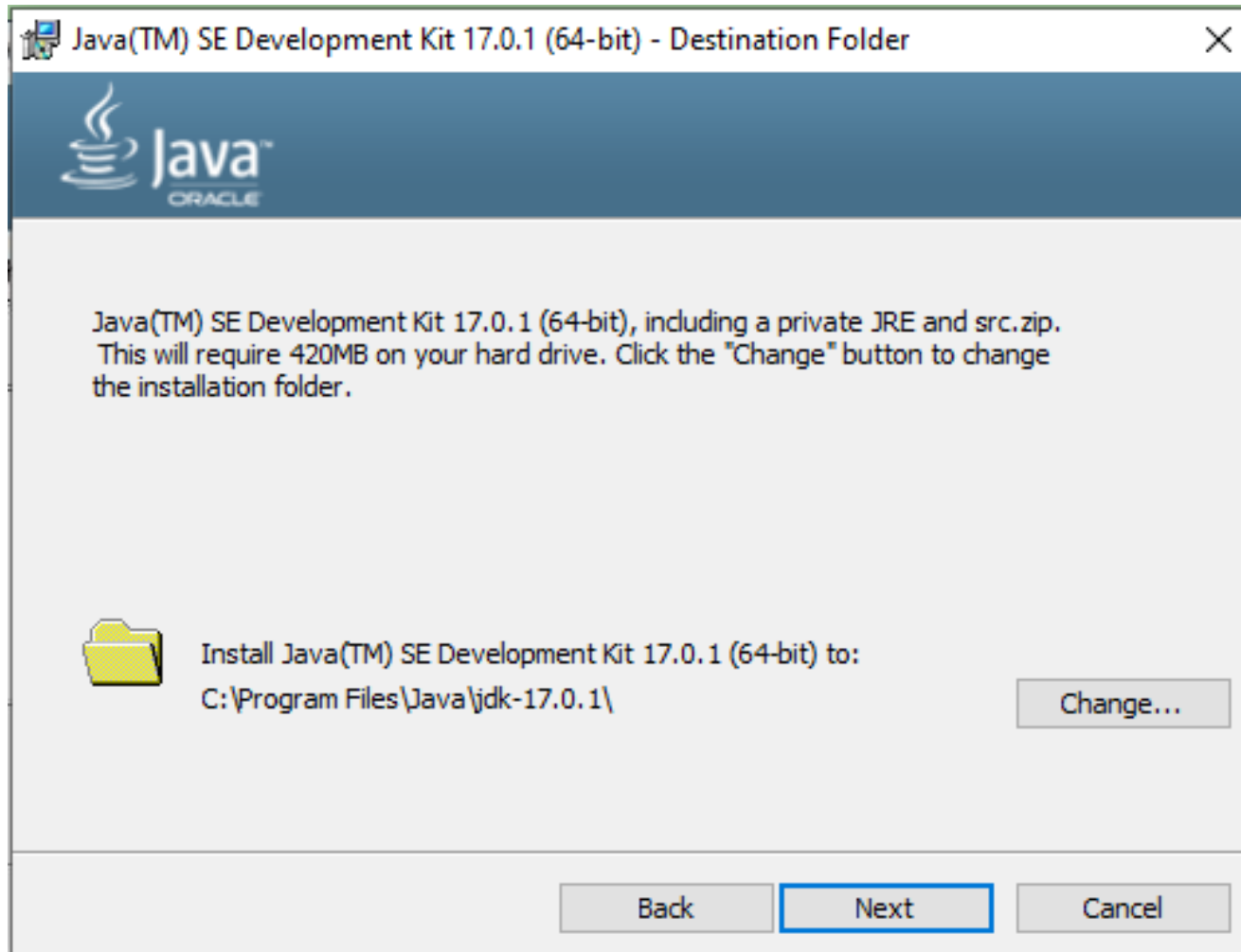
| Product/file description | File size | Download |
|--------------------------|-----------|---|
| x64 Compressed Archive | 170.66 MB | https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256 ↗) |
| x64 Installer | 152 MB | https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256 ↗) |
| x64 MSI Installer | 150.89 MB | https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256 ↗) |

La versión a instalar conviene que sea la última (en este momento disponemos la versión 17)

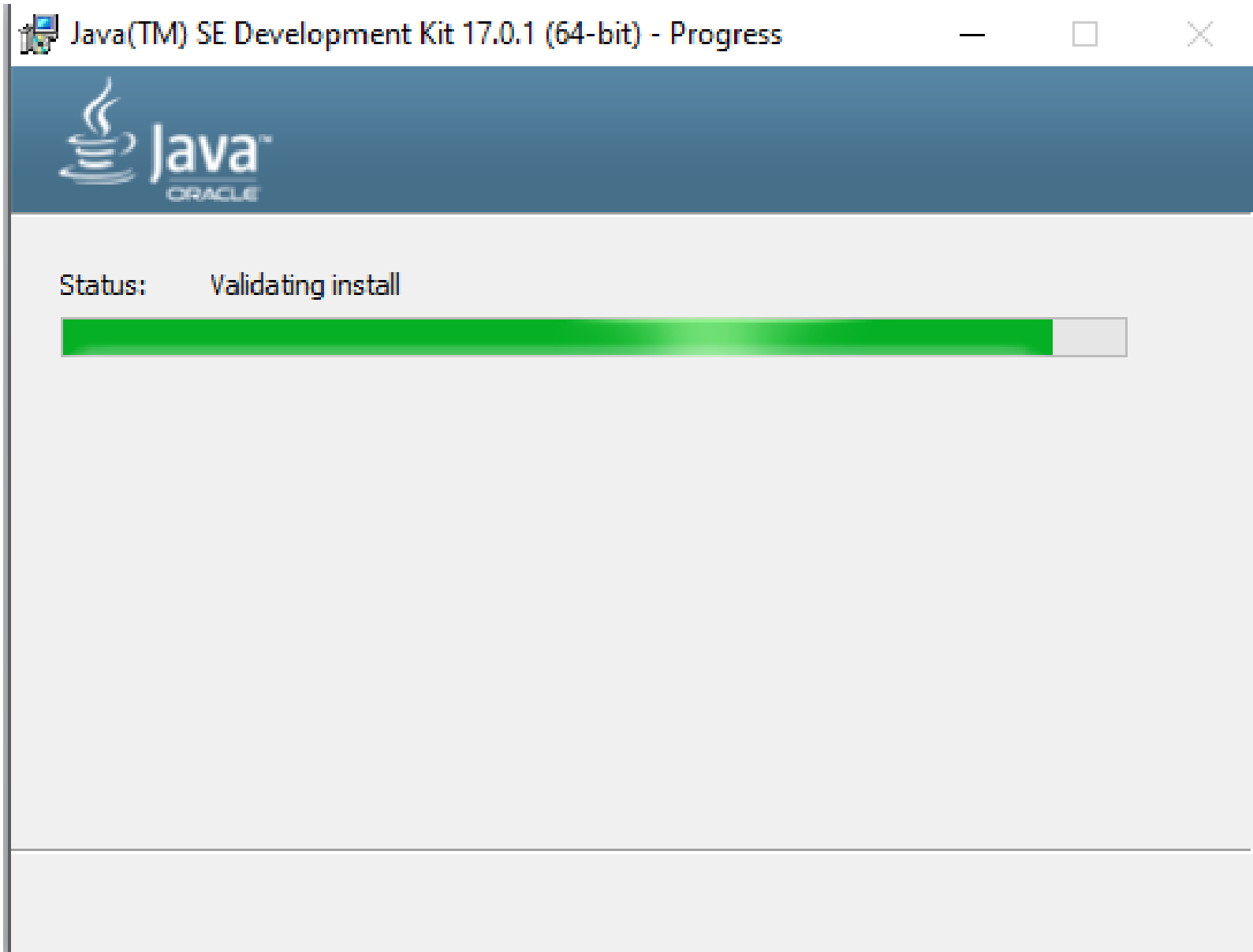
Una vez que tenemos el JDK (Java Development Kit) procedemos a instalarlo:



Presionamos el botón "next". Haremos la instalación por defecto por lo que presionamos el botón next nuevamente (dejaremos el directorio por defecto donde se almacenará Java):



Esperamos unos minutos mientras se procede a la instalación de Java:



Una vez finalizado el proceso de instalación debe aparecer un diálogo similar a este y presionamos el botón Close (por el momento no vamos a instalar "Tutorials, Api documentation etc.):



Con esto ya tenemos instalado en nuestro equipo el compilador de Java y la máquina virtual que nos permitirá ejecutar nuestros programas. En el próximo concepto instalaremos un editor de texto para poder codificar nuestros programas en Java.

2 - Instalación del editor Eclipse

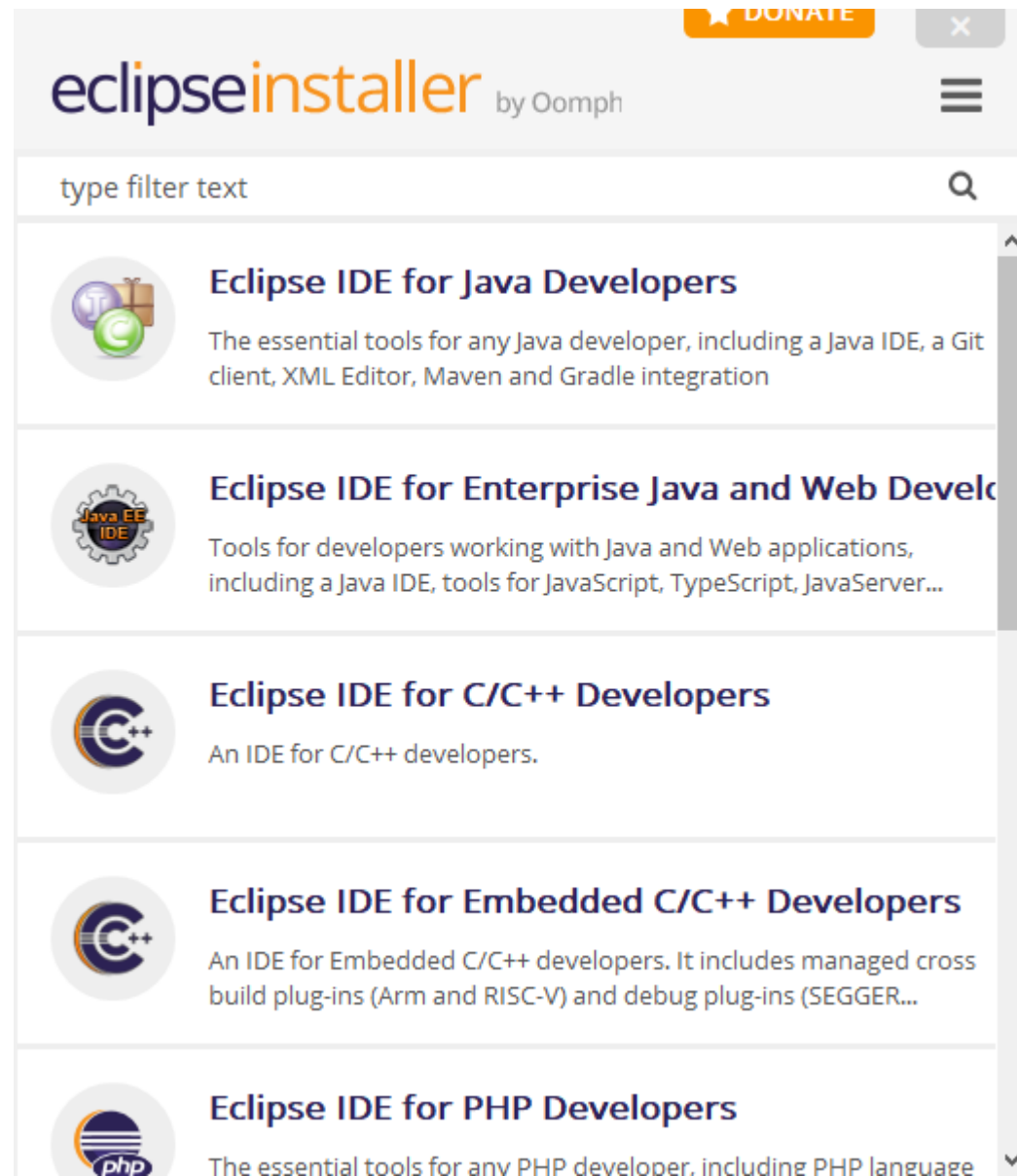
En el concepto anterior instalamos el lenguaje Java (con dichas herramientas podemos ejecutar y compilar programas codificados en java), pero ahora necesitamos instalar Eclipse que es un editor para codificar los programas (si bien podemos utilizar otros editores, nosotros utilizaremos este para seguir el curso)

Para la descarga del editor Eclipse lo hacemos del sitio:


[Eclipse](#)




Una vez que descargamos el instalador de Eclipse procedemos a ejecutarlo y debemos elegir el entorno "Eclipse IDE for Java Developers":





Seleccionamos la carpeta donde se instalará el Eclipse:

by Oomph

[★ DONATE](#)








Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

[details](#)


Java 11+ VM

C:\Program Files\Java\jdk-17.0.1




Installation Folder

C:\Users\emilio\eclipse\java-2021-12



☒ create start menu entry

☒ create desktop shortcut

 **INSTALL**



Eclipse IDE for Java Developers

[details](#)

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 11+ VM

C:\Program Files\Java\jdk-17.0.1



Installation Folder

C:\Users\emilio\eclipse\java-2021-12



create start menu entry



create desktop shortcut

▶ LAUNCH

show readme file

open in system explorer

keep installer

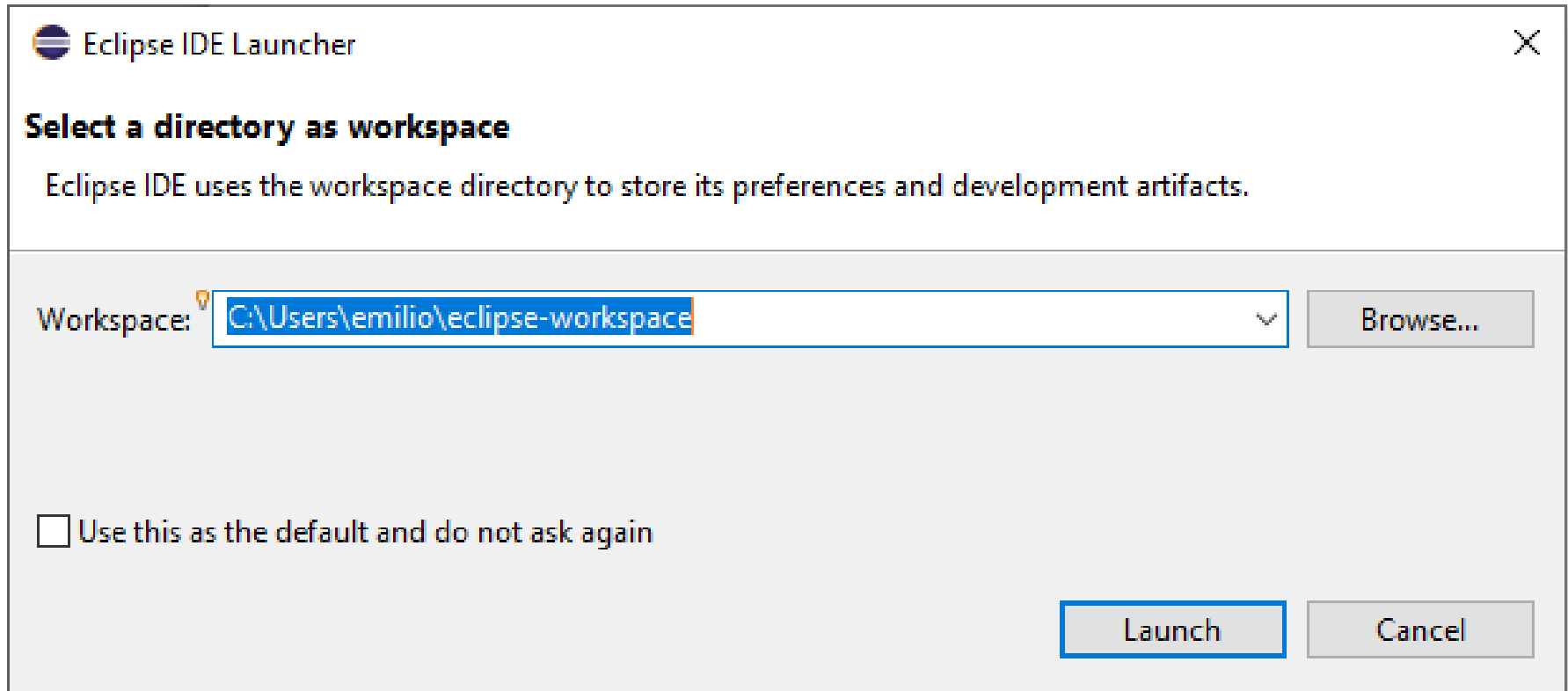
◀ BACK

Una vez instalado en forma completo el "Eclipse IDE for Java Developers" nos dirigimos a la carpeta donde se instalaron los archivos y procedemos a ejecutar el programa eclipse.exe (podemos ingresar desde el acceso directo que se crea en el escritorio)

Primero aparece un mensaje de inicio del Eclipse:



La primera vez que ejecutemos el editor Eclipse aparece un diálogo para seleccionar la carpeta donde se almacenarán los programas que desarrollaremos (podemos crear una carpeta donde almacenaremos todos los proyectos que desarrollaremos en el curso, si indicamos una carpeta que no existe el mismo Eclipse la crea):

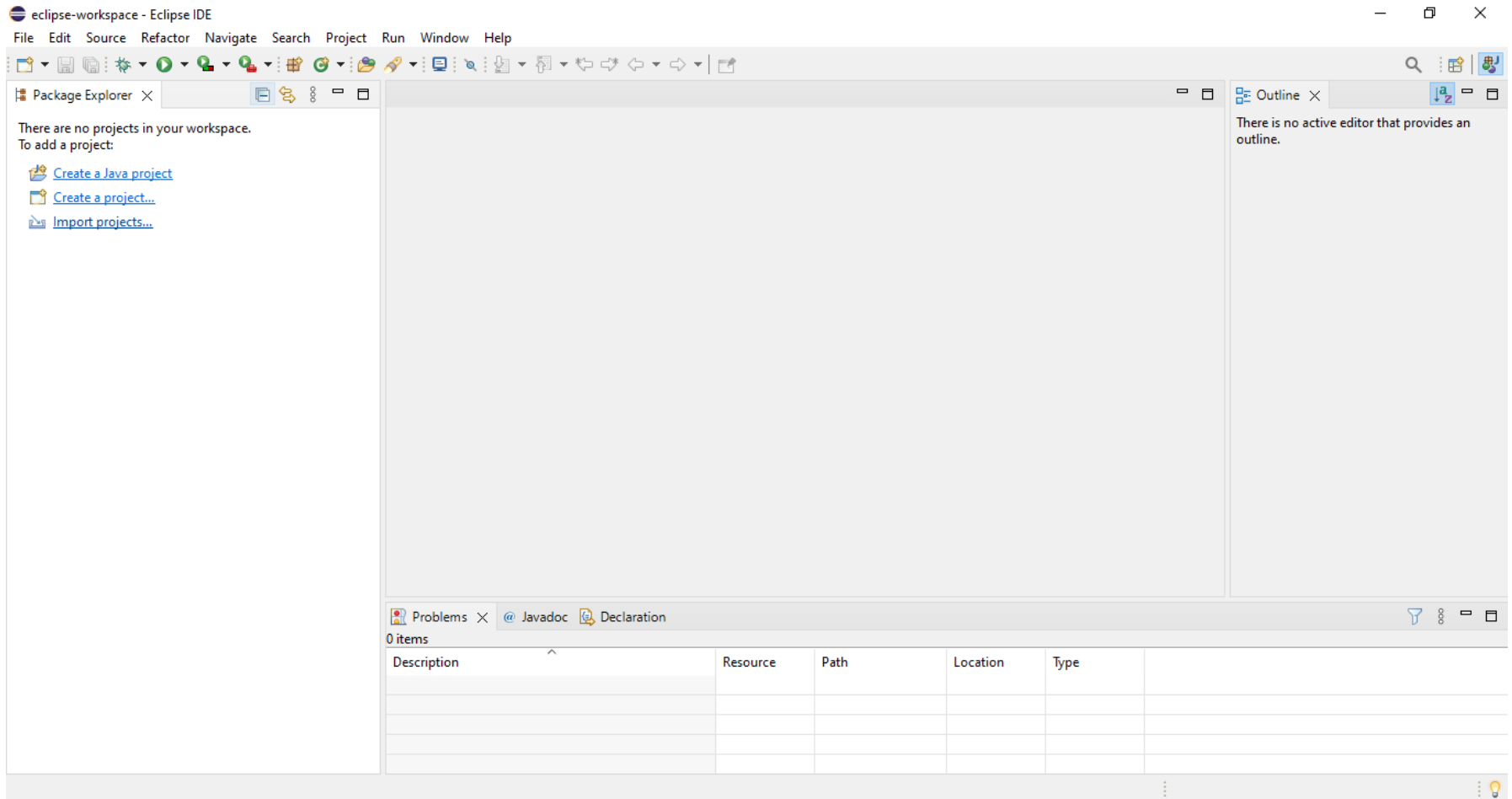


Una vez configurada la carpeta donde se creará el proyecto aparece el editor con una pantalla de presentación (Welcome):



Esta ventana de bienvenida la podemos cerrar seleccionando el ícono: "Workbench", con lo que aparece el entorno de trabajo del Eclipse (si queremos nuevamente ver la ventana de bienvenida podemos activarla desde el menú de opciones: Help -> Welcome)

El entorno de trabajo del Eclipse es:



3 - Pasos para crear un programa con Eclipse

Eclipse es un entorno de trabajo profesional, por lo que en un principio puede parecer complejo el desarrollo de nuestros primeros programas.

Todo programa en Eclipse requiere la creación de un "Proyecto", para esto debemos seleccionar desde el menú de opciones:

File Edit Source Refactor Navigate Search Project Run Window Help

New **Alt+Shift+N**

Open File...

Open Projects from File System...

Recent Files

Close Editor **Ctrl+W**

Close All Editors **Ctrl+Shift+W**

Save **Ctrl+S**

Save As...

Save All **Ctrl+Shift+S**

Revert

Move...

Rename... **F2**

Refresh **F5**

Convert Line Delimiters To

Print... **Ctrl+P**

Import...

Export...

Properties **Alt+Enter**

Switch Workspace

Restart

Exit

Java Project

Project...

Package

Class

Interface

Enum

Record

Annotation

Source Folder

Java Working Set

Folder

File

Untitled Text File

JUnit Test Case

Other... **Ctrl+N**

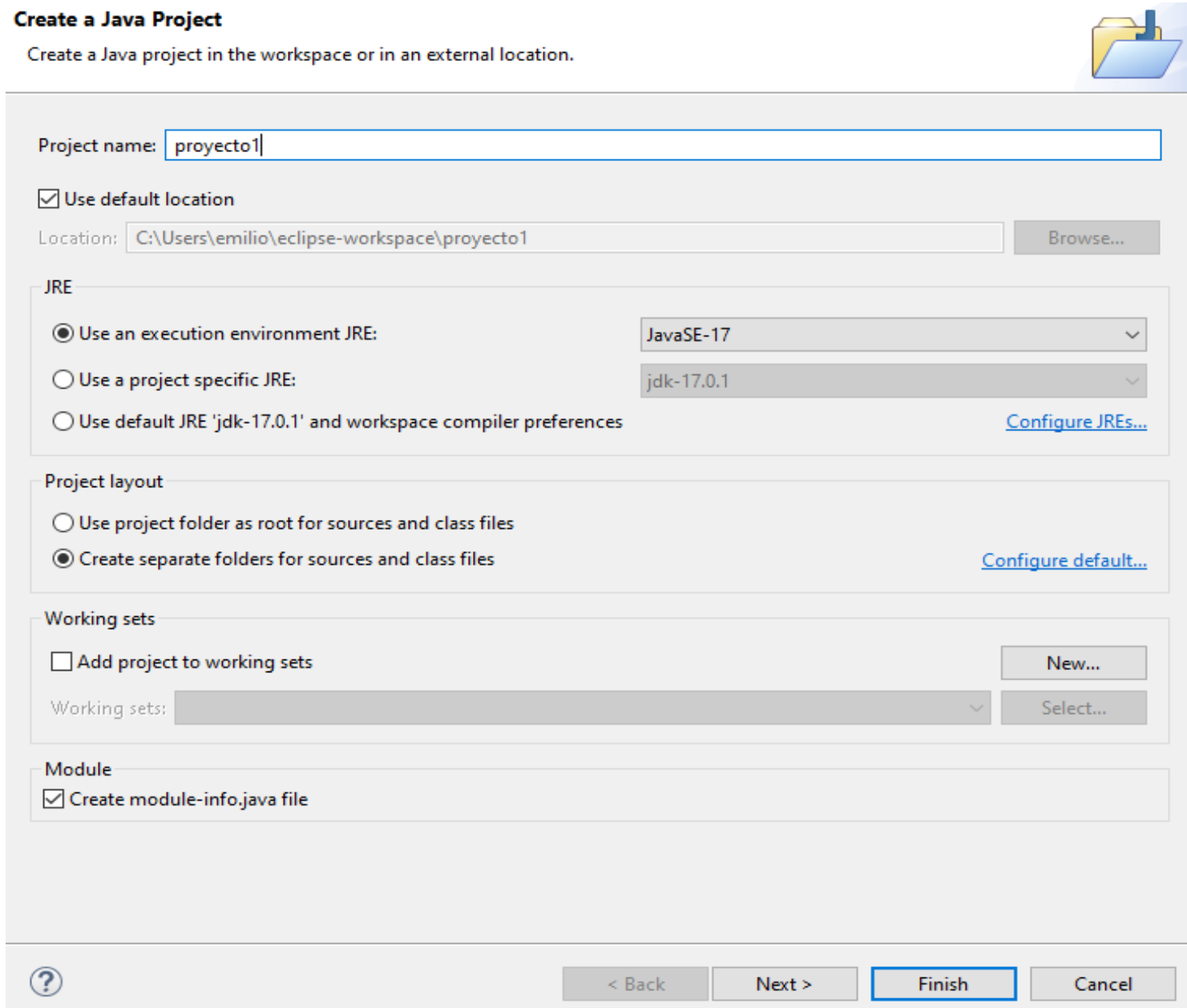
Create a Java project

Problems X Javadoc Declaration

0 items

| Description | Resource |
|-------------|----------|
| | |
| | |

Ahora aparece el diálogo donde debemos definir el nombre de nuestro proyecto:



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location: [Browse...](#)

JRE

☒ Use an execution environment JRE: [▼](#)

☐ Use a project specific JRE: [▼](#)

☐ Use default JRE 'jdk-17.0.1' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

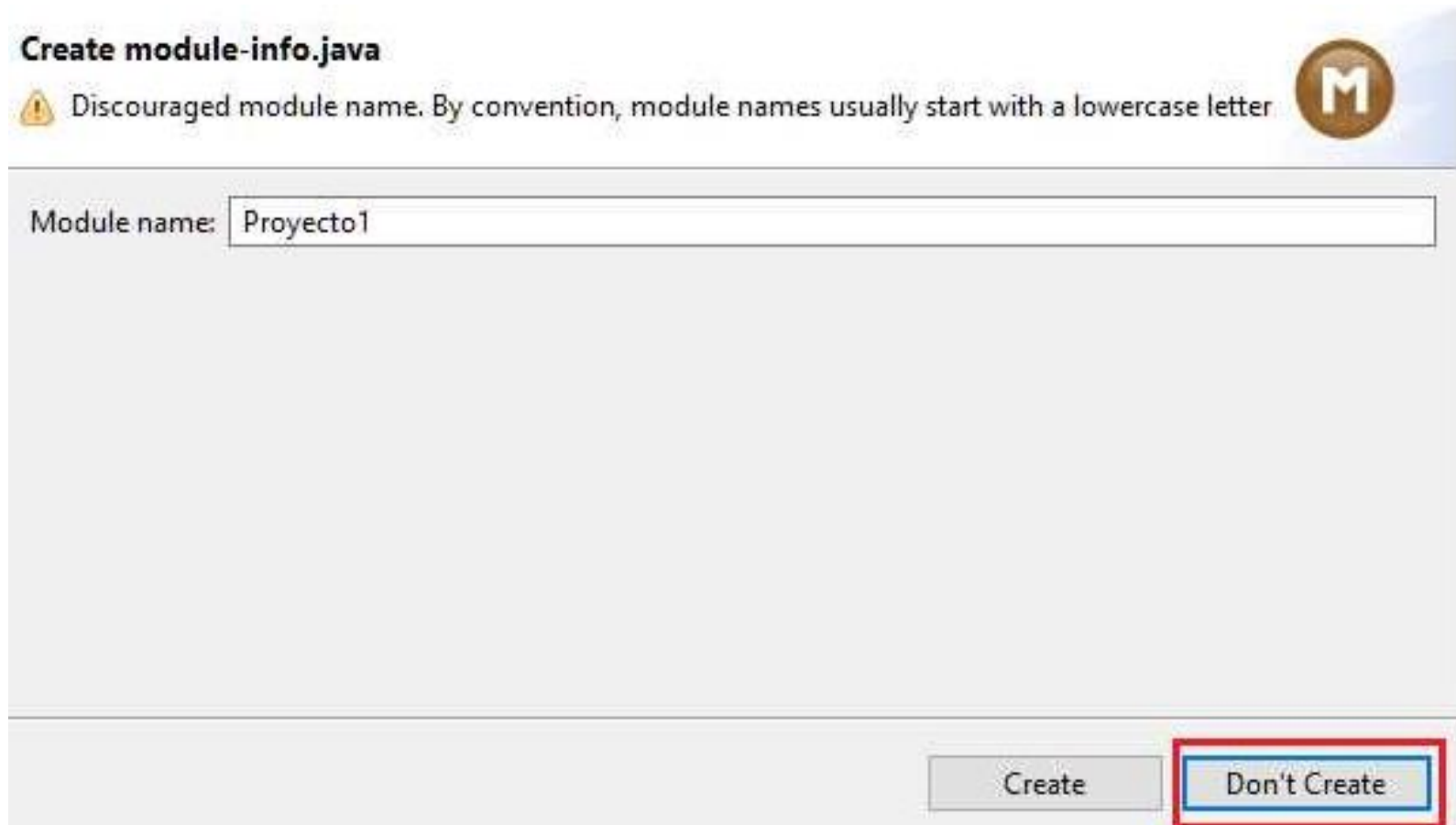
Module

☒ Create module-info.java file

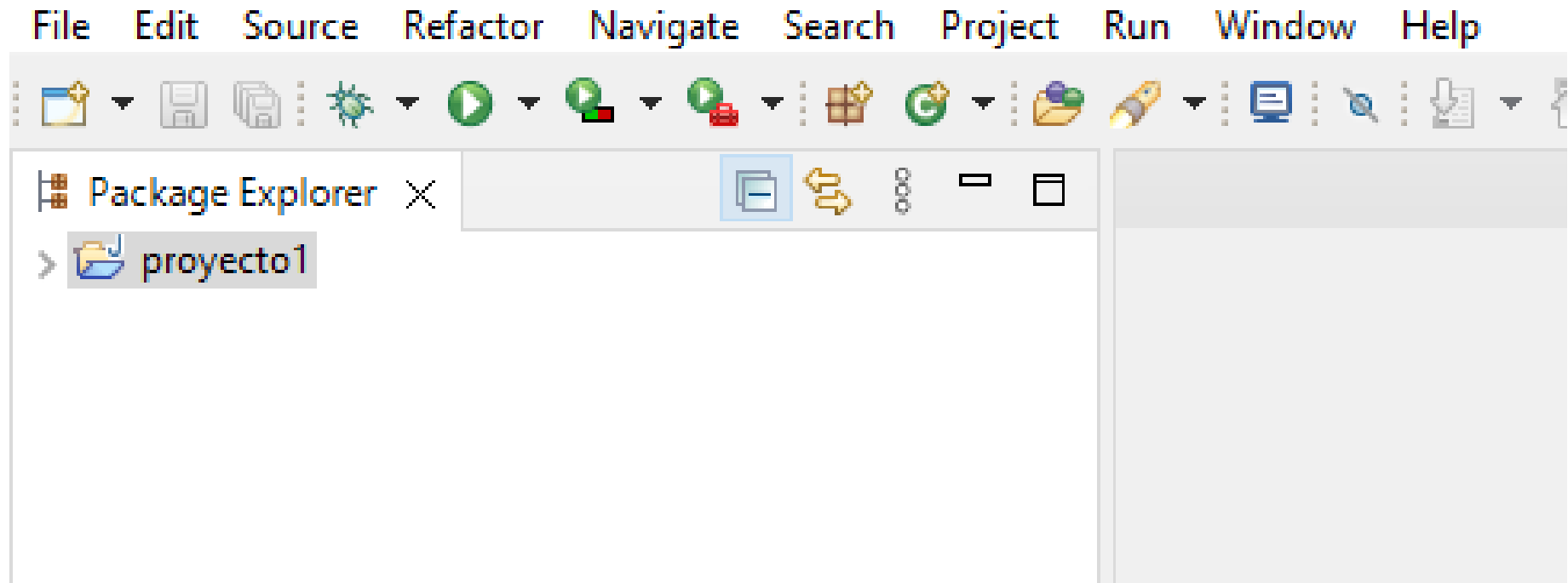
[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

En el campo de texto "Project Name" introducimos como nombre: Proyecto1 y dejamos todas las otras opciones del diálogo con los valores por defecto. Presionamos el botón "Finish".

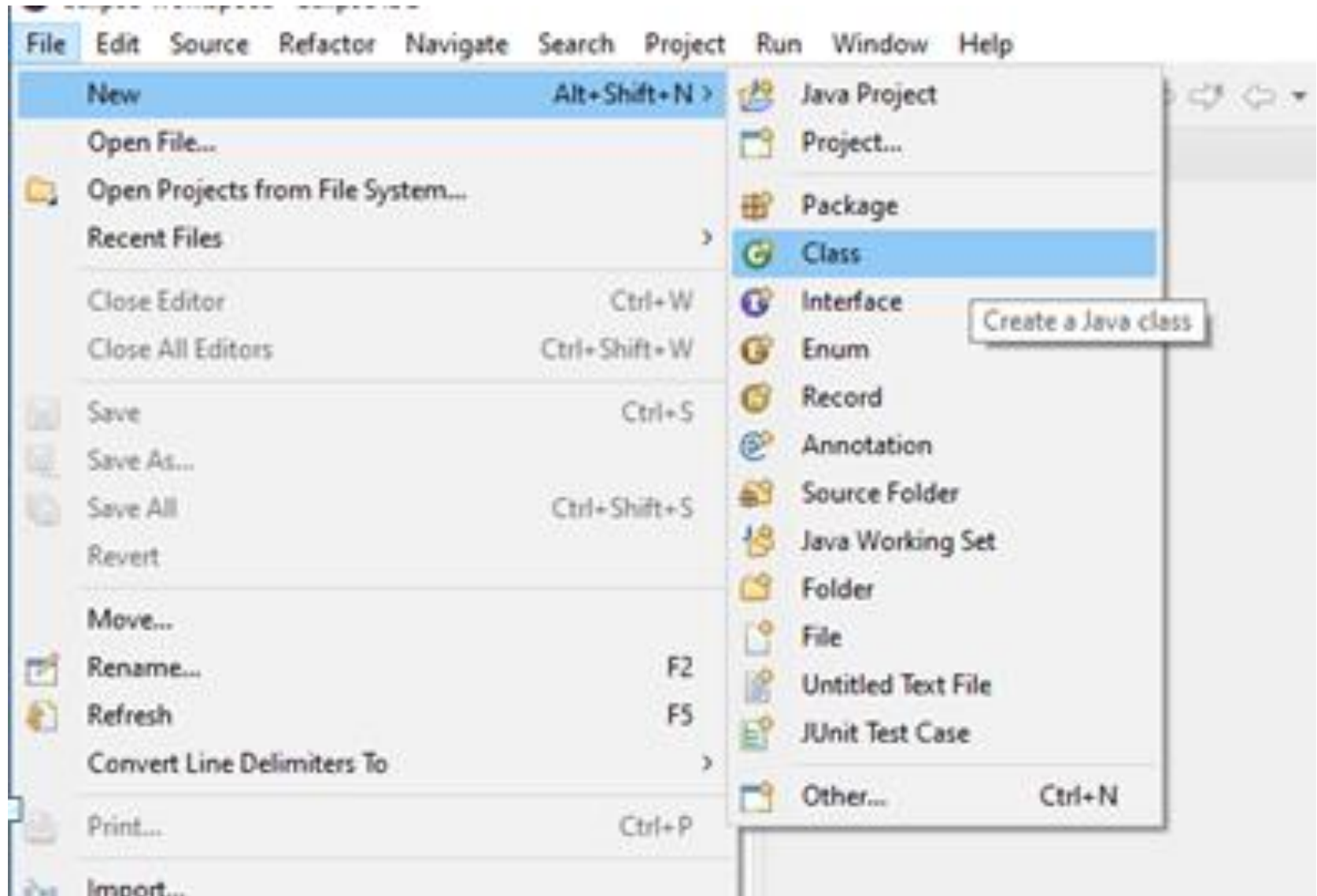
Aparece un nuevo diálogo que nos pide si queremos crear un módulo para nuestro proyecto, elegimos la opción para que no lo cree:



Ahora en la ventana de "Package Explorer" aparece el proyecto que acabamos de crear:



Como segundo paso veremos que todo programa en Java requiere como mínimo una clase. Para crear una clase debemos seleccionar desde el menú de opciones:



O desde la barra de íconos del Eclipse:



En el diálogo que aparece debemos definir el nombre de la clase (en nuestro primer ejemplo la llamaremos Clase1 (con mayúscula la letra C), luego veremos que es importante definir un nombre que represente al objetivo de la misma), los otros datos del diálogo los dejamos con los valores por defecto:

Source folder: Proyecto1/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: Clase1

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

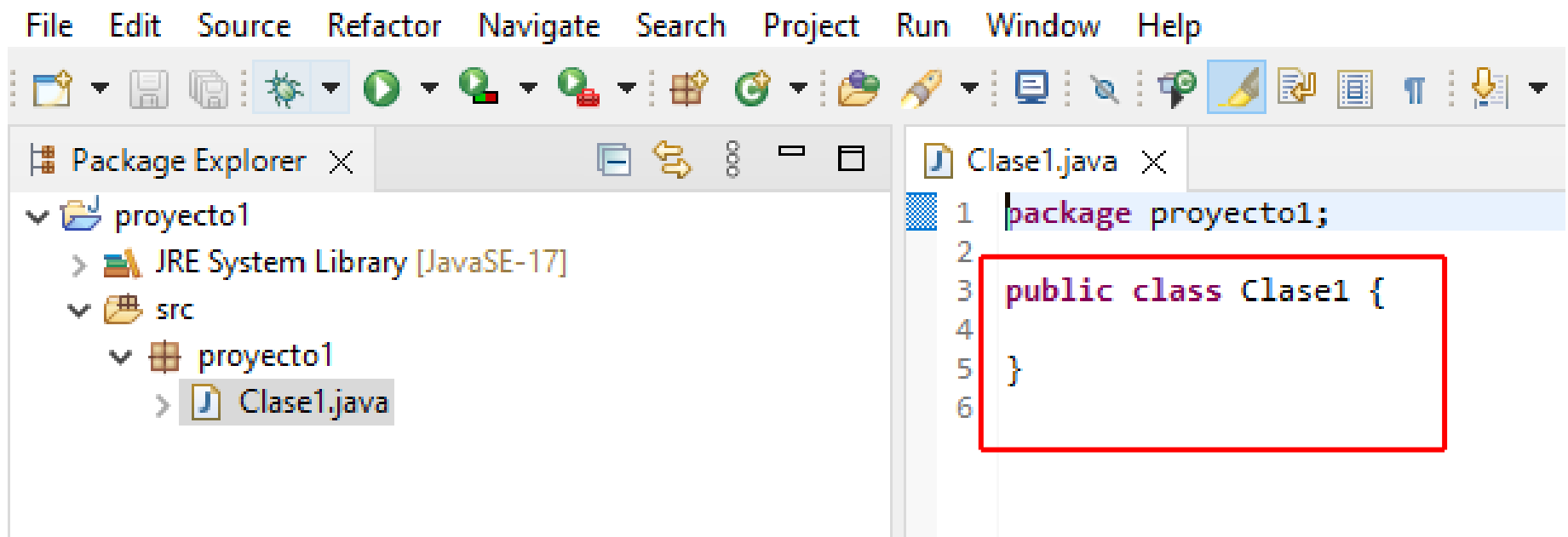
Interfaces: Add... Remove

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

Una vez presionado el botón "Finish" tenemos el archivo donde podemos codificar nuestro primer programa:



Más adelante veremos los archivos que se crean en un proyecto, ahora nos dedicaremos a codificar nuestro primer programa. En la ventana de edición ya tenemos el esqueleto de una clase de Java que el entorno Eclipse nos creó automáticamente.

```
public class Clase1 {  
  
}
```

Todo programa en Java debe definir la función main. Esta función la debemos codificar dentro de la clase: "Clase1".

Procedemos a escribir lo siguiente:

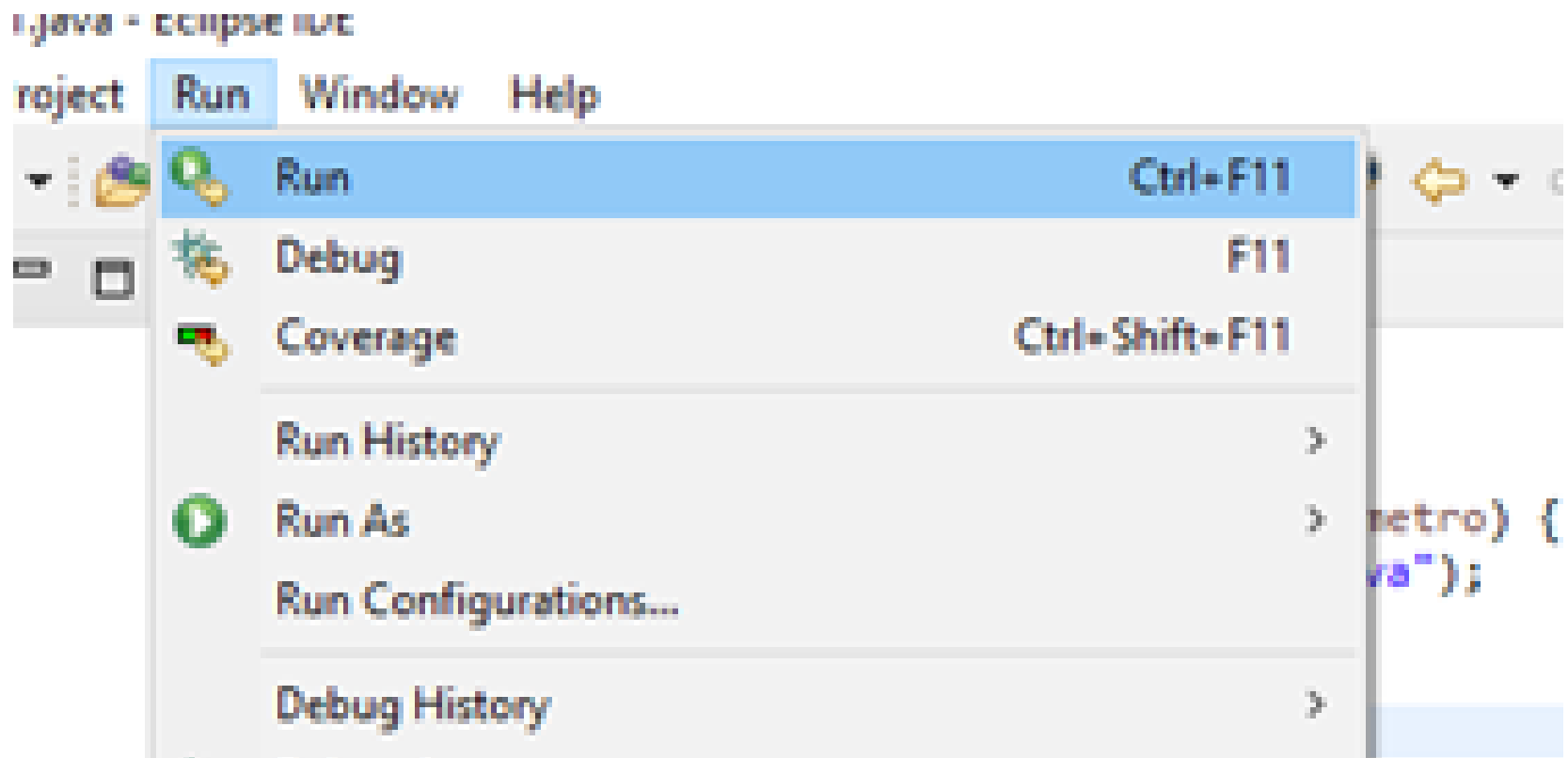
```
public class Clase1 {  
  
    public static void main(String[] parametro) {  
        System.out.println("Hola Mundo Java");  
    }  
  
}
```

Es decir tenemos codificado en el entorno del Eclipse nuestro primer programa:

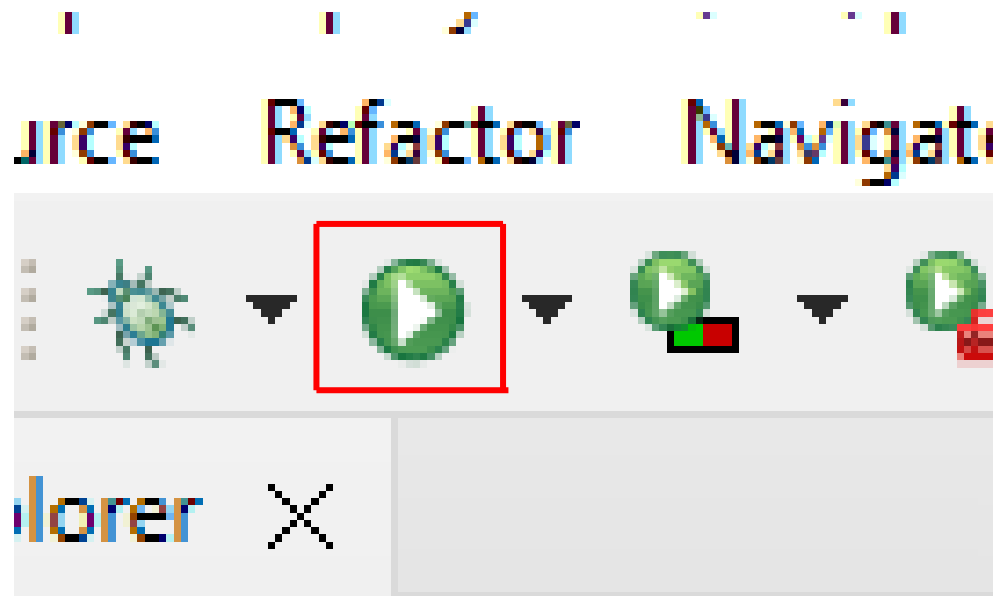
 *Clase1.java ×

```
1 package proyecto1;
2
3 public class Clase1 {
4     public static void main(String[] parametro) {
5         System.out.println("Hola Mundo Java");
6     }
7 }
8
```

Como último paso debemos compilar y ejecutar el programa, esto lo podemos hacer desde el menú de opciones:

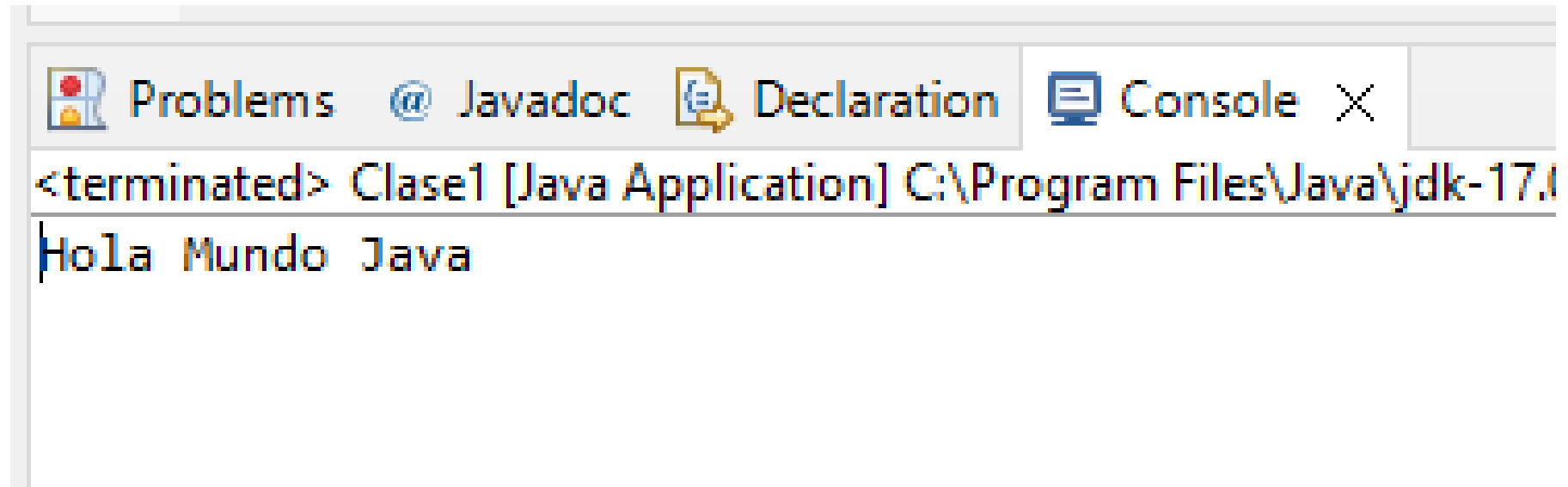


O desde la barra de íconos del Eclipse:



1

Si no hay errores de codificación debemos ver el resultado de la ejecución en una ventana de Eclipse llamada "Console" que aparece en la parte inferior (puede aparecer un diálogo pidiendo que grabemos el archivo, el cual confirmamos):



Lo más importante es que queden claro los pasos que debemos dar para crear un proyecto en Java. El objetivo de una clase, la función main etc. los veremos a lo largo de este curso.

4 - Objetivos del curso y nociones básicas indispensables

El curso está ideado para ser desarrollado por una persona que no conoce nada de programación y que utilice Java como primer lenguaje.

El objetivo fundamental de este curso es permitir que el estudiante pueda resolver problemas de distinta índole (matemáticos, administrativos, gráficos, contables etc.) empleando como herramienta el ordenador.

Hay que tener en cuenta que para llegar a ser programador se debe recorrer un largo camino donde cada tema es fundamental para conceptos futuros. Es importante no dejar temas sin entender y relacionar.

La programación a diferencia de otras materias como podría ser la historia requiere un estudio metódico y ordenado (en historia se puede estudiar la edad media sin tener grandes conocimientos de la edad antigua)

La programación es una actividad nueva para el estudiante, no hay en los estudios primarios y secundarios una materia parecida.

Es bueno tener paciencia cuando los problemas no se resuelven por completo, pero es de fundamental importancia dedicar tiempo al análisis individual de los problemas.

Qué es un programa?

Programa: Conjunto de instrucciones que entiende un ordenador para realizar una actividad.

Todo programa tiene un objetivo bien definido: un procesador de texto es un programa que permite cargar, modificar e imprimir textos, un programa de ajedrez permite jugar al ajedrez contra el ordenador u otro contrincante humano.

La actividad fundamental del programador es resolver problemas empleando el ordenador como herramienta fundamental.

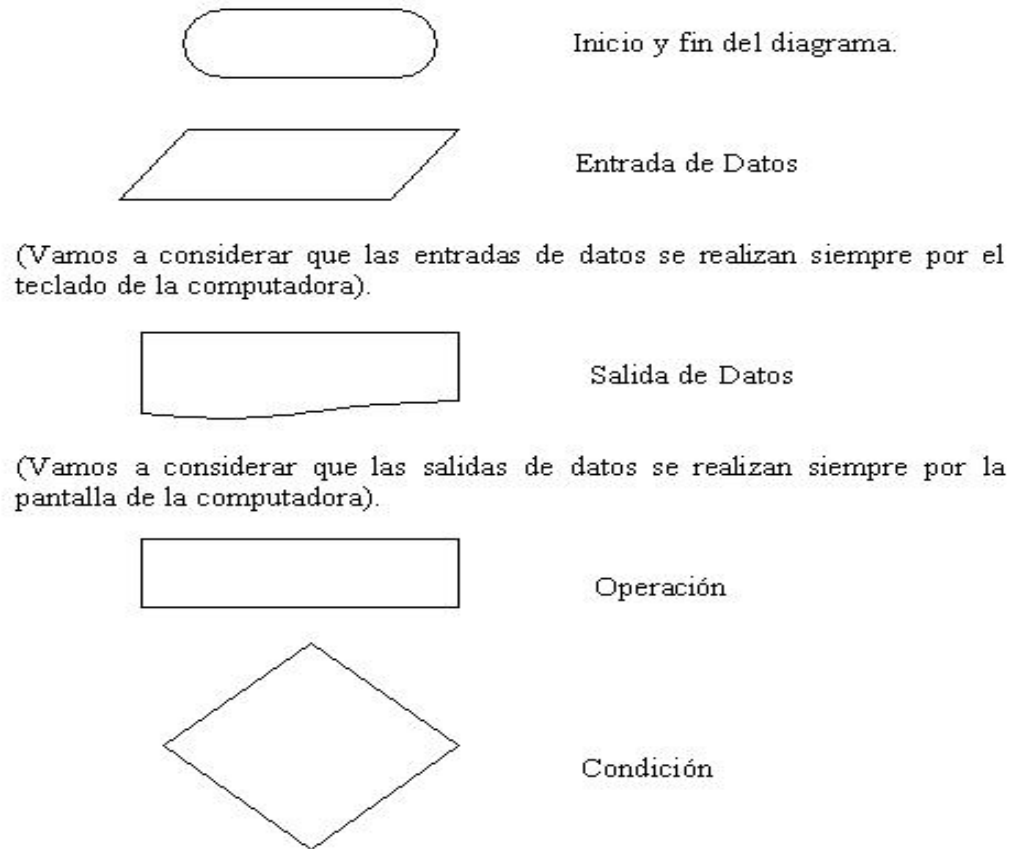
Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir para resolver un problema.

Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Los símbolos gráficos a utilizar para el planteo de diagramas de flujo son:



Estos son los elementos esenciales que intervienen en el desarrollo de un diagrama de flujo.

Planteamos un problema utilizando diagramas de flujo.

Para plantear un diagrama de flujo debemos tener muy en claro el problema a resolver.

Ejemplo : Calcular el sueldo mensual de un operario conociendo la cantidad de horas trabajadas y el pago por hora.

Podemos identificar:

Datos conocidos:

Horas trabajadas en el mes.

Pago por hora.

Proceso:

Cálculo del sueldo multiplicando la cantidad de horas por el pago por hora.

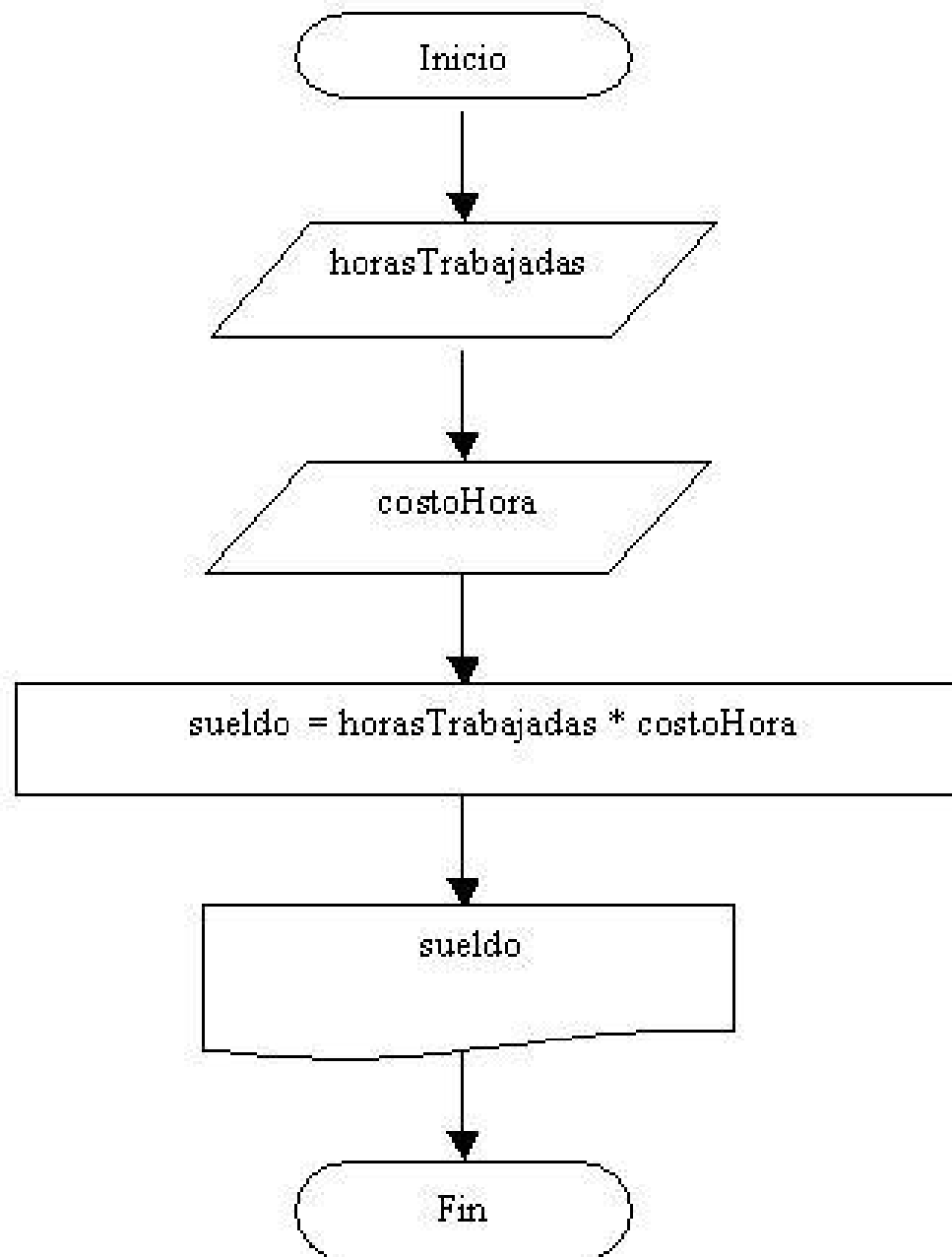
Información resultante:

Sueldo mensual.

Si hacemos un análisis todo problema está constituido por:

- **Datos conocidos:** Datos con los que se cuenta al plantear el problema.
- **Proceso:** Operaciones a realizar con los datos conocidos.
- **Información resultante:** Es la información que resuelve el problema.

Esta forma de expresar un problema identificando sus datos conocidos, procesos e información resultante puede llegar a ser engorrosa para problemas complejos donde hay muchos datos conocidos y procesos. Es por eso que resulta mucho más efectivo representar los pasos para la resolución del problema mediante un diagrama de flujo.



Resulta mucho más fácil entender un gráfico que un texto.

El diagrama de flujo nos identifica claramente los datos de entrada, operaciones y datos de salida.

En el ejemplo tenemos dos datos de entrada: horasTrabajadas y costoHora, a las entradas las representamos con un paralelogramo y hacemos un paralelogramo por cada dato de entrada.

La operación se representa con un rectángulo, debemos hacer un rectángulo por cada operación. A la salida la representamos con la hoja rota.

El diagrama de flujo nos da una idea del orden de ejecución de las actividades en el tiempo. Primero cargamos los datos de entrada, luego hacemos las operaciones necesarias y por último mostramos los resultados.

Codificación del problema con el lenguaje Java.

No debemos perder de vista que el fin último es realizar un programa de computación que permita automatizar una actividad para que muchos procesos sean desarrollados por la computadora.

El diagrama de flujo es un paso intermedio para poder ser interpretado por la computadora.

El paso siguiente es la codificación del diagrama de flujo en un lenguaje de computación, en nuestro caso emplearemos el lenguaje Java.

Lenguaje de computación: Conjunto de instrucciones que son interpretadas por una computadora para realizar operaciones, mostrar datos por pantalla, sacar listados por impresora, entrar datos por teclado, etc.

Conceptos básicos para codificar un programa.

Variable: Es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo.

Para el ejemplo planteado la variable horasTrabajadas almacena la cantidad de horas trabajadas por el operario. La variable valorHora almacena el precio de una hora de trabajo. La variable sueldo almacena el sueldo a abonar al operario.

En el ejemplo tenemos tres variables.

Tipos de variable:

Una variable puede almacenar:

- Valores Enteros (100, 260, etc.)
- Valores Reales (1.24, 2.90, 5.00, etc.)
- Cadenas de caracteres ("Juan", "Compras", "Listado", etc.)

Elección del nombre de una variable:

Debemos elegir nombres de variables representativas. En el ejemplo el nombre horasTrabajadas es lo suficientemente claro para darnos una idea acabada sobre su contenido. Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo hTr. Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos las horas trabajadas por el operario pero cuando pase el tiempo y leamos el diagrama probablemente no recordemos ni entendamos qué significa hTr.

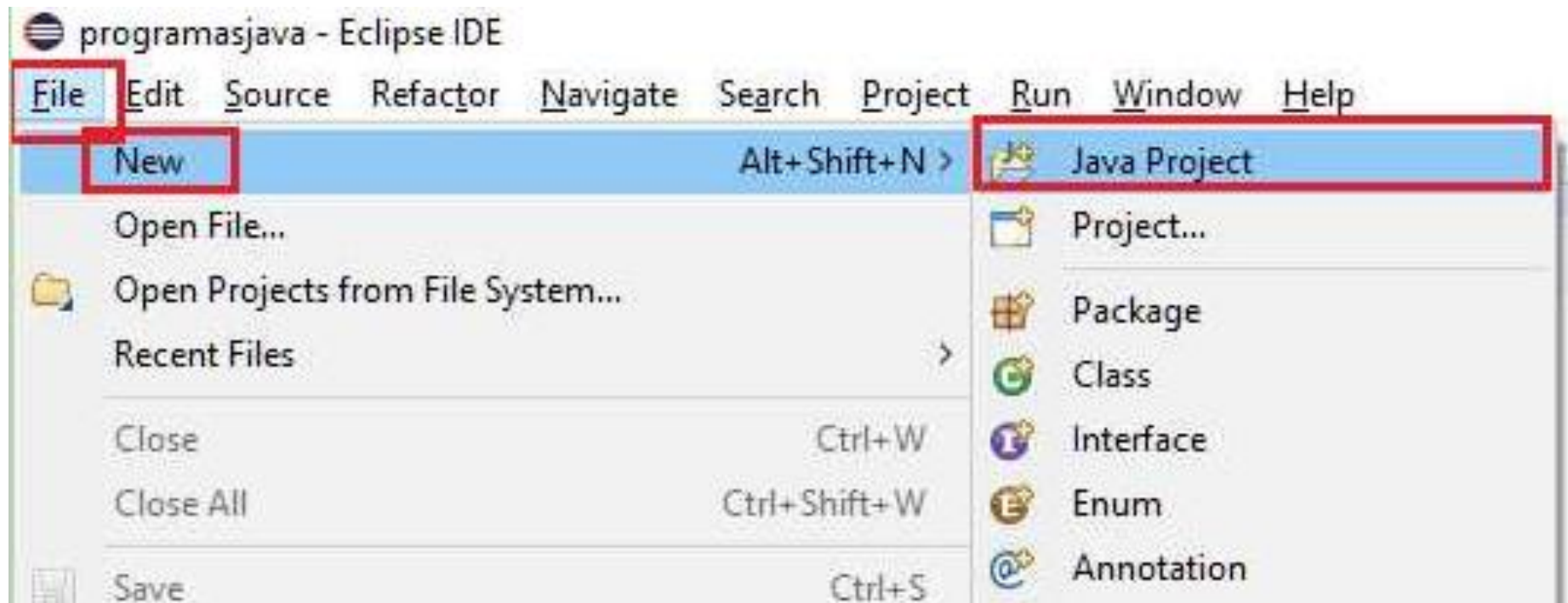
Consideraciones a tener en cuenta en cada proyecto.

Hay que tener en cuenta que el entorno de programación "Eclipse" no ha sido desarrollado pensando en un principiante de la programación. Lo mismo ocurre con el propio lenguaje Java, es decir su origen no tiene como principio el aprendizaje de la programación. Debido a estos dos puntos veremos que a medida que avanzamos **a lo largo del curso muchos conceptos que iremos dejando pendientes se irán aclarando.**

Codificaremos el problema propuesto para repasar los pasos para la creación de un proyecto en Eclipse, creación de la clase principal, definición de la función main y el posterior desarrollo del algoritmo del problema.

Pasos.

1 - Creación del proyecto (tema visto anteriormente). Podemos asignarle como nombre: SueldoOperario (normalmente uno busca un nombre representativo al programa que desarrolla)



Create a Java Project

Create a Java project in the workspace or in an external location.



Project name:

☒ Use default location

Location:

[Browse...](#)

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE 'jdk-17.0.1' and workspace compiler preferences

[Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files

[Configure default...](#)

Working sets

☐ Add project to working sets

[New...](#)

Working sets:

[Select...](#)

Module

☒ Create module-info.java file



< Back

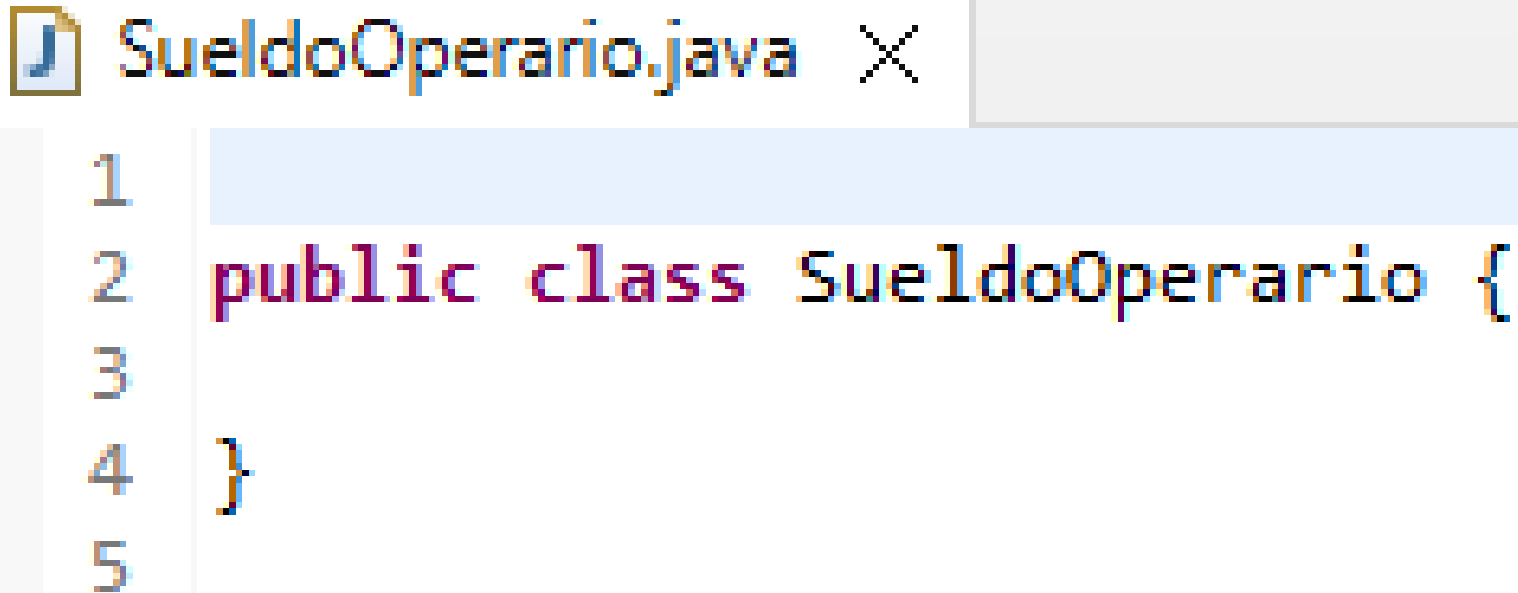
Next >

Finish

Cancel

2 - Creación de la clase. Definiremos como nombre el mismo que le asignamos al proyecto (esto no es obligatorio como veremos más adelante, un proyecto puede contener varias clases). Es decir disponemos como nombre de la clase: SueldoOperario.

Inicializamos el campo que solicita el "Name" con "SueldoOperario".



The screenshot shows a code editor window titled "SueldoOperario.java" with a close button (X). The editor contains the following Java code:

```
1  
2 public class SueldoOperario {  
3  
4 }  
5
```

3 - Codificamos el algoritmo en la clase: SueldoOperario.

```
import java.util.Scanner;

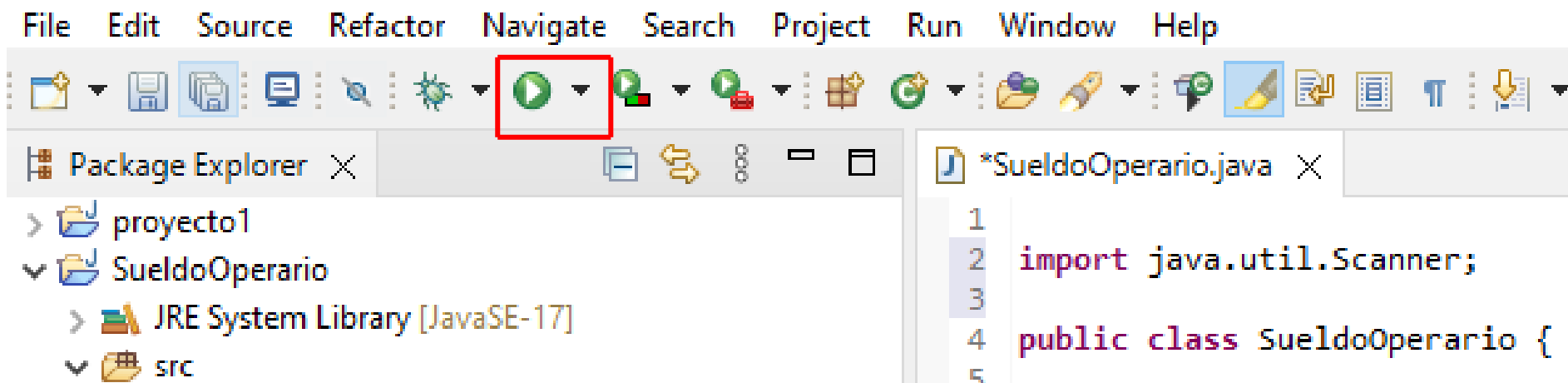
public class SueldoOperario {

    public static void main(String[] ar) {
        Scanner teclado = new Scanner(System.in);
        int horasTrabajadas;
        float costoHora;
        float sueldo;
        System.out.print("Introduce la cantidad de horas trabajadas por el empleado:");
        horasTrabajadas=teclado.nextInt();
        System.out.print("Introduce el valor de la hora:");
        costoHora = teclado.nextFloat();
        sueldo = horasTrabajadas * costoHora;
        System.out.print("El empleado debe cobrar:");
        System.out.print(sueldo);
    }
}
```

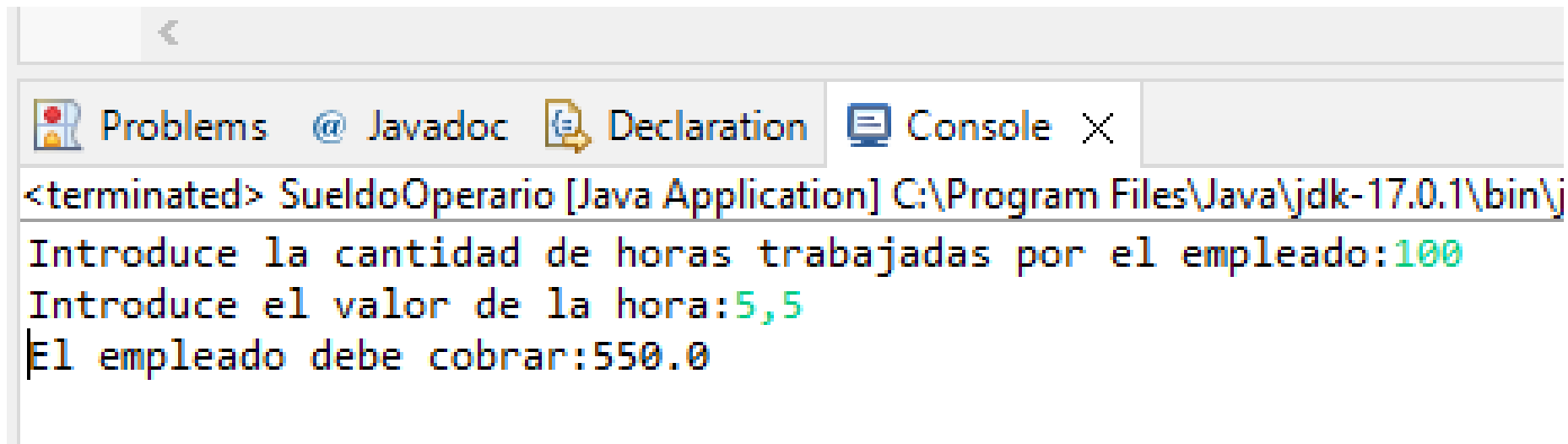
*SueldoOperario.java X

```
1
2 import java.util.Scanner;
3
4 public class SueldoOperario {
5
6     public static void main(String[] ar) {
7         Scanner teclado=new Scanner(System.in);
8         int horasTrabajadas;
9         float costoHora;
10        float sueldo;
11        System.out.print("Introduce la cantidad de horas trabajadas por el empleado:");
12        horasTrabajadas=teclado.nextInt();
13        System.out.print("Introduce el valor de la hora:");
14        costoHora=teclado.nextFloat();
15        sueldo=horasTrabajadas * costoHora;
16        System.out.print("El empleado debe cobrar:");
17        System.out.print(sueldo);
18    }
19 }
20
21
```


4 - Ejecutamos el programa:



5 - Si no hay errores sintácticos procedemos a activar la ventana de la "Console" con el mouse y cargamos por teclado los dos datos que se solicitan (la cantidad de horas trabajadas y el precio de la hora):



The screenshot shows the 'Console' tab of a Java IDE. The title bar of the console window reads: `<terminated> SueldoOperario [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\j`. The console output consists of three lines: `Introduce la cantidad de horas trabajadas por el empleado:100`, `Introduce el valor de la hora:5,5`, and `El empleado debe cobrar:550.0`. The input values '100' and '5,5' are highlighted in green, and the final result '550.0' is also highlighted in green.

```
<terminated> SueldoOperario [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\j
Introduce la cantidad de horas trabajadas por el empleado:100
Introduce el valor de la hora:5,5
El empleado debe cobrar:550.0
```

Estos cinco pasos fundamentales son los que debemos llevar a cabo cada vez que desarrollemos un nuevo programa en Java.

Explicación.

Ahora veremos una explicación de varias partes de nuestro programa y otras partes quedarán pendientes para más adelante ya que en este momento difícilmente se entenderán.

Conceptos que quedarán pendientes para explicar:

1. Concepto de una clase. Veremos más adelante que en Java todo debe estar contenido en clases, por lo que hasta el problema más elemental debe estar contenido en una clase. Para declarar una clase utilizamos la sintaxis:

```
public class SueldoOperario {  
  
}
```

El nombre de la clase no puede tener espacios en blanco, comienza con una letra mayúscula y en caso de estar constituida por dos o más palabras el primer carácter va en mayúsculas, no puede empezar con un número, pero si puede llevar números a partir del segundo carácter. Toda clase debe tener una llave de apertura y una llave de cierre.

2. Todo programa constituido por una única clase debe tener definida la función main:

```
public static void main(String[] ar) {  
  
}
```

La función main es la primera que se ejecuta y debe llevar la sintaxis indicada anteriormente (más adelante veremos que significa el parámetro ar, las palabras claves public, static y void). La función main tiene una llave de apertura y una llave de cierre (similar a la clase). **La función main debe estar contenida en la clase.**

3. Cuando se **requieren utilizar otras clases** debemos **importarlas** previo a la declaración de la clase, en nuestro problema utilizamos **la clase Scanner que se encuentra en el paquete java.util** por lo que la importamos con la siguiente sintaxis:

```
import java.util.Scanner;
```

En la **main** creamos un **objeto de la clase Scanner** que nos permitirá **introducir por teclado los valores**:

```
Scanner teclado=new Scanner(System.in);
```

Conceptos que deben quedar claros:

1. Por el momento haremos todo el algoritmo dentro de la función main. Es decir el resto siempre será lo mismo (declarar un proyecto, declarar una clase, definir una función main)

2. Si observamos el diagrama de flujos vemos que debemos definir tres variables: (horasTrabajadas, costoHora,sueldo), aquí es donde debemos definir que tipos de datos se almacenarán en las mismas. La cantidad de horas normalmente será un valor entero (ej. 100 - 150 - 230 etc.), pero el coste de la hora es muy común que sea un valor real (ej. 5,35 - 7,50 etc.) y como el sueldo resulta de multiplicar las horas trabajadas por el costo por hora el mismo deberá ser real.

La definición de las variables las hacemos en la main:

```
int horasTrabajadas;  
float costoHora;  
float sueldo;
```

Utilizamos la palabra clave `int` para definir variables enteras (en Java las palabras claves deben ir obligatoriamente en minúsculas, sino se produce un error sintáctico) Luego de la palabra clave debemos indicar el nombre de la variable, por ejemplo: `horasTrabajadas` (se propone que el nombre de la variable comience con minúsculas y en caso de estar constituida por dos palabras o más a partir de la segunda palabra el primer caracter se especifique con mayúsculas (un **nombre de variable no puede tener espacios en blanco**, empezar con un número, ni tampoco utilizar caracteres especiales).

Debemos buscar siempre nombres de variables que nos indiquen que almacenan (no es conveniente llamar a nombres de variables con letras individuales).

3. Para mostrar mensajes en la "Console" utilizamos la siguiente sintaxis:

```
System.out.print("Introduce la cantidad de horas trabajadas por el  
empleado:");
```

Con esta sintaxis todo lo que se encuentra contenido entre comillas aparecerá exactamente en la ventana de la "Console".

Si disponemos una variable:

```
System.out.print(sueldo);
```

Aparecerá el contenido de la variable. Es decir el valor almacenado en la variable sueldo y no el mensaje "sueldo".

4. Para hacer la entrada de datos por teclado en Java se complica. Utilizaremos una clase llamada Scanner que nos facilita la introducción de datos. Por eso tuvimos que importar la clase Scanner que se encuentra en el paquete java.util en la primer línea de nuestro programa.

En la función main debemos crear un objeto de la clase Scanner con la siguiente sintaxis:

```
Scanner teclado=new Scanner(System.in);
```

Luego para cargar valores enteros por teclado debemos implementar la siguiente sintaxis:

```
horasTrabajadas=teclado.nextInt();
```

Pero si el dato a cargar se trata de un valor float, debemos utilizar la siguiente sintaxis:

```
costoHora=teclado.nextFloat();
```

5. Las operaciones que indicamos en el diagrama de flujo mediante la figura rectángulo las codificamos tal cual:

$\text{suelo} = \text{horasTrabajadas} * \text{costoHora};$

Podemos ver una relación entre las instrucciones que debemos utilizar para cada símbolo del diagrama de flujo:

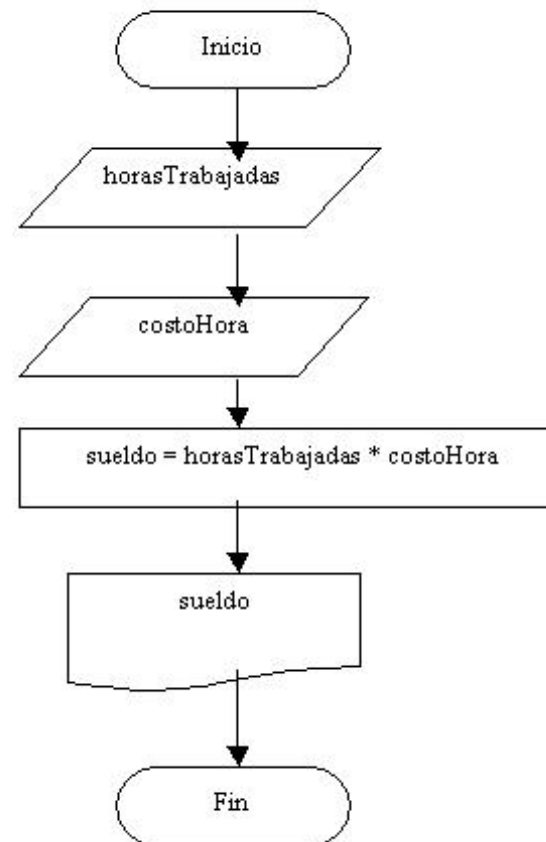
main

`horasTrabajadas=teclado.nextInt();`
(Carga por teclado)

`costoHora=teclado.nextFloat();`
(Carga por teclado)

`suelo=horasTrabajadas*costoHora;`
(Operación)

`System.out.print(suelo);`
(Salida por pantalla)



En el diagrama de flujo no indicamos la definición de variables:

```
int horasTrabajadas;  
float costoHora;  
float sueldo;
```

No indicamos la creación del objeto de la clase Scanner:

```
Scanner teclado=new Scanner(System.in);
```

No representamos con símbolos los mensajes a mostrar previo a la carga de datos por teclado:

```
System.out.print("Introduce la cantidad de horas trabajadas por el  
empleado:");
```

Como hemos visto hasta ahora hay muchas partes de nuestro código que no entendemos pero son indispensables para la implementación de nuestros programas, a medida que avancemos con el curso muchos de estos conceptos se irán aclarando.

5 - Errores sintácticos y lógicos

Confeccionaremos un problema y agregaremos adrede una serie de errores tipográficos. Este tipo de errores siempre son detectados por el **COMPILADOR**, antes de ejecutar el programa.

A los errores tipográficos, como por ejemplo la falta de puntos y comas, nombres de variables incorrectas, falta de paréntesis, palabras claves mal escritas, etc. los llamamos errores **SINTACTICOS**.

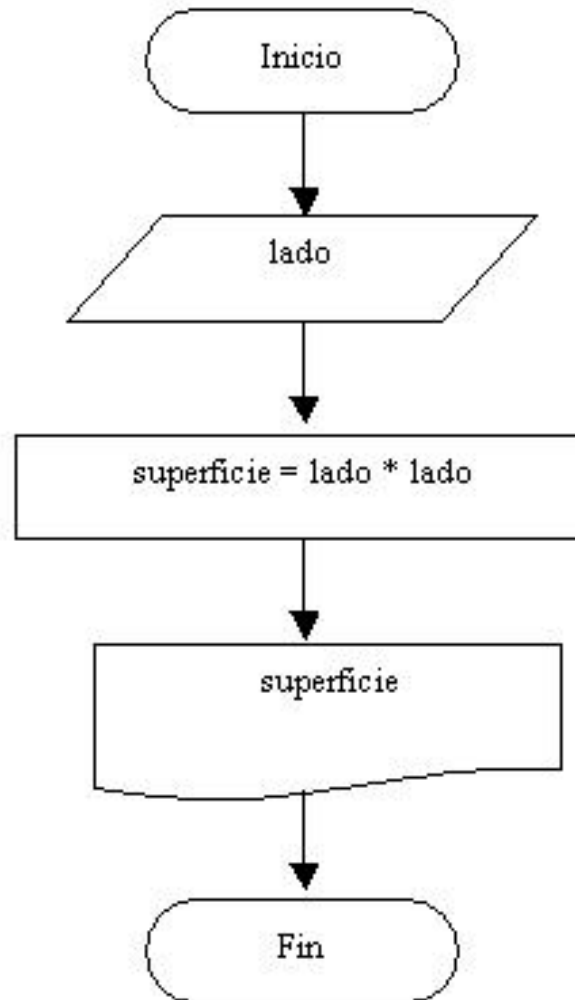
Un programa no se puede ejecutar sin corregir absolutamente todos los errores sintácticos.

Existe otro tipo de errores llamados **ERRORES LOGICOS**. Este tipo de errores en programas grandes (miles de líneas) son más difíciles de localizar. Por ejemplo un programa que permite hacer la facturación pero la salida de datos por impresora es incorrecta.

Problema:

Hallar la superficie de un cuadrado conociendo el valor de un lado.

Diagrama de flujo:



Proyecto:

Creemos un proyecto llamado SuperficieCuadrado y una clase llamada SuperficieCuadrado.

Codificamos el algoritmo en Java e introducimos dos errores sintáctico:

- 1 - Disponemos el nombre del objeto System con minúsculas.
- 2 - Tratamos de imprimir el nombre de la variable superficie con el primer caracter en mayúsculas.

```
9      lado=teclado.nextInt();
10     superficie=lado * lado;
11     system.out.print("La superficie del cuadrado es:");
12     System.out.print(Superficie);
13 }
14 }
15
```

Como podemos observar aparece subrayado la línea donde disponemos System con minúsculas como en la línea que imprimimos la variable superficie con mayúsculas. Si modificamos y corregimos los dos errores sintácticos podremos ejecutar nuestro programa.

Programa correctamente codificado:

```
import java.util.Scanner;

public class SuperficieCuadrado {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int lado;
        int superficie;
        System.out.print("Introduce el valor del lado del cuadrado:");
        lado=teclado.nextInt();
        superficie=lado * lado;
        System.out.print("La superficie del cuadrado es:");
        System.out.print(superficie);
    }
}
```

SuperficieCuadrado.java X

```
1 import java.util.Scanner;
2
3 public class SuperficieCuadrado {
4     public static void main(String[] ar) {
5         Scanner teclado=new Scanner(System.in);
6         int lado;
7         int superficie;
8         System.out.print("Introduce el valor del lado del cuadrado:");
9         lado=teclado.nextInt();
10        superficie=lado * lado;
11        System.out.print("La superficie del cuadrado es:");
12        System.out.print(superficie);
13    }
14 }
15
16
```

< Problems @ Javadoc Declaration Console X

<terminated> SuperficieCuadrado [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe
Introduce el valor del lado del cuadrado:15
La superficie del cuadrado es:225

Programa con un error lógico:

```
import java.util.Scanner;

public class SuperficieCuadrado {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int lado;
        int superficie;
        System.out.print("Introduce el valor del lado del cuadrado:");
        lado=teclado.nextInt();
        superficie = lado * lado * lado;
        System.out.print("La superficie del cuadrado es:");
        System.out.print(superficie);
    }
}
```

Como podemos observar si ejecutamos el programa no presenta ningún error de compilación. Pero al introducir el valor del lado del cuadrado (por ejemplo el valor 10) obtenemos como resultado un valor incorrecto (imprime el 1000), esto debido que definimos incorrectamente la fórmula para calcular la superficie del cuadrado:

```
superficie = lado * lado * lado;
```

6 - Estructura de programación secuencial

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.

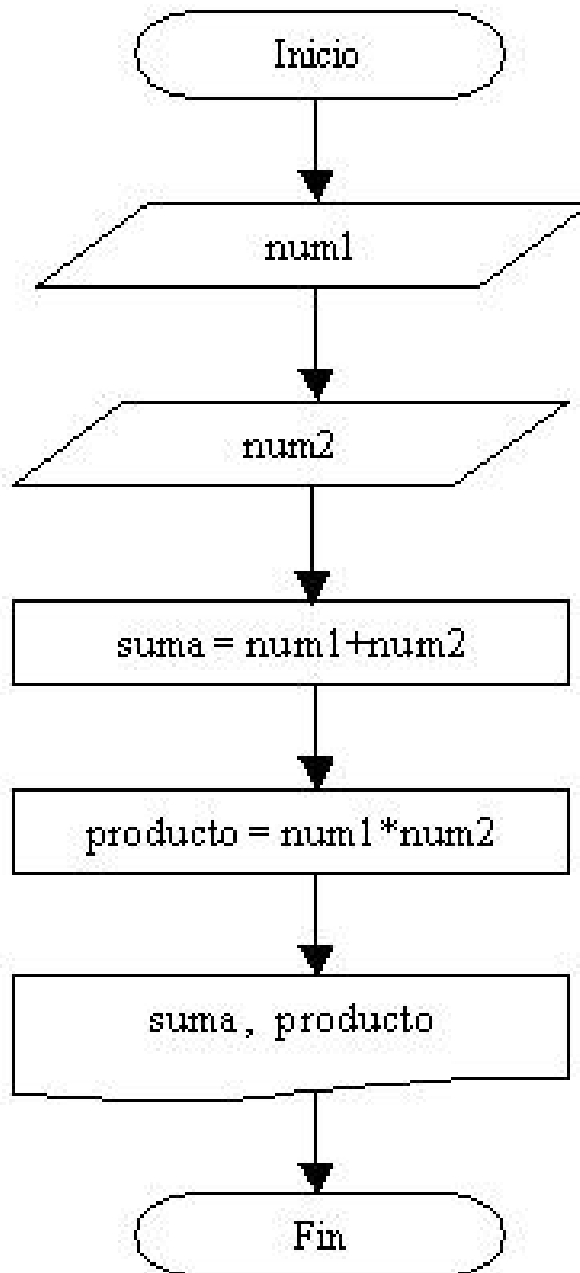
Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.

Problema:

Realizar la carga de dos números enteros por teclado e imprimir su suma y su producto.

Diagrama de flujo:



Tenemos dos entradas num1 y num2 (recordar cuáles son los nombres de variables correctas), dos operaciones: realización de la suma y del producto de los valores introducidos y dos salidas, que son los resultados de la suma y el producto de los valores introducidos. En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

Programa:

```
SumaProductoNumeros.java X
1
2 import java.util.Scanner;
3
4 public class SumaProductoNumeros {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int num1,num2,suma,producto;
8         System.out.print("Introduce primer valor:");
9         num1=teclado.nextInt();
10        System.out.print("Introduce segundo valor");
11        num2=teclado.nextInt();
12        suma=num1 + num2;
13        producto=num1 * num2;
14        System.out.print("La suma de los dos valores es:");
15        System.out.println(suma);
16        System.out.print("El producto de los dos valores es:");
17        System.out.println(producto);
18    }
19 }
```

< Problems @ Javadoc Declaration Console X

<terminated> SumaProductoNumeros [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java.exe
Introduce primer valor:18
Introduce segundo valor27
La suma de los dos valores es:45
El producto de los dos valores es:486

```
import java.util.Scanner;
```

```
public class SumaProductoNumeros {  
    public static void main(String[] ar) {  
        Scanner teclado=new Scanner(System.in);  
        int num1,num2,suma,producto;  
        System.out.print("Introduce primer valor:");  
        num1=teclado.nextInt();  
        System.out.print("Introduce segundo valor");  
        num2=teclado.nextInt();  
        suma=num1 + num2;  
        producto=num1 * num2;  
        System.out.print("La suma de los dos valores es:");  
        System.out.println(suma);  
        System.out.print("El producto de los dos valores es:");  
        System.out.println(producto);  
    }  
}
```

Recordemos que tenemos que seguir todos los pasos vistos para la creación de un proyecto, su clase, definición de la función main y la codificación del diagrama de flujo (como son problemas muy sencillos con una única clase puede especificar el nombre del proyecto con el mismo nombre de la clase: SumaProductoNumeros)

Algunas cosas nuevas que podemos notar:

Podemos definir varias variables en la misma línea:

```
int num1,num2,suma,producto;
```

Si llamamos a la función println en lugar de print, la impresión siguiente se efectuará en la próxima línea:

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


Problemas propuestos

1. Realizar la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro)
2. Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y el cuarto.
3. Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.
4. Se debe desarrollar un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

7 - Estructuras condicionales simples y compuestas

No todos los problemas pueden resolverse empleando estructuras secuenciales.

Cuando hay que tomar una decisión aparecen las estructuras condicionales.

En nuestra vida diaria se nos presentan situaciones donde debemos decidir.

¿Elijo la carrera A o la carrera B?

¿Me pongo este pantalón?

Para ir al trabajo, ¿elijo el camino A o el camino B?

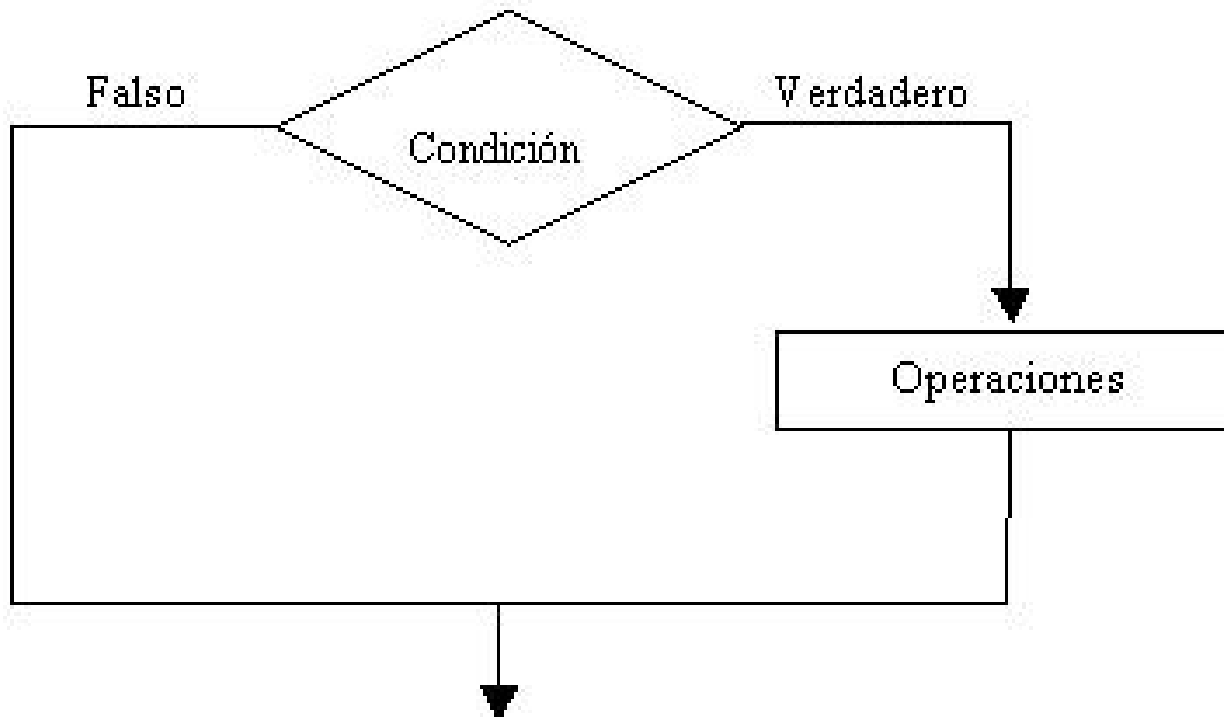
Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple.

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.

Representación gráfica:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.

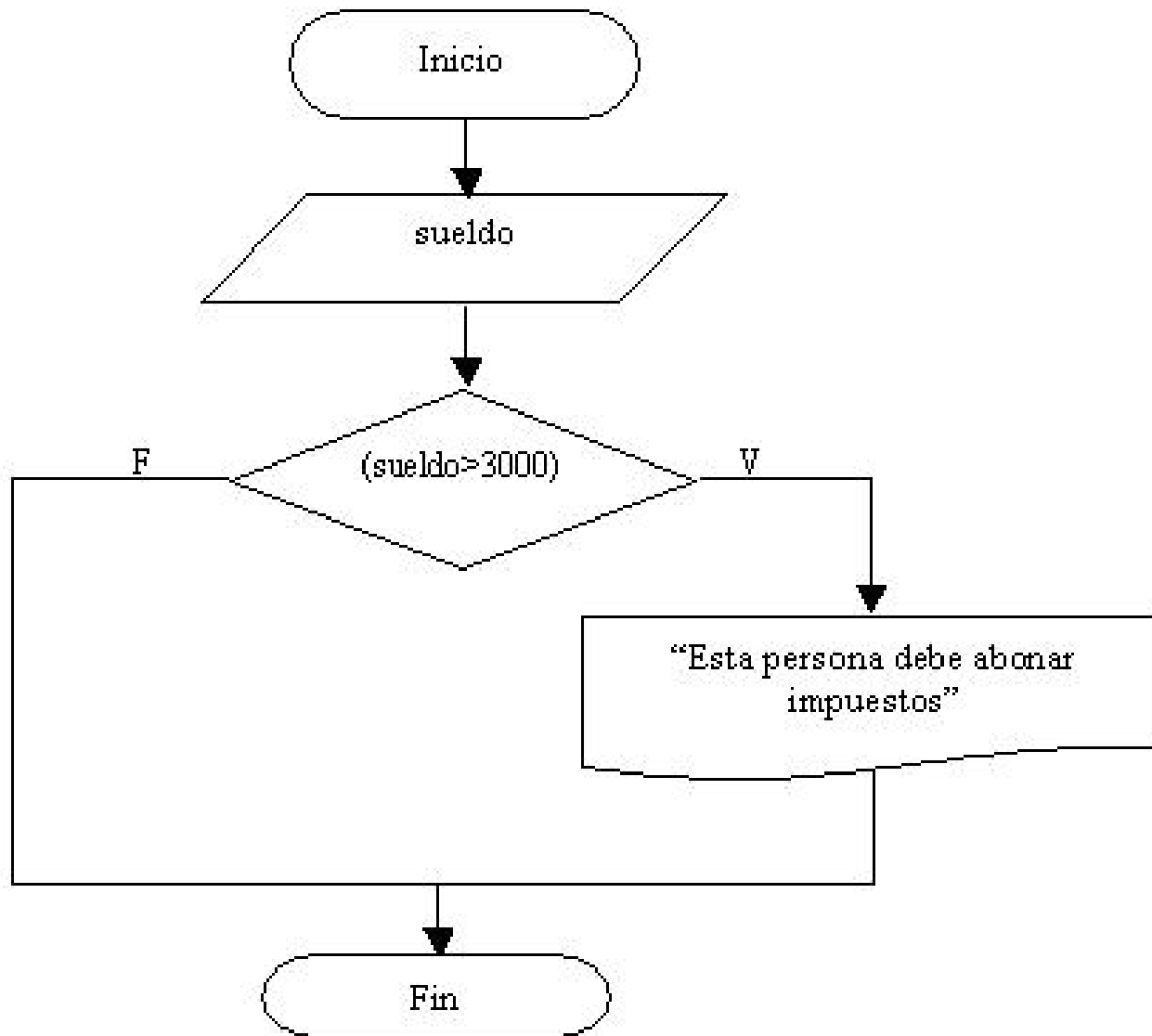
Se trata de una estructura **CONDICIONAL SIMPLE** porque por el camino del verdadero hay actividades y por el camino del falso no hay actividades.

Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

Problema:

Introducir el sueldo de una persona, si supera los 3000 euros mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo:



Podemos observar lo siguiente: Siempre se hace la carga del sueldo, pero si el sueldo que introducimos supera 3000 euros se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 3000 o menos no aparece nada por pantalla.

Programa:

```
import java.util.Scanner;

public class EstructuraCondicionalSimple{
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        float sueldo;
        System.out.print("Introduce el sueldo:");
        sueldo=teclado.nextFloat();
        if (sueldo>3000) {
            System.out.println("Esta persona debe abonar impuestos");
        }
    }
}
```

EstructuraCondicionalSimple.java X

```
1
2 import java.util.Scanner;
3
4 public class EstructuraCondicionalSimple {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         float sueldo;
8         System.out.print("Introduce el sueldo:");
9         sueldo=teclado.nextFloat();
10        if (sueldo>3000) {
11            System.out.println("Esta persona debe abonar impuestos");
12        }
13    }
14 }
15
16
```

Problems @ Javadoc Declaration Console X

<terminated> EstructuraCondicionalSimple [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\

Introduce el sueldo:3500

Esta persona debe abonar impuestos

La palabra clave "if" indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves las instrucciones de la rama del verdadero.

Es necesario que las instrucciones a ejecutar en caso que la condición sea verdadera estén encerradas entre llaves { }, con ellas marcamos el comienzo y el fin del bloque del verdadero.

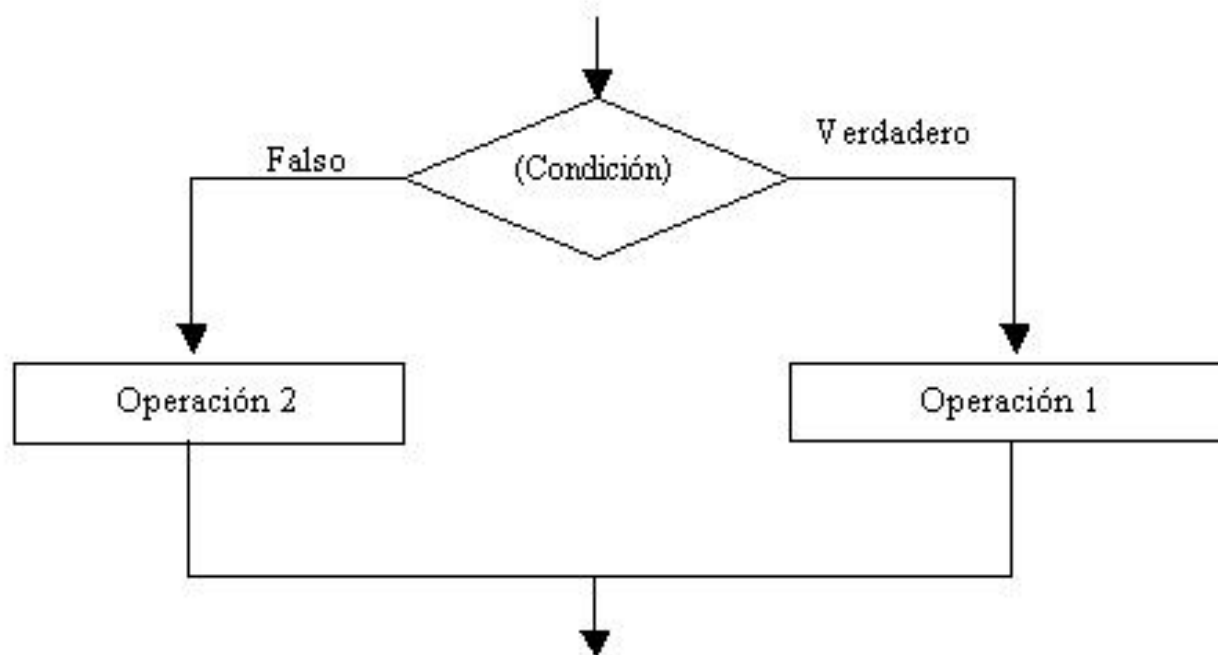
Ejecutando el programa e introduciendo un sueldo superior a 3000 euros. Podemos observar como aparece en pantalla el mensaje "Esta persona debe abonar impuestos", ya que la condición del if es verdadera.

Volvamos a ejecutar el programa y cargamos un sueldo menor o igual a 3000 euros. No debe aparecer el mensaje en pantalla.

Estructura condicional compuesta.

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

Representación gráfica:

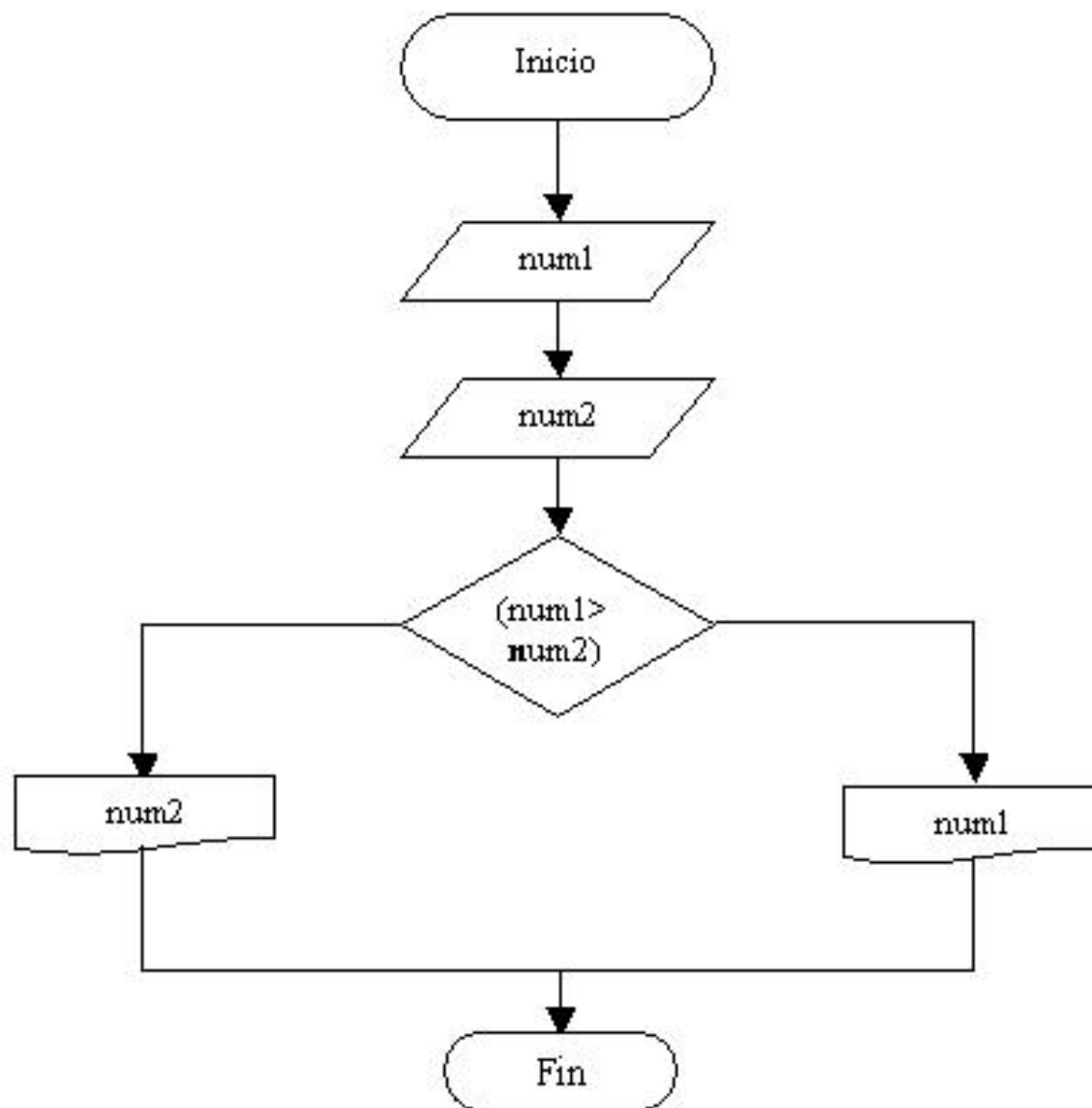


En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

Problema:

Realizar un programa que solicite introducir dos números distintos y muestre por pantalla el mayor de ellos.

Diagrama de flujo:



Se hace la entrada de num1 y num2 por teclado. Para saber que variable tiene un valor mayor preguntamos si el contenido de num1 es mayor ($>$) que el contenido de num2, si la respuesta es verdadera vamos por la rama de la derecha e imprimimos num1, en caso que la condición sea falsa vamos por la rama de la izquierda (Falsa) e imprimimos num2.

Como podemos observar nunca se imprimen num1 y num2 simultáneamente.

Estamos en presencia de una ESTRUCTURA CONDICIONAL COMPUESTA ya que tenemos actividades por la rama del verdadero y del falso.

Programa:

```
import java.util.Scanner;

public class EstructuraCondicionalCompuesta{
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2;
        System.out.print("Introduce primer valor:");
        num1=teclado.nextInt();
        System.out.print("Introduce segundo valor:");
        num2=teclado.nextInt();
        if (num1>num2) {
            System.out.print(num1);
        } else {
            System.out.print(num2);
        }
    }
}
```

EstructuraCondicionalCompuesta.java X

```
1
2 import java.util.Scanner;
3
4 public class EstructuraCondicionalCompuesta{
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int num1,num2;
8         System.out.print("Introduce primer valor:");
9         num1=teclado.nextInt();
10        System.out.print("Introduce segundo valor:");
11        num2=teclado.nextInt();
12        if (num1>num2) {
13            System.out.print(num1);
14        } else {
15            System.out.print(num2);
16        }
17    }
18 }
19
20
```

Problems @ Javadoc Declaration Console X

<terminated> EstructuraCondicionalCompuesta [Java Application] C:\Program
Introduce primer valor:128
Introduce segundo valor:356
356

Cotejemos el diagrama de flujo y la codificación y observemos que el primer bloque de llaves después del if representa la rama del verdadero y el segundo bloque de llaves representa la rama del falso.

Compilamos el programa, si hubo errores sintácticos corriamos y carguamos dos valores:

Introduce el primer valor: 10

Introduce el segundo valor: 4

10

Si introducimos los valores 10 y 4 la condición del if retorna verdadero y ejecuta el primer bloque.

Un programa se controla y corrige probando todos sus posibles resultados.

Ejecutemos nuevamente el programa e introducimos:

Introduce el primer valor: 10

Introduce el segundo valor: 54

54

Cuando a un programa le corregimos todos los errores sintácticos y lógicos ha terminado nuestra tarea y podemos entregar el mismo al USUARIO.

Operadores

En una condición deben disponerse únicamente variables, valores constantes y operadores relacionales.

Operadores Relacionales:

> (mayor)

< (menor)

>= (mayor o igual)

<= (menor o igual)

== (igual)

!= (distinto)

Operadores Matemáticos:

+

-

*

/

% (resto de una división) Ej.: $x=13\%5$; {se guarda 3}

Hay que tener en cuenta que al disponer una condición debemos seleccionar que operador relacional se adapta a la pregunta.

Ejemplos:

Se introduce un número multiplicarlo por 10 si es distinto a 0. (\neq)

Se introducen dos números mostrar una advertencia si son iguales. ($=$)

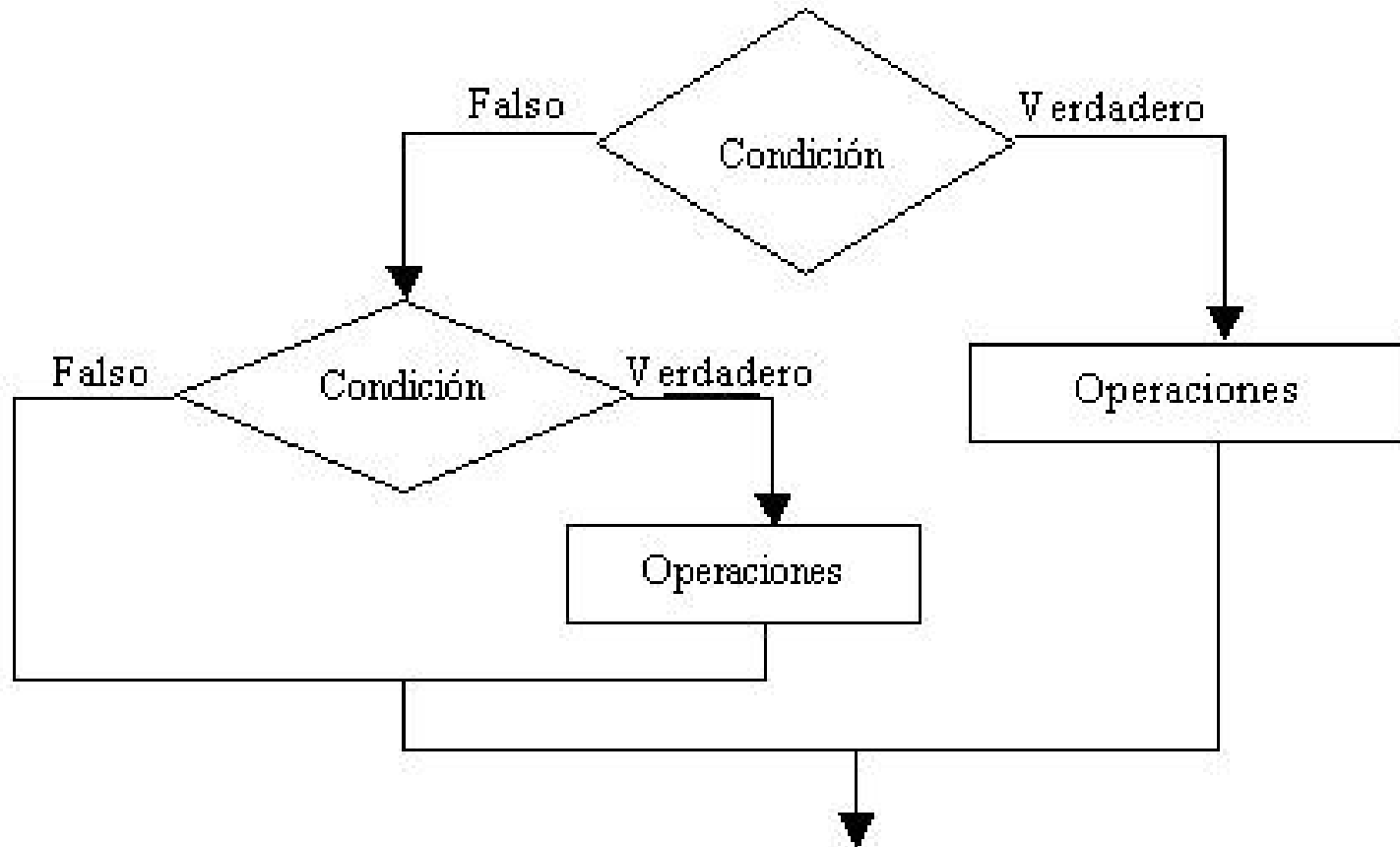
Los problemas que se pueden presentar son infinitos y la correcta elección del operador sólo se alcanza con la práctica intensiva en la resolución de problemas.

Problemas propuestos

1. Realizar un programa que lea por teclado dos números, si el primero es mayor al segundo informar su suma y diferencia, en caso contrario informar el producto y la división del primero respecto al segundo.
2. Se introducen tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Promocionado".
3. Se introducen por teclado un número positivo de uno o dos dígitos (1..99) mostrar un mensaje indicando si el número tiene uno o dos dígitos.
(Tener en cuenta que condición debe cumplirse para tener dos dígitos, un número entero)

8 - Estructuras condicionales anidadas

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.



El diagrama de flujo que se presenta contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primer estructura.

Es común que se presenten estructuras condicionales anidadas aún más complejas.

Problema:

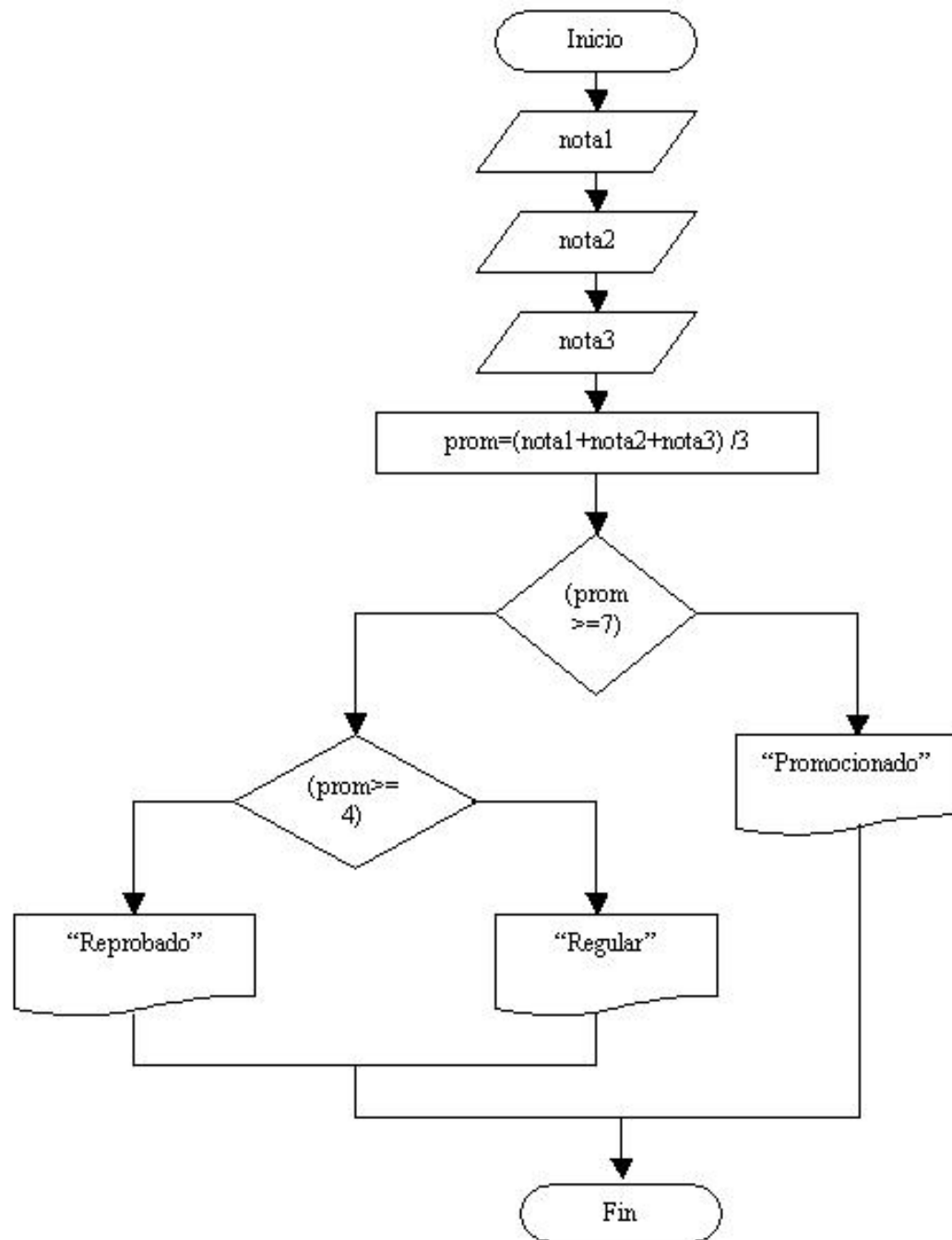
Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:

Si el promedio es ≥ 7 mostrar "Promocionado".

Si el promedio es ≥ 4 y < 7 mostrar "Regular".

Si el promedio es < 4 mostrar "Reprobado".

Diagrama de flujo:



Analicemos el siguiente diagrama. Se ingresan tres valores por teclado que representan las notas de un alumno, se obtiene el promedio sumando los tres valores y dividiendo por 3 dicho resultado (Tener en cuenta que si el resultado es un valor real solo se almacena la parte entera).

Primeramente preguntamos si el promedio es superior o igual a 7, en caso afirmativo va por la rama del verdadero de la estructura condicional mostramos un mensaje que indica "Promocionado" (con comillas indicamos un texto que debe imprimirse en pantalla).

En caso que la condición nos de falso, por la rama del falso aparece otra estructura condicional, porque todavía debemos averiguar si el promedio del alumno es superior o igual a cuatro o inferior a cuatro.

Estamos en presencia de dos estructuras condicionales compuestas.

Programa:

```
import java.util.Scanner;
```

```
public class EstructuraCondicionalAnidada1 {  
    public static void main(String[] ar) {  
        Scanner teclado=new Scanner(System.in);  
        int nota1,nota2,nota3;  
        System.out.print("Introduce primera nota:");  
        nota1=teclado.nextInt();  
        System.out.print("Introduce segunda nota:");  
        nota2=teclado.nextInt();  
        System.out.print("Introduce tercera nota:");  
        nota3=teclado.nextInt();  
        int promedio=(nota1 + nota2 + nota3) / 3;  
        if (promedio>=7) {  
            System.out.print("Promocionado");  
        } else {  
            if (promedio>=4) {  
                System.out.print("Regular");  
            } else {  
                System.out.print("Reprobado");  
            }  
        }  
    }  
}
```

```
1
2 import java.util.Scanner;
3
4 public class EstructuraCondicionalAnidada1 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int nota1,nota2,nota3;
8         System.out.print("Introduce primera nota:");
9         nota1=teclado.nextInt();
10        System.out.print("Introduce segunda nota:");
11        nota2=teclado.nextInt();
12        System.out.print("Introduce tercera nota:");
13        nota3=teclado.nextInt();
14        int promedio=(nota1 + nota2 + nota3) / 3;
15        if (promedio>=7) {
16            System.out.print("Promocionado");
17        } else {
18            if (promedio>=4) {
19                System.out.print("Regular");
20            } else {
21                System.out.print("Reprobado");
22            }
23        }
24    }
25 }
```

<terminated> EstructuraCondicionalAnidada1 [Java Application] C:\Program

Introduce primera nota:4

Introduce segunda nota:6

Introduce tercera nota:7

Regular

Codifiquemos y ejecutemos este programa. Al correr el programa deberá solicitar por teclado la carga de tres notas y mostrarnos un mensaje según el promedio de las mismas.

Podemos definir un conjunto de variables del mismo tipo en una misma línea:

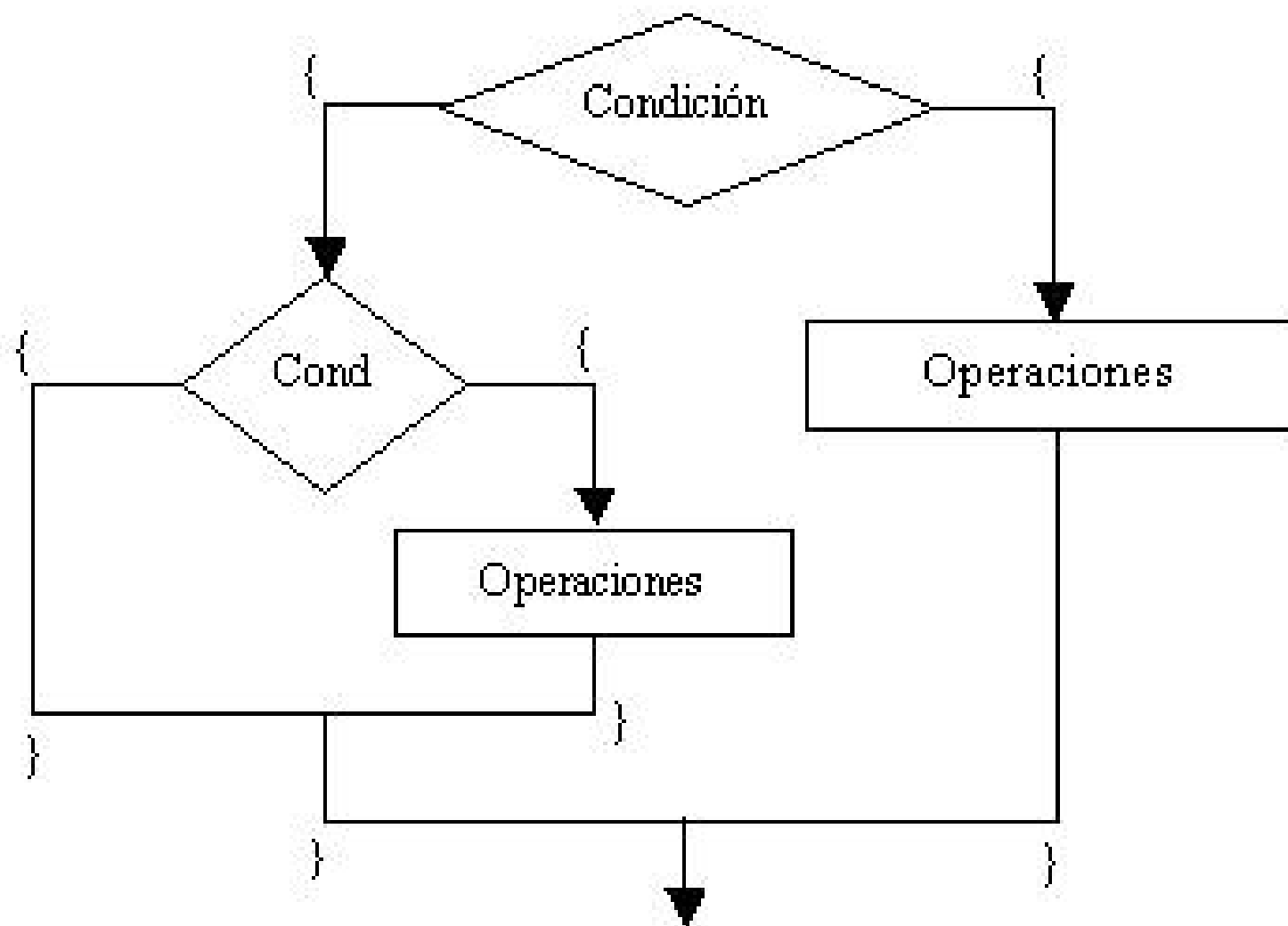
```
int nota1,nota2,nota3;
```

Esto no es obligatorio pero a veces, por estar relacionadas, conviene.

A la codificación del if anidado podemos observarla por el else del primer if.

Para no tener problemas (olvidarnos) con las llaves de apertura y cerrado podemos ver la siguiente regla:

Cada vértice representa una llave de apertura y una de cierre:



Problemas propuestos

1. Se cargan por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.
2. Se introduce por teclado un valor entero, mostrar una leyenda que indique si el número es positivo, nulo o negativo.
3. Confeccionar un programa que permita cargar un número entero positivo de hasta tres cifras y muestre un mensaje indicando si tiene 1, 2, o 3 cifras. Mostrar un mensaje de error si el número de cifras es mayor.
4. Un postulante a un empleo, realiza un test de capacitación, se obtuvo la siguiente información: cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido, y sabiendo que:

| | |
|-----------------|-------------------------------------|
| Nivel máximo: | Porcentaje $\geq 90\%$. |
| Nivel medio: | Porcentaje $\geq 75\%$ y $< 90\%$. |
| Nivel regular: | Porcentaje $\geq 50\%$ y $< 75\%$. |
| Fuera de nivel: | Porcentaje $< 50\%$. |

9 - Condiciones compuestas con operadores lógicos

Hasta ahora hemos visto los operadores:

relacionales ($>$, $<$, $>=$, $<=$, $==$, $!=$)

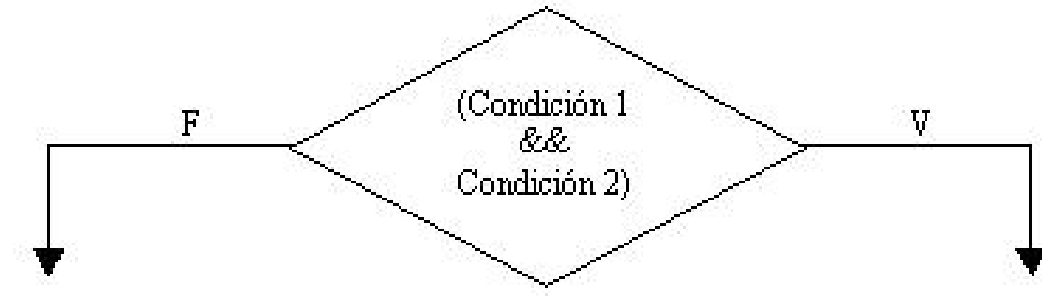
matemáticos ($+$, $-$, $*$, $/$, $\%$)

pero nos están faltando otros operadores imprescindibles:

lógicos ($\&\&$, $\|\|$).

Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples.

Operador &&



Traducido se lee como Y. Si la Condición 1 es verdadera Y la condición 2 es verdadera luego ejecutar la rama del verdadero.

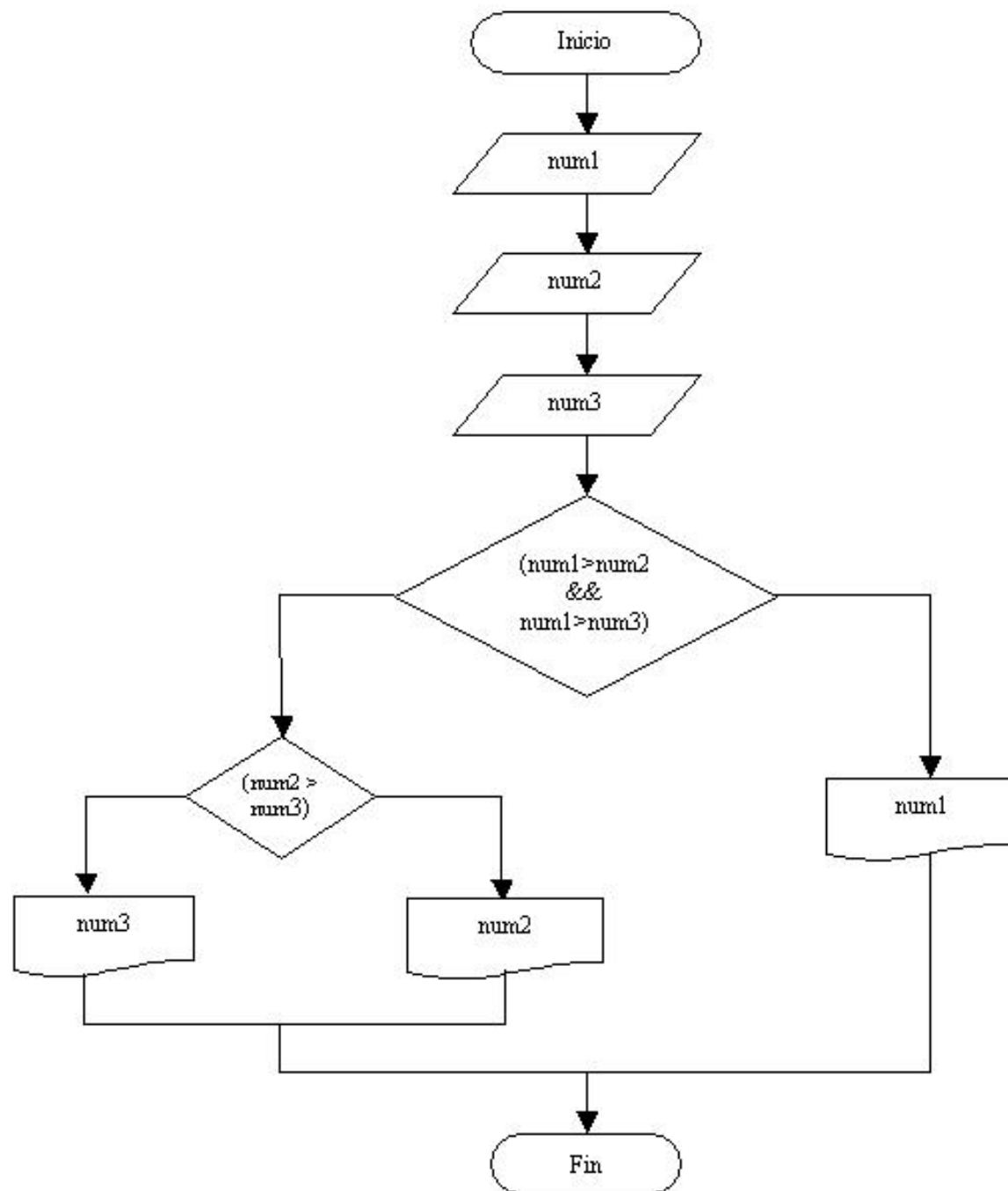
Cuando vinculamos dos o más condiciones con el operador &&, las dos condiciones deben ser verdaderas para que el resultado de la condición compuesta de Verdadero y continúe por la rama del verdadero de la estructura condicional.

La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

Problema:

Confeccionar un programa que lea por teclado tres números distintos y nos muestre el mayor.

Diagrama de flujo:



Este ejercicio está resuelto sin emplear operadores lógicos en un concepto anterior del tutorial. La primera estructura condicional es una ESTRUCTURA CONDICIONAL COMPUESTA con una CONDICION COMPUESTA.

Podemos leerla de la siguiente forma:

Si el contenido de la variable num1 es mayor al contenido de la variable num2
Y si el contenido de la variable num1 es mayor al contenido de la variable num3 entonces la CONDICION COMPUESTA resulta Verdadera.

Si una de las condiciones simples da falso la CONDICION COMPUESTA da Falso y continua por la rama del falso.

Es decir que se mostrará el contenido de num1 si y sólo si $\text{num1} > \text{num2}$ y $\text{num1} > \text{num3}$.

En caso de ser Falsa la condición, analizamos el contenido de num2 y num3 para ver cual tiene un valor mayor.

En esta segunda estructura condicional no se requieren operadores lógicos al haber una condición simple.

Programa:

```
import java.util.Scanner;

public class CondicionesCompuestas1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,num3;
        System.out.print("Introduce primer valor:");
        num1=teclado.nextInt();
        System.out.print("Introduce segundo valor:");
        num2=teclado.nextInt();
        System.out.print("Introduce tercer valor:");
        num3=teclado.nextInt();
        if (num1>num2 && num1>num3) {
            System.out.print(num1);
        } else {
            if (num2>num3) {
                System.out.print(num2);
            } else {
                System.out.print(num3);
            }
        }
    }
}
```


CondicionesCompuestas1.java X

```
4 public class CondicionesCompuestas1 {  
5     public static void main(String[] ar) {  
6         Scanner teclado=new Scanner(System.in);  
7         int num1,num2,num3;  
8         System.out.print("Introduce primer valor:");  
9         num1=teclado.nextInt();  
10        System.out.print("Introduce segundo valor:");  
11        num2=teclado.nextInt();  
12        System.out.print("Introduce tercer valor:");  
13        num3=teclado.nextInt();  
14        if (num1>num2 && num1>num3) {  
15            System.out.print(num1);  
16        } else {  
17            if (num2>num3) {  
18                System.out.print(num2);  
19            }else {  
20                System.out.print(num3);  
21            }  
22        }  
23    }  
24 }  
25
```

Problems @ Javadoc Declaration Console X

<terminated> CondicionesCompuestas1 [Java Application] C:\Program Files\Ja

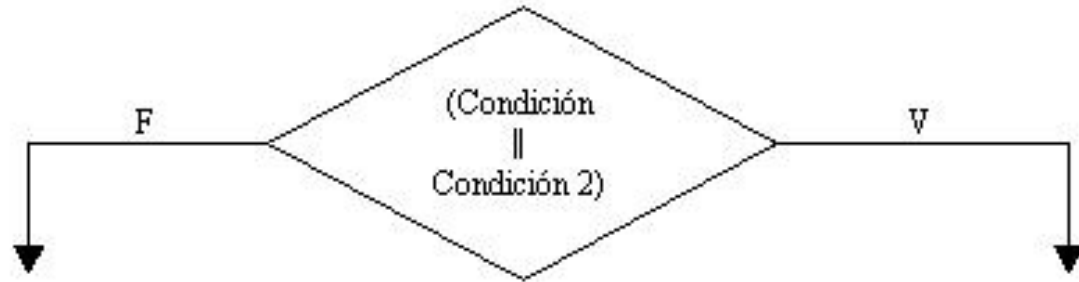
Introduce primer valor:46

Introduce segundo valor:128

Introduce tercer valor:37

128

Operador ||



Traducido se lo lee como O. Si la condición 1 es Verdadera O la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.

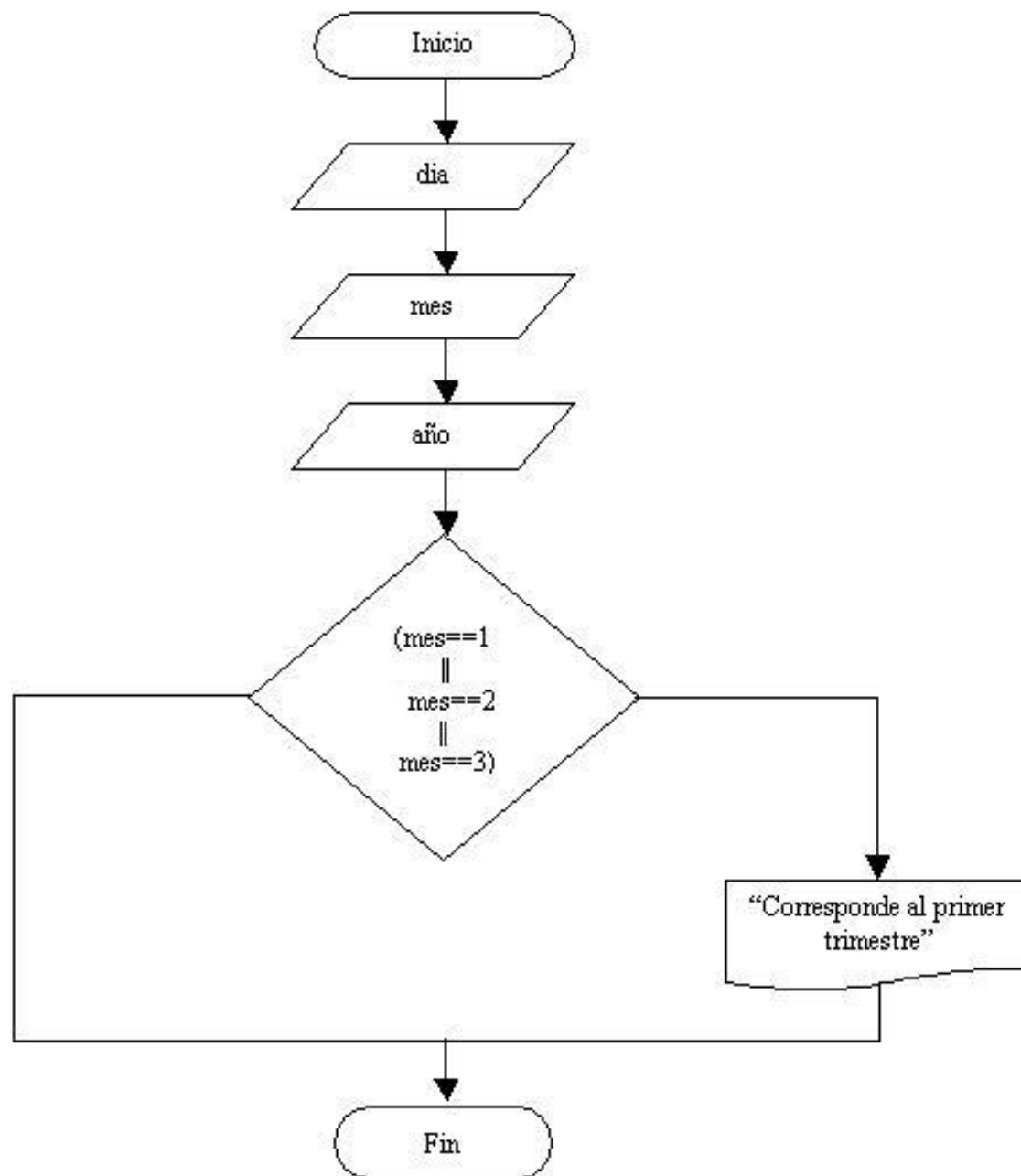
Cuando vinculamos dos o más condiciones con el operador “Or”, con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

Problema:

Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primer trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y año.

Ejemplo: dia:10 mes:1 año: 2010.

Diagrama de flujo:



La carga de una fecha se hace por partes, ingresamos las variables día, mes y año.

Mostramos el mensaje "Corresponde al primer trimestre" en caso que el mes introducido por teclado sea igual a 1, 2 ó 3.

En la condición no participan las variables día y año.

Programa:

```
import java.util.Scanner;

public class CondicionesCompuestas2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int dia,mes,año;
        System.out.print("Introduce numero de día:");
        dia=teclado.nextInt();
        System.out.print("Introduce numero de mes:");
        mes=teclado.nextInt();
        System.out.print("Introduce numero de año:");
        año=teclado.nextInt();
        if (mes==1 || mes==2 || mes==3) {
            System.out.print("Corresponde al primer trimestre");
        }
    }
}
```

CondicionesCompuestas2.java X

```
1
2 import java.util.Scanner;
3
4 public class CondicionesCompuestas2 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int dia,mes,año;
8         System.out.print("Introduce numero de día:");
9         dia=teclado.nextInt();
10        System.out.print("Introduce numero de mes:");
11        mes=teclado.nextInt();
12        System.out.print("Introduce numero de año:");
13        año=teclado.nextInt();
14        if (mes==1 || mes==2 || mes==3) {
15            System.out.print("Corresponde al primer trimestre");
16        }
17    }
18 }
19
```

Problems @ Javadoc Declaration Console X

<terminated> CondicionesCompuestas2 [Java Application] C:\Program Files\Java\jdk-17.0.1\

Introduce numero de día:5

Introduce numero de mes:3

Introduce numero de año:2015

Corresponde al primer trimestre

Problemas propuestos

1. Realizar un programa que pida cargar una fecha cualquiera, luego verificar si dicha fecha corresponde a Navidad.
2. Se introducen tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.
3. Se introducen por teclado tres números, si todos los valores ingresados son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores a diez".
4. Se introducen por teclado tres números, si al menos uno de los valores ingresados es menor a 10, imprimir en pantalla la leyenda "Alguno de los números es menor a diez".
5. Escribir un programa que pida introducir la coordenada de un punto en el plano, es decir dos valores enteros x e y (distintos a cero).
Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1° Cuadrante si $x > 0$ Y $y > 0$, 2° Cuadrante: $x < 0$ Y $y > 0$, etc.)

6. De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entrada e informe:
 - a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20 %, mostrar el sueldo a pagar.
 - b) Si el sueldo es inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5 %.
 - c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.
7. Escribir un programa en el cual: dada una lista de tres valores numéricos distintos se calcule e informe su rango de variación (debe mostrar el mayor y el menor de ellos)

10 - Estructura repetitiva while

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.

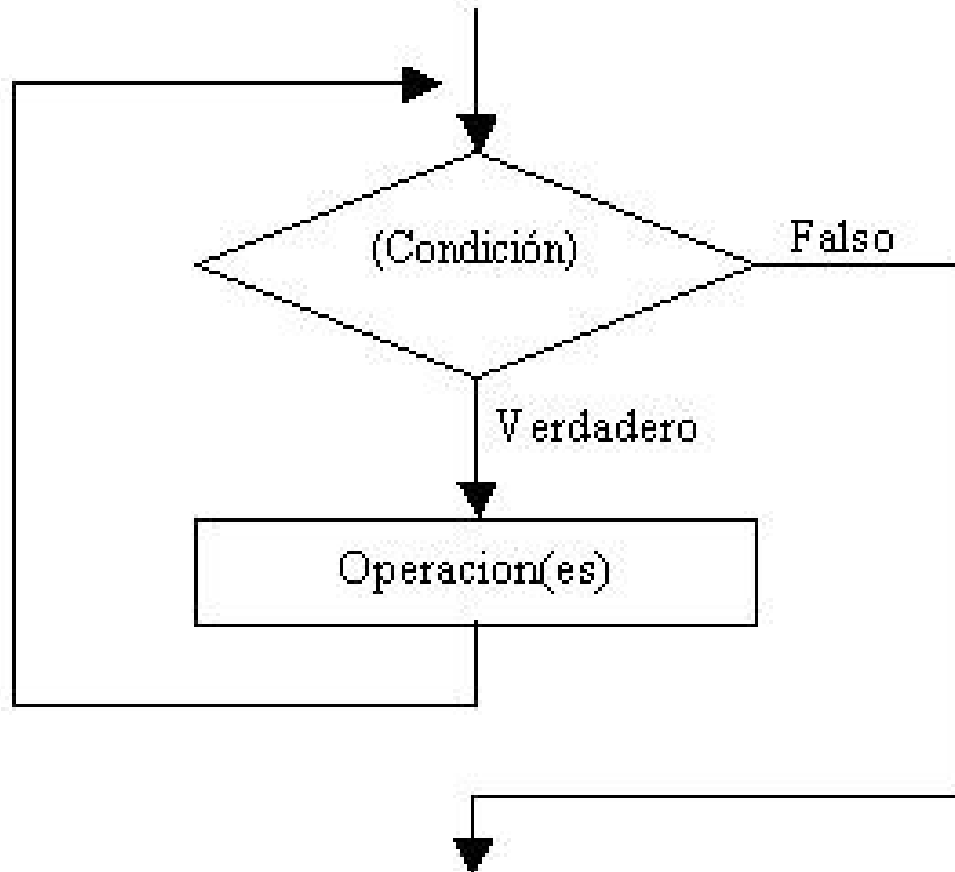
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Estructura repetitiva while.

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero. A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.

En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea Verdadera.

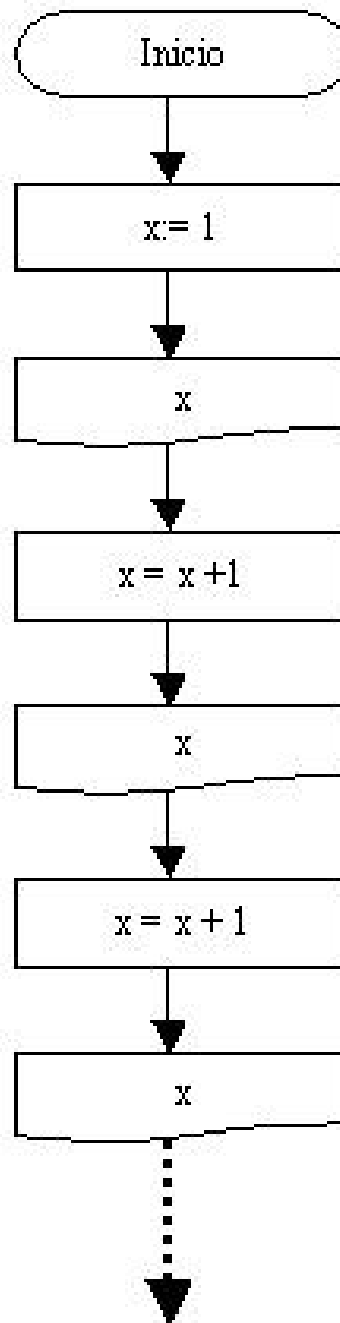
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

Diagrama de flujo:



Se imprime un 1

Se incrementa la variable x con el valor que tiene más uno.

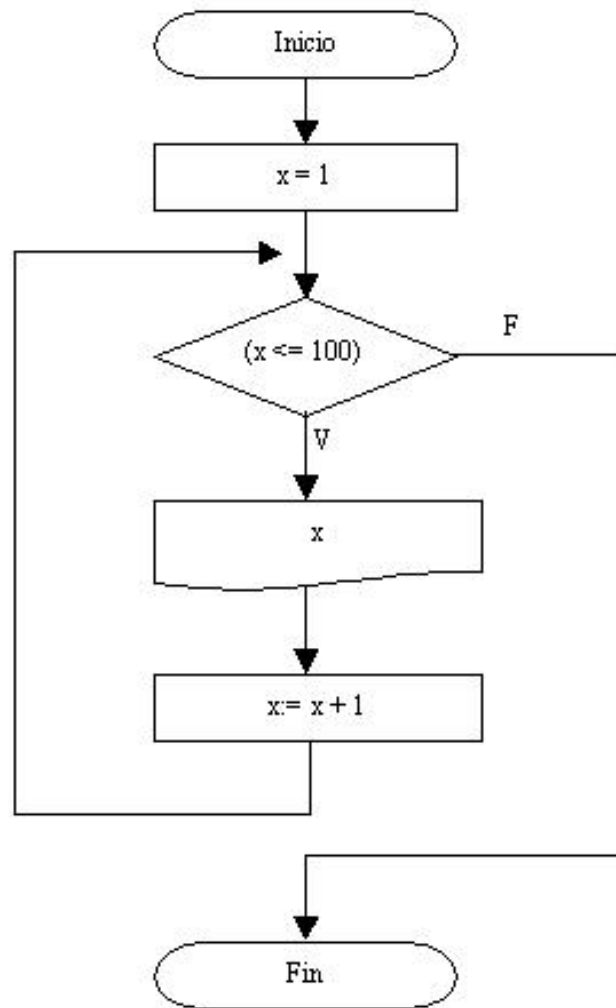
Se imprime un 2

Se imprime un 3

Continúa hasta mostrar el 100

Si continuamos con el diagrama no nos alcanzarían las próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en Java muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:



Es muy importante analizar este diagrama:

La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva `while` y disponemos la siguiente condición ($x \leq 100$), se lee **MIENTRAS** la variable x sea menor o igual a 100.

Al ejecutarse la condición retorna **VERDADERO** porque el contenido de x (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura `while`. El bloque de instrucciones contiene una salida y una operación.

Se imprime el contenido de x , y seguidamente se incrementa la variable x en uno.

La operación $x = x + 1$ se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.

Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.

Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.

Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición $x \geq 100$ (si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.

Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer un seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

x

1

2

3

4

.

.

100

101 Cuando x vale 101 la condición de la estructura repetitiva retorna falso, en este caso finaliza el diagrama.

Importante: Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso la primera vez.

La variable x debe estar inicializada con algún valor antes que se ejecute la operación $x = x + 1$ en caso de no estar inicializada aparece un error de compilación.

Programa:

```
public class EstructuraRepetitivaWhile1 {  
    public static void main(String[] ar) {  
        int x;  
        x=1;  
        while (x<=100) {  
            System.out.print(x);  
            System.out.print(" - ");  
            x = x + 1;  
        }  
    }  
}
```

```
1
2 public class EstructuraRepetitivaWhile1 {
3     public static void main(String[] ar) {
4         int x;
5         x=1;
6         while (x<=100) {
7             System.out.print(x);
8             System.out.print(" - ");
9             x = x + 1;
10        }
11    }
12 }
13
14
```

Importante: Como podemos observar no hemos creado un objeto de la clase Scanner. Esto es debido a que en este programa no hay que introducir datos por teclado. Para las salidas utilizamos la función print, que se encuentra creada por defecto en cualquier programa que codifiquemos en Java.

Recordemos que un problema no estará 100% solucionado si no hacemos el programa en Java que muestre los resultados buscados.

Probemos algunas modificaciones de este programa y veamos que cambios se deberían hacer para:

- 1 - Imprimir los números del 1 al 500.
- 2 - Imprimir los números del 50 al 100.
- 3 - Imprimir los números del -50 al 0.
- 4 - Imprimir los números del 2 al 100 pero de 2 en 2 (2,4,6,8100).

Respuestas:

- 1 - Debemos cambiar la condición del while con $x \leq 500$.
- 2 - Debemos inicializar x con el valor 50.
- 3 - Inicializar x con el valor -50 y fijar la condición $x \leq 0$.
- 4 - Inicializar a x con el valor 2 y dentro del bloque repetitivo incrementar a x en 2 ($x = x + 2$).

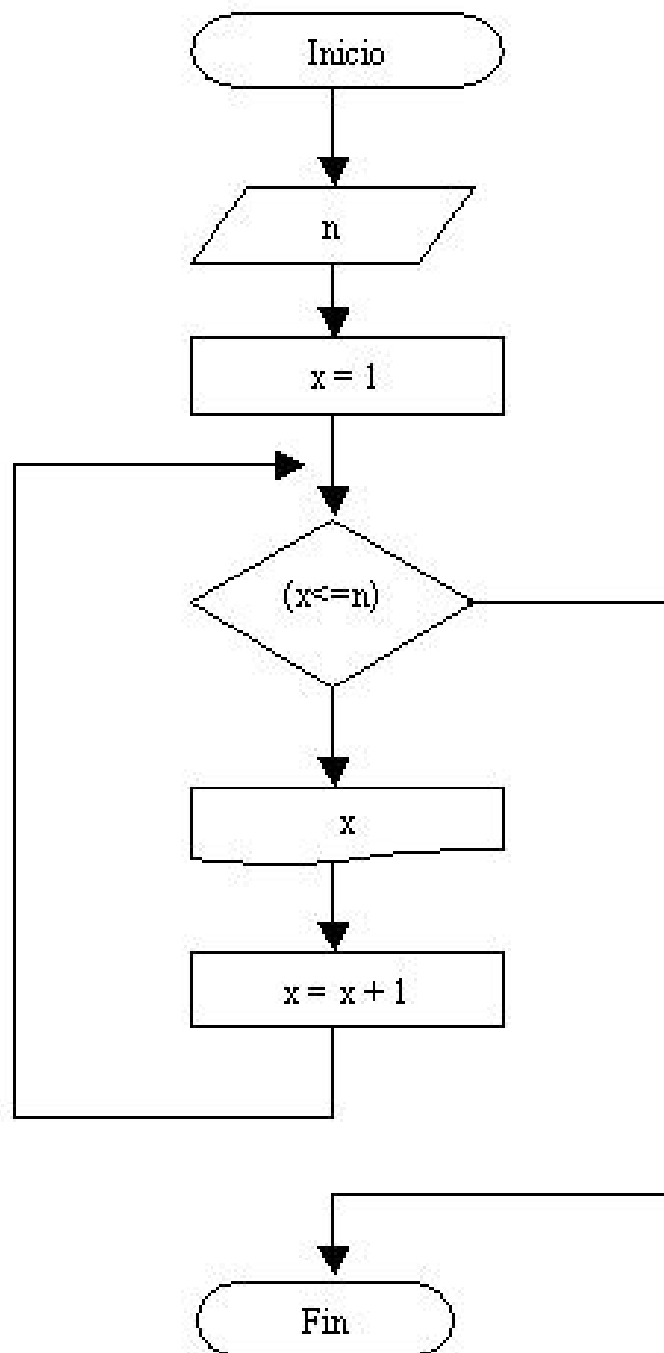
Problema 2:

Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor introducido de uno en uno.

Ejemplo: Si introducimos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en Java de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

Diagrama de flujo:



Podemos observar que se introduce por teclado la variable n. El operador puede cargar cualquier valor.

Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es Mientras $x \leq n$, es decir, mientras x sea menor o igual a 10; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

A la prueba del diagrama la podemos realizar dándole valores a las variables; por ejemplo, si introducimos 5 el seguimiento es el siguiente:

| n | x |
|---|---|
| 5 | 1 (Se imprime el contenido de x) |
| | 2 " " |
| | 3 " " |
| | 4 " " |
| | 5 " " |
| | 6 (Sale del while porque 6 no es menor o igual a 5) |

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int n,x;
        System.out.print("Introduce el valor final:");
        n=teclado.nextInt();
        x=1;
        while (x <= n) {
            System.out.print(x);
            System.out.print(" - ");
            x = x + 1;
        }
    }
}
```


EstructuraRepetitivaWhile2.java ×

```
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaWhile2 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int n,x;
8         System.out.print("Introduce el valor final:");
9         n=teclado.nextInt();
10        x=1;
11        while (x<=n) {
12            System.out.print(x);
13            System.out.print(" - ");
14            x = x + 1;
15        }
16    }
17 }
18
```

Problems @ Javadoc Declaration Console ×

<terminated> EstructuraRepetitivaWhile2 [Java Application] C:\Program Files\Java

Introduce el valor final:12

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 -

Los nombres de las variables n y x pueden ser palabras o letras (como en este caso)

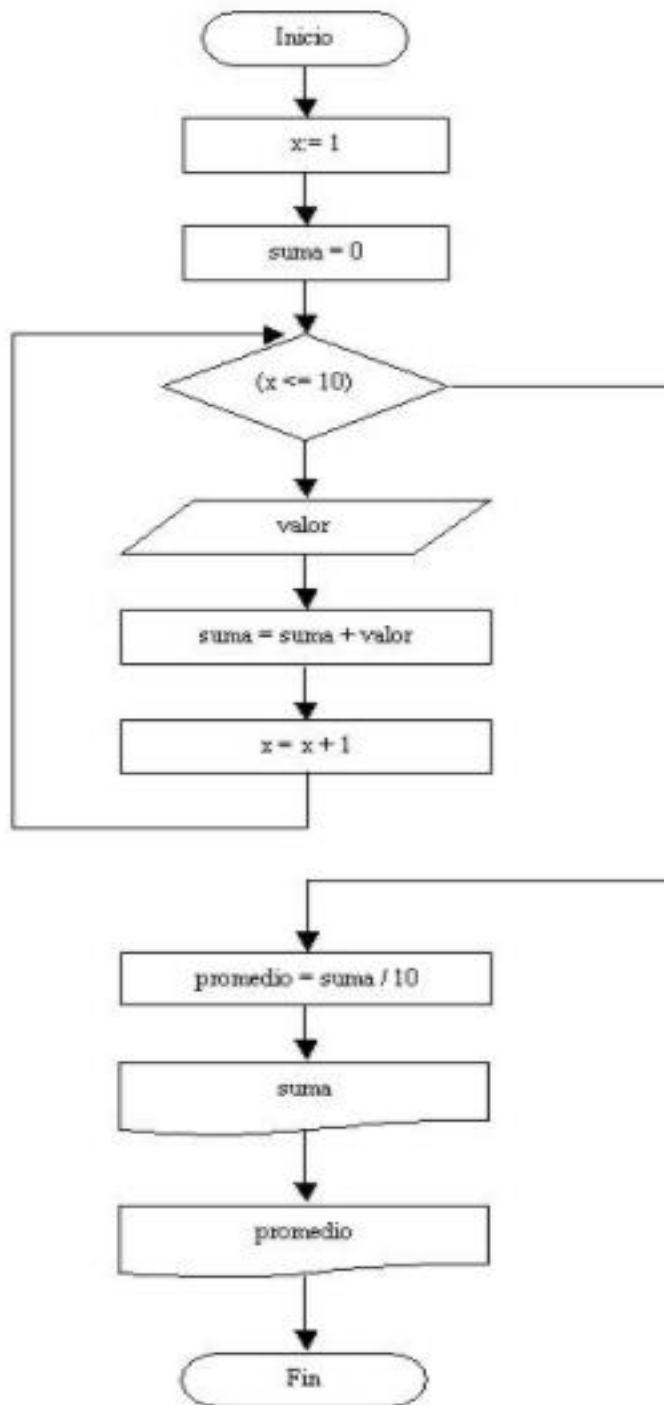
La variable x recibe el nombre de CONTADOR. Un contador es un tipo especial de variable que se incrementa o decrementa con valores constantes durante la ejecución del programa.

El contador x nos indica en cada momento la cantidad de valores impresos en pantalla.

Problema 3:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores introducidos y su promedio.

Diagrama de flujo:



En este problema, a semejanza de los anteriores, llevamos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while.

También aparece el concepto de ACUMULADOR (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa)

Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido introducido en la variable valor.

La prueba del diagrama se realiza dándole valores a las variables:

| valor | suma | x | promedio |
|-------|------|---|----------|
| | 0 | 0 | |

(Antes de entrar a la estructura repetitiva estos son los valores).

| | | |
|----|----|----|
| 5 | 5 | 1 |
| 16 | 21 | 2 |
| 7 | 28 | 3 |
| 10 | 38 | 4 |
| 2 | 40 | 5 |
| 20 | 60 | 6 |
| 5 | 65 | 7 |
| 5 | 70 | 8 |
| 10 | 80 | 9 |
| 2 | 82 | 10 |
| 8 | 90 | 11 |

Este es un seguimiento del diagrama planteado. Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del programa.

El promedio se calcula al salir de la estructura repetitiva (es decir primero sumamos los 10 valores introducidos y luego los dividimos por 10)

Hay que tener en cuenta que cuando en la variable valor se carga el primer valor (en este ejemplo 5) al cargarse el segundo valor (16) el valor anterior 5 se pierde, por ello la necesidad de ir almacenando en la variable suma los valores introducidos.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,suma,valor,promedio;
        x=1;
        suma=0;
        while (x<=10) {
            System.out.print("Introduce un valor:");
            valor=teclado.nextInt();
            suma=suma+valor;
            x=x+1;
        }
        promedio=suma/10;
        System.out.print("La suma de los 10 valores es:");
        System.out.println(suma);
        System.out.print("El promedio es:");
        System.out.print(promedio);
    }
}
```

```
EstructuraRepetitivaWhile3.java ×
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaWhile3 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int x,suma,valor,promedio;
8         x=1;
9         suma=0;
10        while (x<=10) {
11            System.out.print("Introduce un valor:");
12            valor=teclado.nextInt();
13            suma=suma+valor;
14            x=x+1;
15        }
16        promedio=suma/10;
17        System.out.print("La suma de los 10 valores es:");
18        System.out.println(suma);
19        System.out.print("El promedio es:");
20        System.out.print(promedio);
21    }
22 }
```

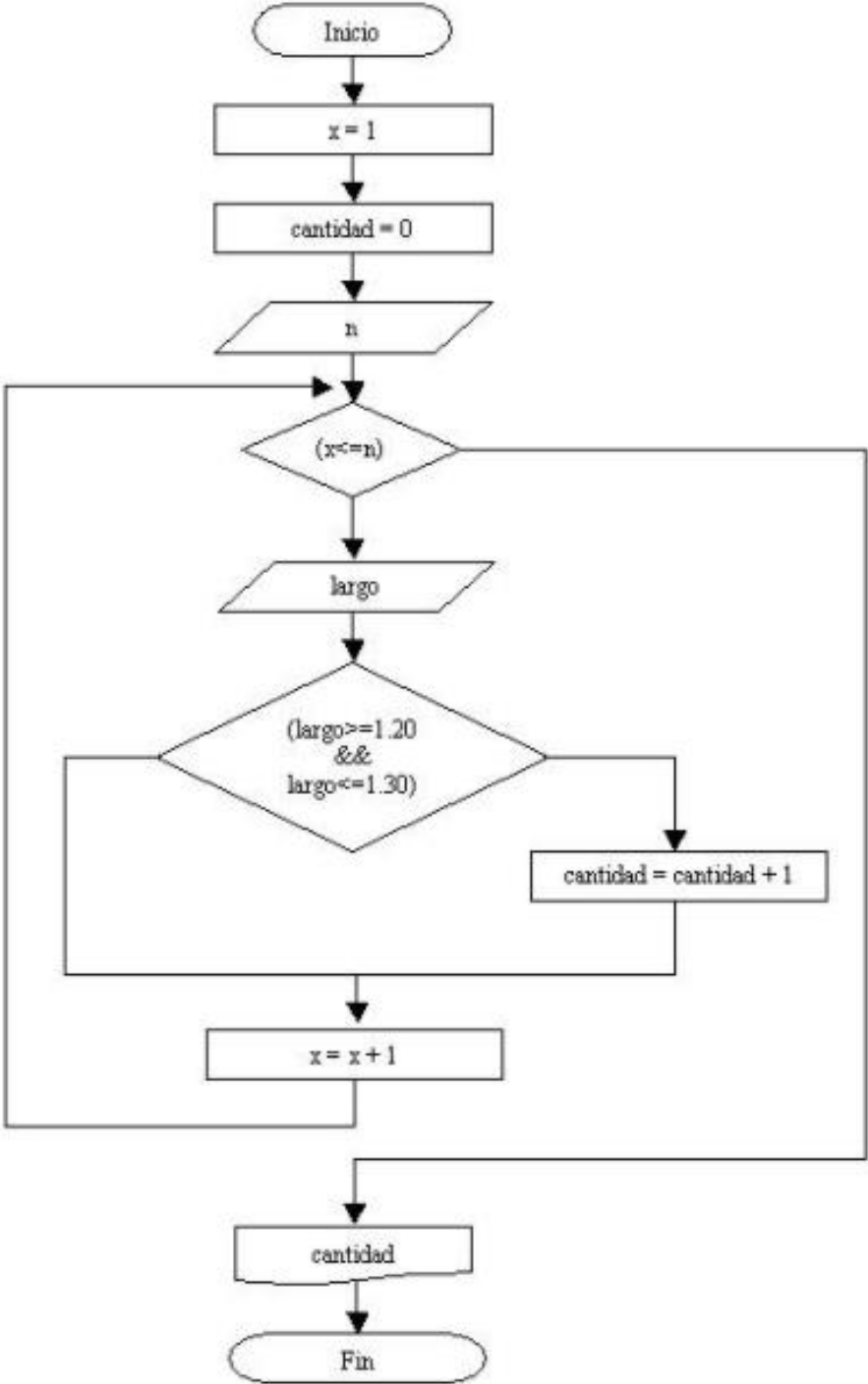
Problems @ Javadoc Declaration Console ×
<terminated> EstructuraRepetitivaWhile3 [Java Application] C:\Program Files\Java\jdk1

```
Introduce un valor:36
Introduce un valor:42
Introduce un valor:159
Introduce un valor:223
Introduce un valor:458
Introduce un valor:65
Introduce un valor:33
Introduce un valor:20
Introduce un valor:32
Introduce un valor:147
La suma de los 10 valores es:1215
El promedio es:121
```


Problema 4:

Una planta que fabrica perfiles de hierro posee un lote de n piezas. Confeccionar un programa que pida introducir por teclado la cantidad de piezas a procesar y luego introduzca la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1,20 y 1,30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

Diagrama de flujo:



Podemos observar que dentro de una estructura repetitiva puede haber estructuras condicionales (inclusive puede haber otras estructuras repetitivas que veremos más adelante)

En este problema hay que cargar inicialmente la cantidad de piezas a introducir(n), seguidamente se cargan n valores con las longitudes de las piezas.

Cada vez que introducimos una longitud de pieza (largo) verificamos si es una medida correcta (debe estar entre 1.20 y 1.30 el largo para que sea correcta), en caso de ser correcta la CONTAMOS (incrementamos la variable cantidad en 1)

Al contador cantidad lo inicializamos en cero porque inicialmente no se ha cargado ningún largo de medida.

Cuando salimos de la estructura repetitiva porque se han cargado n largos de piezas mostramos por pantalla el contador cantidad (que representa la cantidad de piezas aptas)

En este problema tenemos dos CONTADORES:

x (Cuenta la cantidad de piezas cargadas hasta el momento)

cantidad (Cuenta los perfiles de hierro aptos)

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile4 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,cantidad,n;
        float largo;
        x=1;
        cantidad=0;
        System.out.print("Cuantas piezar procesará:");
        n=teclado.nextInt();
        while (x<=n) {
            System.out.print("Introduce la medida de la pieza:");
            largo=teclado.nextFloat();
            if (largo>=1.20 && largo<=1.30) {
                cantidad = cantidad +1;
            }
            x=x + 1;
        }
        System.out.print("La cantidad de piezas aptas son:");
        System.out.print(cantidad);
    }
}
```

```
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaWhile4 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int x,cantidad,n;
8         float largo;
9         x=1;
10        cantidad=0;
11        System.out.print("Cuántas piezas procesará:");
12        n=teclado.nextInt();
13        while (x<=n) {
14            System.out.print("Introduce la medida de la pieza:");
15            largo=teclado.nextFloat();
16            if (largo>=1.20 && largo<=1.30) {
17                cantidad = cantidad +1;
18            }
19            x=x + 1;
20        }
21        System.out.print("La cantidad de piezas aptas son:");
22        System.out.print(cantidad);
```

<terminated> EstructuraRepetitivaWhile4 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java.exe

```
Cuántas piezas procesará:5
Introduce la medida de la pieza:1,20
Introduce la medida de la pieza:1,25
Introduce la medida de la pieza:1,22
Introduce la medida de la pieza:1,23
Introduce la medida de la pieza:1,28
La cantidad de piezas aptas son:5
```

Problemas propuestos

Ha llegado la parte fundamental, que es el momento donde uno desarrolla individualmente un algoritmo para la resolución de problemas.

El tiempo a dedicar a esta sección EJERCICIOS PROPUESTOS debe ser mucho mayor que el empleado a la sección de EJERCICIOS RESUELTOS.

La experiencia dice que debemos dedicar el 80% del tiempo a la resolución individual de problemas y el otro 20% al análisis y codificación de problemas ya resueltos por otras personas.

Es de vital importancia para llegar a ser un buen PROGRAMADOR poder resolver problemas de forma individual.

1. Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.
2. Se ingresan un conjunto de n alturas de personas por teclado. Mostrar la altura promedio de las personas.
3. En una empresa trabajan n empleados cuyos sueldos oscilan entre \$100 y \$500, realizar un programa que lea los sueldos que cobra cada empleado e informe cuántos empleados cobran entre \$100 y \$300 y cuántos cobran más de \$300. Además el programa deberá informar el importe que gasta la empresa en sueldos al personal.
4. Realizar un programa que imprima 25 términos de la serie 11 - 22 - 33 - 44, etc. (No se ingresan valores por teclado)
5. Mostrar los múltiplos de 8 hasta el valor 500. Debe aparecer en pantalla 8 - 16 - 24, etc.

6. Realizar un programa que permita cargar dos listas de 15 valores cada una. Informar con un mensaje cual de las dos listas tiene un valor acumulado mayor (mensajes "Lista 1 mayor", "Lista 2 mayor", "Listas iguales")
Tener en cuenta que puede haber dos o más estructuras repetitivas en un algoritmo.
7. Desarrollar un programa que permita cargar n números enteros y luego nos informe cuántos valores fueron pares y cuántos impares.

Emplear el operador % en la condición de la estructura condicional:

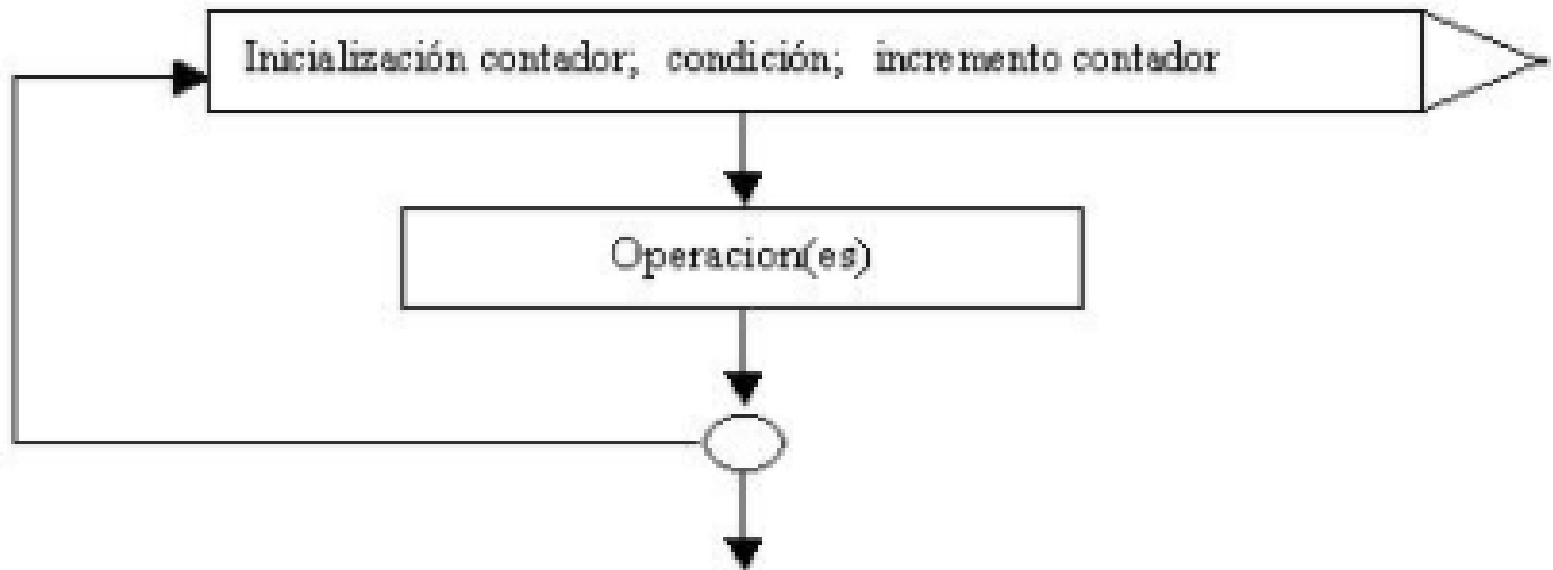
```
if (valor%2==0)      //Si el if da verdadero luego es par.
```


11 - Estructura repetitiva for

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje Java la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:

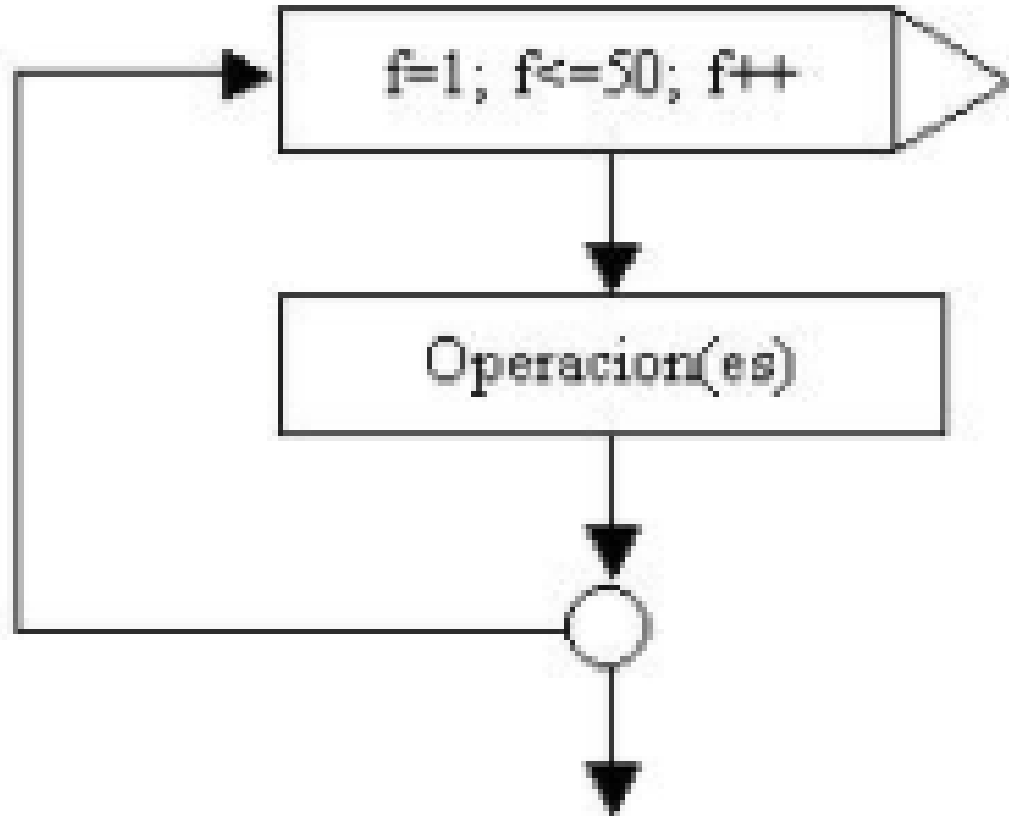


En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercer sección.

Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:



La variable del for puede tener cualquier nombre. En este ejemplo se la ha definido con el nombre f.

Analicemos el ejemplo:

- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale 1 y esto es menor que 50, la condición da verdadero.
- Como la condición fue verdadera, se ejecutan la/s operación/es.
- Al finalizar de ejecutarlas, se retorna a la instrucción f++, por lo que la variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50. Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucciones e incrementa nuevamente la variable del for al terminar el mismo.
- El proceso se repetirá hasta que la variable f sea incrementada al valor 51. En este momento la condición será falsa, y el ciclo se detendrá.

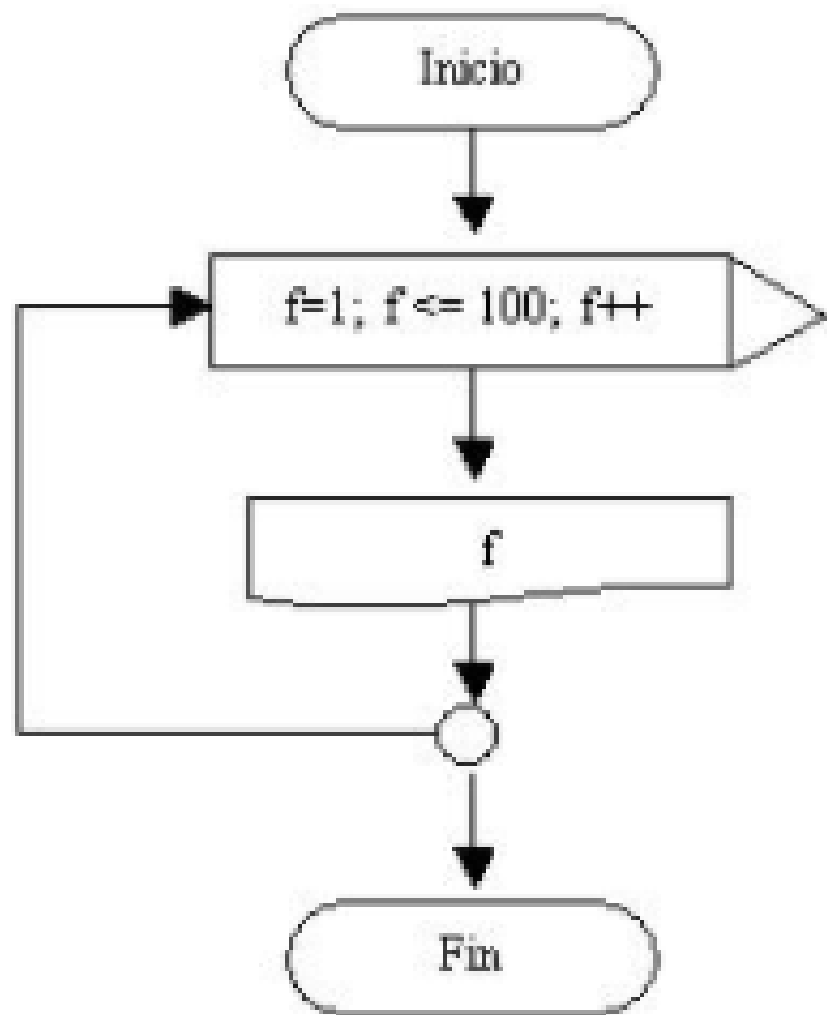
La variable `f` PUEDE ser modificada dentro del bloque de operaciones del `for`, aunque esto podría causar problemas de lógica si el programador es inexperto. La variable `f` puede ser inicializada en cualquier valor y finalizar en cualquier valor. Además, no es obligatorio que la instrucción de modificación sea un incremento del tipo contador (`f++`).

Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe `f=f+2` en lugar de `f++`, el valor de `f` será incrementado de `a` 2 en cada vuelta, y no de `a` 1. En este caso, esto significará que el ciclo no efectuará las 50 vueltas sino sólo 25.

Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

Diagrama de flujo:



Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve para contar las vueltas. Con el for el CONTADOR f cumple dicha función.

Inicialmente f vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de f, al finalizar el bloque repetitivo se incrementa la variable f en 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo.

La variable f (o como sea que se decida llamarla) debe estar definida como una variable más.

Programa:

```
public class EstructuraRepetitivaFor1 {  
    public static void main(String[] ar) {  
        int f;  
        for(f=1;f<=100;f++) {  
            System.out.print(f);  
            System.out.print("-");  
        }  
    }  
}
```

EstructuraRepetitivaFor1.java ×

```
1
2 public class EstructuraRepetitivaFor1 {
3     public static void main(String[] ar) {
4         int f;
5         for(f=1;f<=100;f++) {
6             System.out.print(f);
7             System.out.print("-");
8         }
9     }
10 }
11
```

<

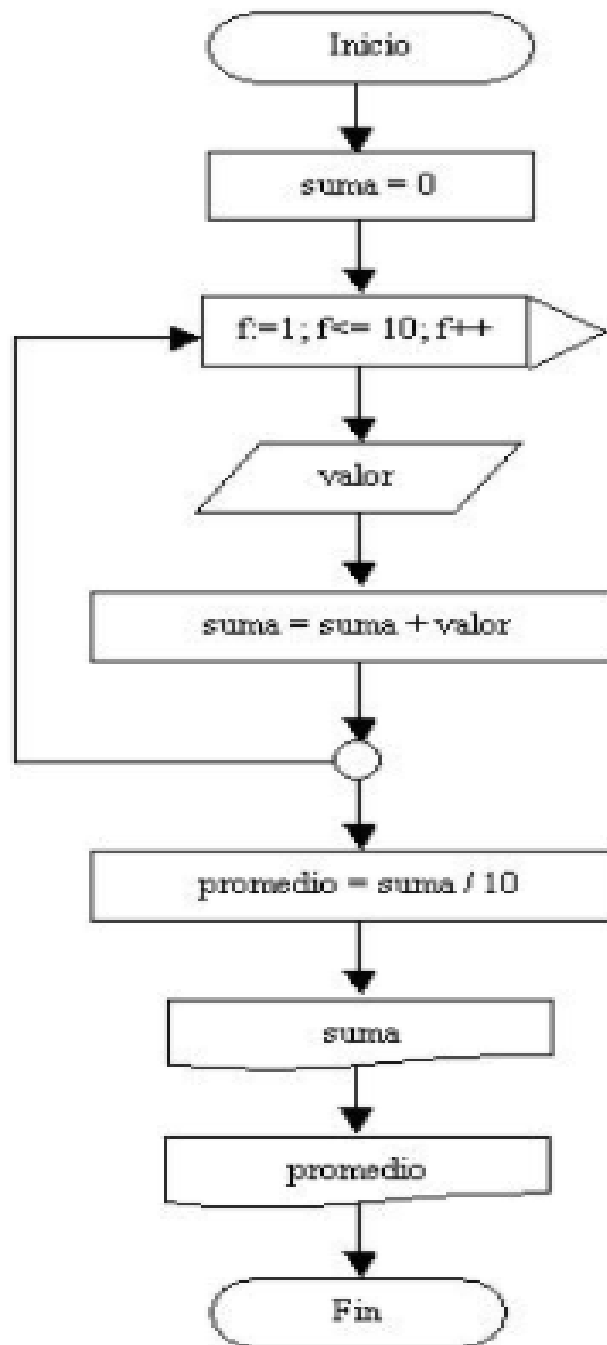
Problems @ Javadoc Declaration Console ×

<terminated> EstructuraRepetitivaFor1 [Java Application] C:\Program Files\Java\jdk-17.0.1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-

Problema 2:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores introducidos y su promedio. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for.

Diagrama de flujo:



En este caso, a la variable del for (f) sólo se la requiere para que se repita el bloque de instrucciones 10 veces.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int suma,f,valor,promedio;
        suma=0;
        for(f=1;f<=10;f++) {
            System.out.print("Introduce valor:");
            valor=teclado.nextInt();
            suma=suma+valor;
        }
        System.out.print("La suma es:");
        System.out.println(suma);
        promedio=suma/10;
        System.out.print("El promedio es:");
        System.out.print(promedio);
    }
}
```

El problema requiere que se carguen 10 valores y se sumen los mismos.

Tener en cuenta encerrar entre llaves bloque de instrucciones a repetir dentro del for.

El promedio se calcula fuera del for despues de haber cargado los 10 valores.

```
EstructuraRepetitivaFor2.java ×
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaFor2 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int suma,f,valor,promedio;
8         suma=0;
9         for(f=1;f<=10;f++) {
10             System.out.print("Introduce valor:");
11             valor=teclado.nextInt();
12             suma=suma+valor;
13         }
14         System.out.print("La suma es:");
15         System.out.println(suma);
16         promedio=suma/10;
17         System.out.print("El promedio es:");
18         System.out.print(promedio);
19     }
20 }
21
```

Problems @ Javadoc Declaration Console ×
<terminated> EstructuraRepetitivaFor2 [Java Application] C:\Program File

```
Introduce valor:7
Introduce valor:14
Introduce valor:21
Introduce valor:32
Introduce valor:88
Introduce valor:63
Introduce valor:49
Introduce valor:128
Introduce valor:236
Introduce valor:452
La suma es:1090
El promedio es:109
```

Problema 3:

Escribir un programa que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

Para resolver este problema se requieren tres contadores:

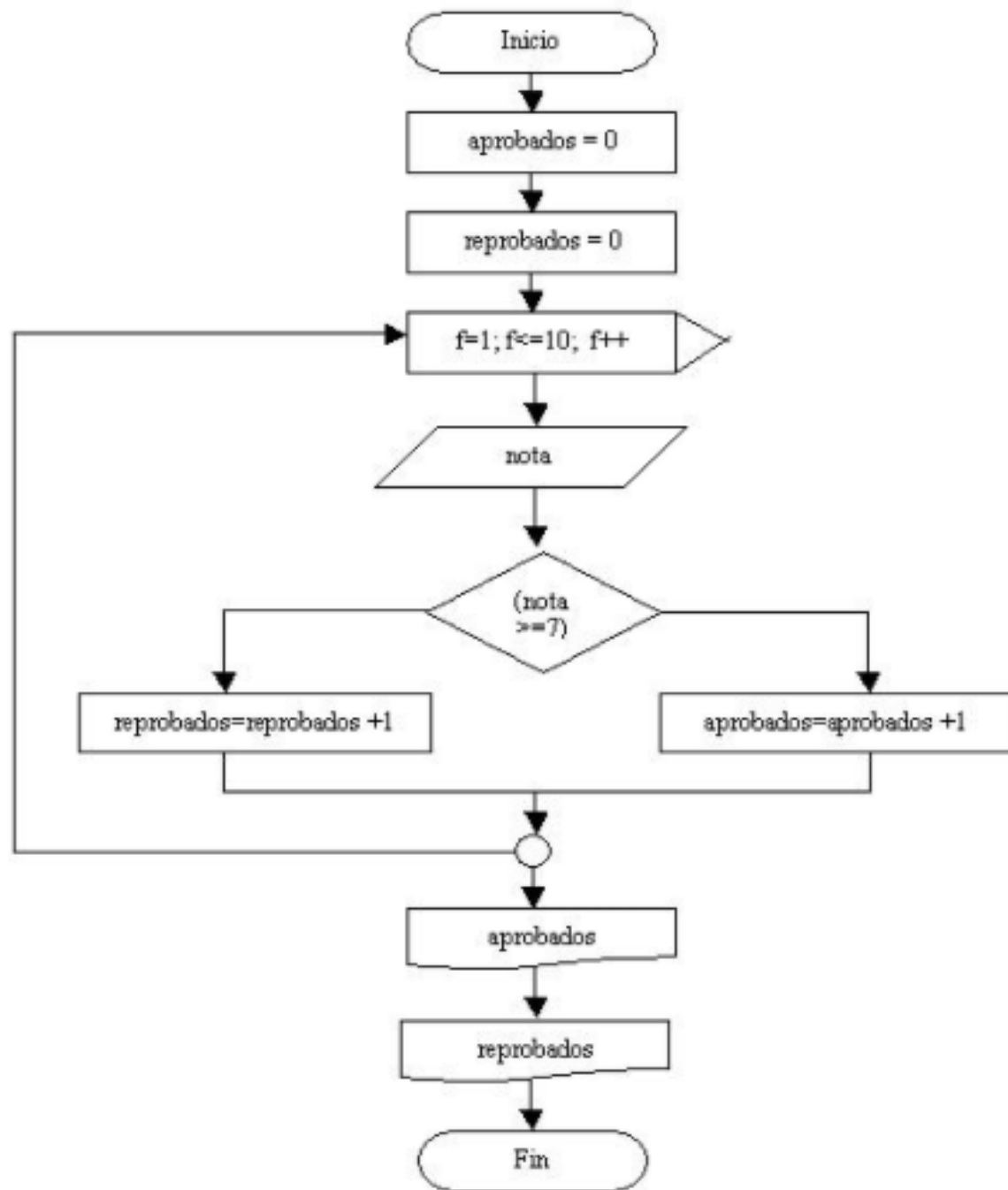
aprobados (Cuenta la cantidad de alumnos aprobados)

reprobados (Cuenta la cantidad de reprobados)

f (es el contador del for)

Dentro de la estructura repetitiva debemos hacer la carga de la variable nota y verificar con una estructura condicional si el contenido de la variable nota es mayor o igual a 7 para incrementar el contador aprobados, en caso de que la condición retorne falso debemos incrementar la variable reprobados.

Diagrama de flujo:



Los contadores aprobados y reprobados deben imprimirse FUERA de la estructura repetitiva.

Es fundamental inicializar los contadores aprobados y reprobados en cero antes de entrar a la estructura for.

Importante: Un error común es inicializar los contadores dentro de la estructura repetitiva. En caso de hacer esto los contadores se fijan en cero en cada ciclo del for, por lo que al finalizar el for como máximo el contador puede tener el valor 1.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int aprobados,reprobados,f,nota;
        aprobados=0;
        reprobados=0;
        for(f=1;f<=10;f++) {
            System.out.print("Introduce la nota:");
            nota=teclado.nextInt();
            if (nota>=7) {
                aprobados=aprobados+1;
            } else {
                reprobados=reprobados+1;
            }
        }
        System.out.print("Cantidad de aprobados:");
        System.out.println(aprobados);
        System.out.print("Cantidad de reprobados:");
        System.out.print(reprobados);
    }
}
```

```
EstructuraRepetitivaFor3.java X
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaFor3 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int aprobados, reprobados, f, nota;
8         aprobados=0;
9         reprobados=0;
10        for(f=1;f<=10;f++) {
11            System.out.print("Introduce la nota:");
12            nota=teclado.nextInt();
13            if (nota>=7) {
14                aprobados=aprobados+1;
15            } else {
16                reprobados=reprobados+1;
17            }
18        }
19        System.out.print("Cantidad de aprobados:");
20        System.out.println(aprobados);
21        System.out.print("Cantidad de reprobados:");
22        System.out.print(reprobados);
```

Problems @ Javadoc Declaration Console X

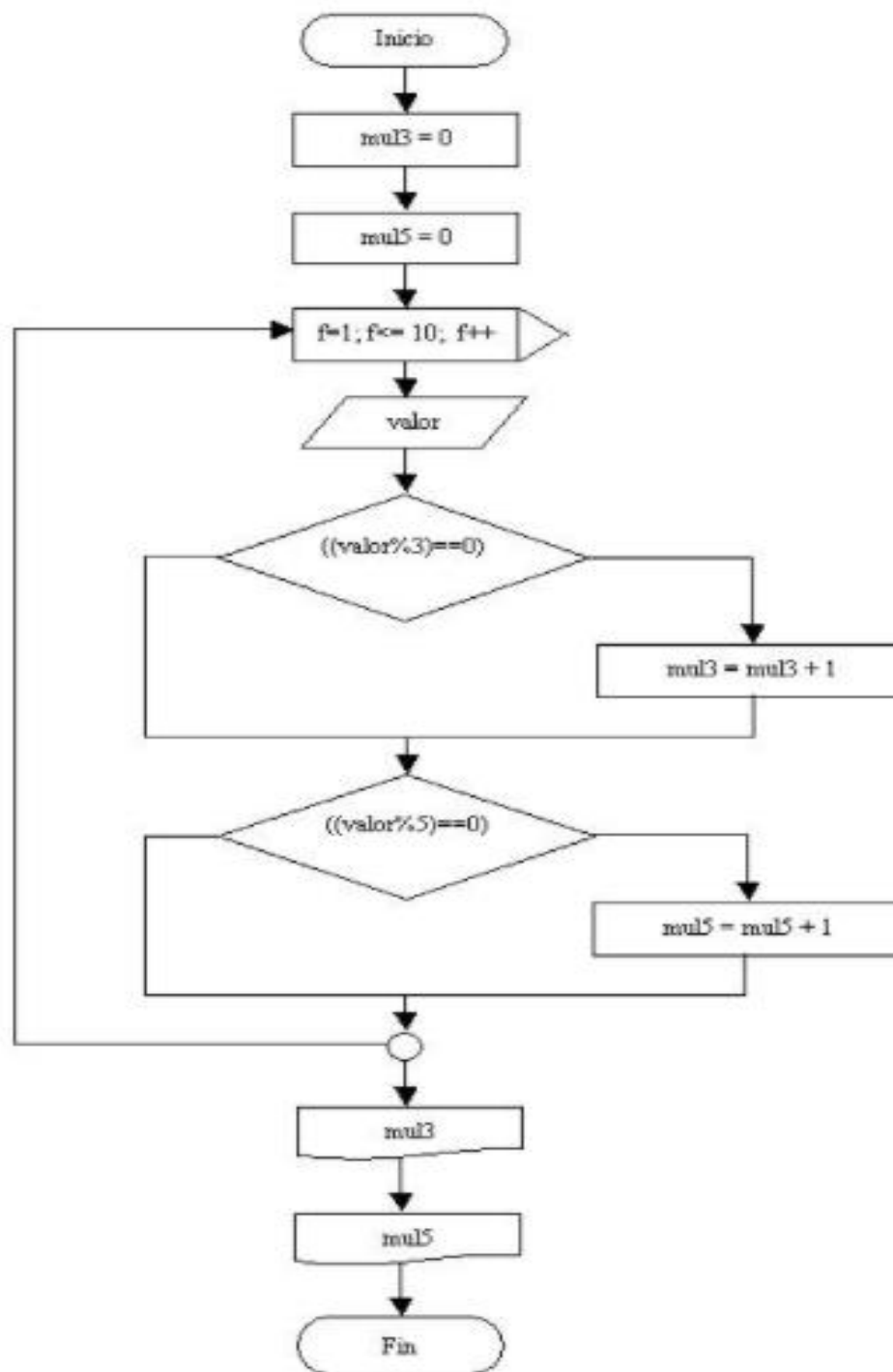
<terminated> EstructuraRepetitivaFor3 [Java Application] C:\Program Files\Ja

```
Introduce la nota:7
Introduce la nota:4
Introduce la nota:5
Introduce la nota:3
Introduce la nota:8
Introduce la nota:6
Introduce la nota:5
Introduce la nota:3
Introduce la nota:5
Introduce la nota:7
Cantidad de aprobados:3
Cantidad de reprobados:7
```

Problema 4:

Escribir un programa que lea 10 números enteros y luego muestre cuántos valores introducidos fueron múltiplos de 3 y cuántos de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y de 5 a la vez.

Diagrama de flujo:



Tengamos en cuenta que el operador matemático % retorna el resto de dividir un valor por otro, en este caso: `valor%3` retorna el resto de dividir el valor que ingresamos por teclado, por tres.

Veamos: si introducimos 6 el resto de dividirlo por 3 es 0, si ingresamos 12 el resto de dividirlo por 3 es 0. Generalizando: cuando el resto de dividir por 3 al valor que ingresamos por teclado es cero, se trata de un múltiplo de dicho valor.

Ahora bien ¿por qué no hemos dispuesto una estructura if anidada? Porque hay valores que son múltiplos de 3 y de 5 a la vez. Por lo tanto con if anidados no podríamos analizar los dos casos.

Es importante darse cuenta cuando conviene emplear if anidados y cuando no debe emplearse.

```
import java.util.Scanner;
```

Programa:

```
public class EstructuraRepetitivaFor4 {  
    public static void main(String[] ar) {  
        Scanner teclado=new Scanner(System.in);  
        int mul3,mul5,valor,f;  
        mul3=0;  
        mul5=0;  
        for(f=1;f<=10;f++) {  
            System.out.print("Ingrese un valor:");  
            valor=teclado.nextInt();  
            if (valor%3==0) {  
                mul3=mul3+1;  
            }  
            if (valor%5==0) {  
                mul5=mul5+1;  
            }  
        }  
        System.out.print("Cantidad de valores ingresados múltiplos de 3:");  
        System.out.println(mul3);  
        System.out.print("Cantidad de valores ingresados múltiplos de 5:");  
        System.out.print(mul5);  
    }  
}
```


*EstructuraRepetitivaFor4.java ×

```
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaFor4 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int mul3,mul5,valor,f;
8         mul3=0;
9         mul5=0;
10        for(f=1;f<=10;f++) {
11            System.out.print("Introduce un valor:");
12            valor=teclado.nextInt();
13            if (valor%3==0) {
14                mul3=mul3+1;
15            }
16            if (valor%5==0) {
17                mul5=mul5+1;
18            }
19        }
20        System.out.print("Cantidad de valores ingresados múltiplos de 3: ");
21        System.out.println(mul3);
22        System.out.print("Cantidad de valores ingresados múltiplos de 5: ");
```

Problems @ Javadoc Declaration Console ×

<terminated> EstructuraRepetitivaFor4 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (5 e

```
Introduce un valor:7
Introduce un valor:5
Introduce un valor:2
Introduce un valor:28
Introduce un valor:36
Introduce un valor:78
Introduce un valor:41
Introduce un valor:12
Introduce un valor:36
Introduce un valor:47
Cantidad de valores ingresados múltiplos de 3:4
Cantidad de valores ingresados múltiplos de 5:1
```

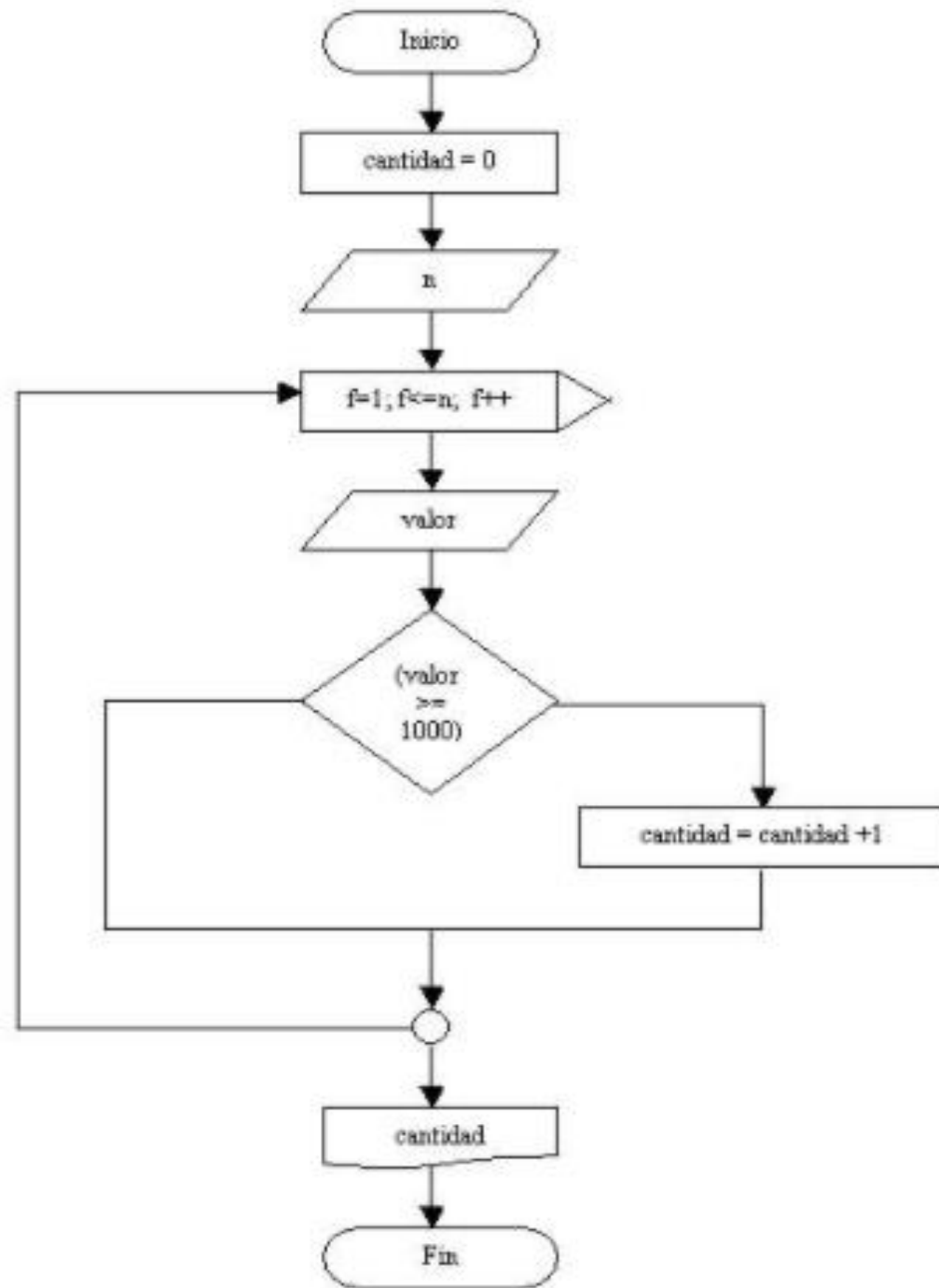
Problema 5:

Escribir un programa que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000.

Este tipo de problemas también se puede resolver empleando la estructura repetitiva for. Lo primero que se hace es cargar una variable que indique la cantidad de valores a introducir. Dicha variable se carga antes de entrar en la estructura repetitiva for.

La estructura for permite que el valor inicial o final dependa de una variable cargada previamente por teclado.

Diagrama de flujo:



Tenemos un contador llamado cantidad y f que es el contador del for.

La variable entera n se carga previo al inicio del for, por lo que podemos fijar el valor final del for con la variable n.

Por ejemplo si el operador carga 5 en n la estructura repetitiva for se ejecutará 5 veces.

La variable valor se introduce dentro de la estructura repetitiva, y se verifica si el valor de la misma es mayor o igual a 1000, en dicho caso se incrementa en uno el contador cantidad.

Fuera de la estructura repetitiva imprimimos el contador cantidad que tiene almacenado la cantidad de valores introducidos mayores o iguales a 1000.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor5 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int cantidad,n,f,valor;
        cantidad=0;
        System.out.print("Cuantos valores introduces:");
        n=teclado.nextInt();
        for(f=1;f<=n;f++) {
            System.out.print("Introduce el valor:");
            valor=teclado.nextInt();
            if (valor>=1000) {
                cantidad=cantidad+1;
            }
        }
        System.out.print("La cantidad de valores introducidos mayores o iguales a 1000 son:");
        System.out.print(cantidad);
    }
}
```

```
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaFor5 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int cantidad,n,f,valor;
8         cantidad=0;
9         System.out.print("Cuantos valores introduces:");
10        n=teclado.nextInt();
11        for(f=1;f<=n;f++) {
12            System.out.print("Introduce el valor:");
13            valor=teclado.nextInt();
14            if (valor>=1000) {
15                cantidad=cantidad+1;
16            }
17        }
18        System.out.print("La cantidad de valores ingresados mayores o iguales a 1000 son:");
19        System.out.print(cantidad);
20    }
21 }
22
```

<terminated> EstructuraRepetitivaFor5 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (5 ene 2022 1:18:26 - 1:18:26)

Cuantos valores introduces:5

Introduce el valor:256

Introduce el valor:1128

Introduce el valor:5369

Introduce el valor:32

Introduce el valor:8

La cantidad de valores ingresados mayores o iguales a 1000 son:2

Problemas propuestos

Ha llegado nuevamente la parte fundamental, que es el momento donde uno desarrolla individualmente un algoritmo para la resolución de un problema.

1. Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:

- a) De cada triángulo la medida de su base, su altura y su superficie.
- b) La cantidad de triángulos cuya superficie es mayor a 12.

2. Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores introducidos.

3. Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50)

4. Confeccionar un programa que permita introducir un valor del 1 al 10 y nos muestre la tabla de multiplicar del mismo (los primeros 12 términos)

Ejemplo: Si introduzco 3 deberá aparecer en pantalla los valores 3, 6, 9, hasta el 36.

5. Realizar un programa que lea los lados de n triángulos, e informar:

- a) De cada uno de ellos, qué tipo de triángulo es: equilátero (tres lados iguales), isósceles (dos lados iguales), o escaleno (ningún lado igual)
- b) Cantidad de triángulos de cada tipo.
- c) Tipo de triángulo que posee menor cantidad.

6. Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano.

Informar de cuántos puntos se han introducido en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que se introduzca la cantidad de puntos a procesar.

7. Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:

- a) La cantidad de valores introducidos negativos.
- b) La cantidad de valores introducidos positivos.
- c) La cantidad de múltiplos de 15.
- d) El valor acumulado de los números ingresados que son pares.

8. Se cuenta con la siguiente información:

Las edades de 50 estudiantes del turno mañana.

Las edades de 60 estudiantes del turno tarde.

Las edades de 110 estudiantes del turno noche.

Las edades de cada estudiante deben introducirse por teclado.

a) Obtener el promedio de las edades de cada turno (tres promedios)

b) Imprimir dichos promedios (promedio de cada turno)

c) Mostrar por pantalla un mensaje que indique cual de los tres turnos tiene un promedio de edades menor.

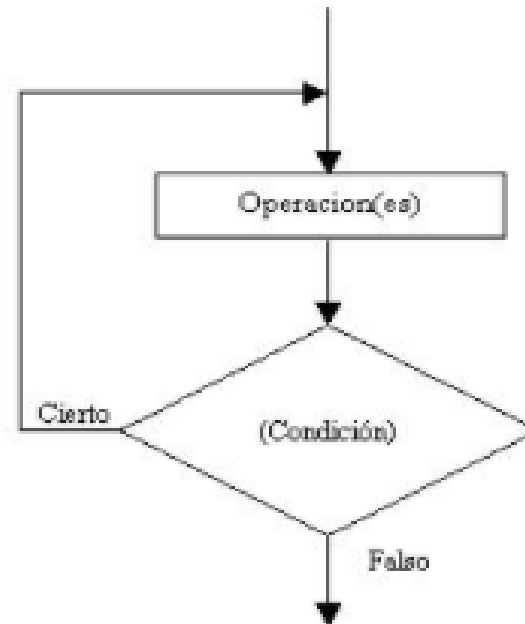
12 - Estructura repetitiva do while

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:



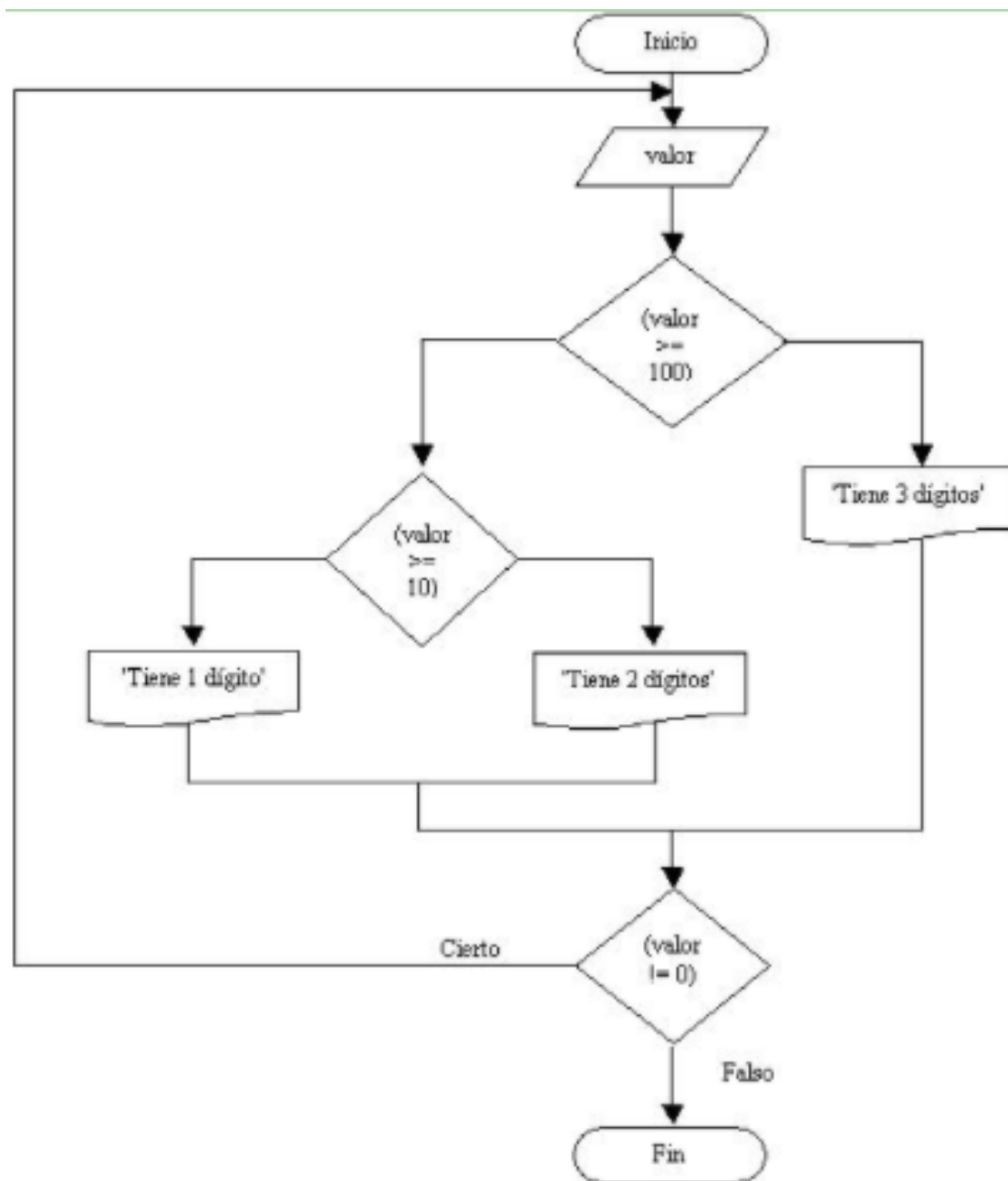
El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera. Si la condición retorna Falso el ciclo se detiene. En Java, todos los ciclos repiten por verdadero y cortan por falso.

Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

Problema 1:

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

Diagrama de flujo:



No hay que confundir los rombos de las estructuras condicionales con los de las estructuras repetitivas do while.

En este problema por lo menos se carga un valor. Si se carga un valor mayor o igual a 100 se trata de un número de tres cifras, si es mayor o igual a 10 se trata de un valor de dos dígitos, en caso contrario se trata de un valor de un dígito. Este bloque se repite hasta que se ingresa en la variable valor el número 0 con lo que la condición de la estructura do while retorna falso y sale del bloque repetitivo finalizando el programa.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaDoWhile1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int valor;
        do {
            System.out.print("Introduce un numero entre 0 y 999 (0 finaliza):");
            valor=teclado.nextInt();
            if (valor>=100) {
                System.out.println("Tiene 3 dígitos.");
            } else {
                if (valor>=10) {
                    System.out.println("Tiene 2 dígitos.");
                } else {
                    System.out.println("Tiene 1 dígito.");
                }
            }
        } while (valor!=0);
    }
}
```

```
EstructuraRepetitivaDoWhile1.java X
3
4 public class EstructuraRepetitivaDoWhile1 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int valor;
8         do {
9             System.out.print("Introduce un numero entre 0 y 999 (0 finaliza:");
10            valor=teclado.nextInt();
11            if (valor>=100) {
12                System.out.println("Tiene 3 dígitos.");
13            } else {
14                if (valor>=10) {
15                    System.out.println("Tiene 2 dígitos.");
16                } else {
17                    System.out.println("Tiene 1 dígito.");
18                }
19            }
20        } while (valor!=0);
21    }
22 }
23
```

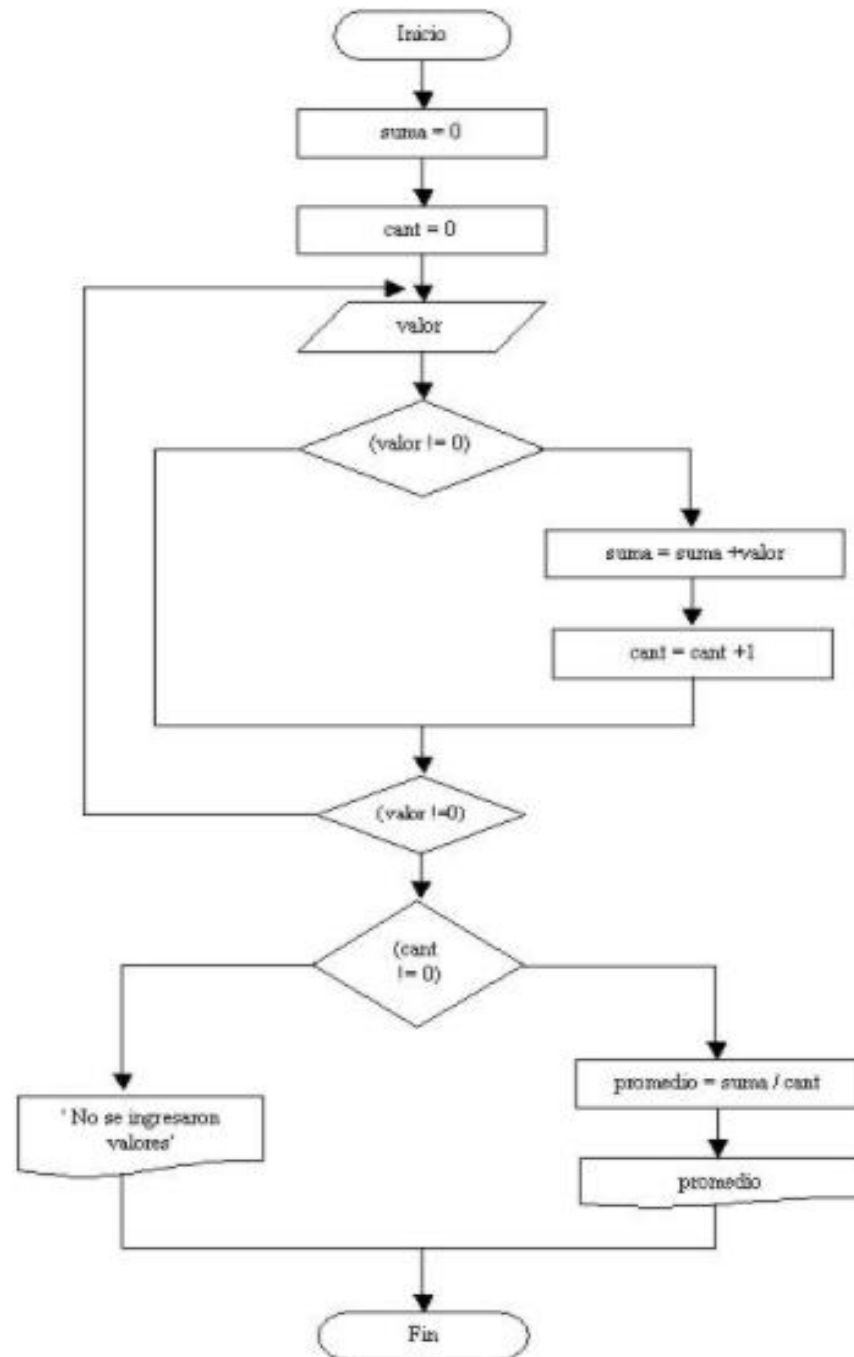
```
Problems @ Javadoc Declaration Console X
<terminated> EstructuraRepetitivaDoWhile1 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (5
Introduce un numero entre 0 y 999 (0 finaliza):185
Tiene 3 dígitos.
Introduce un numero entre 0 y 999 (0 finaliza):246
Tiene 3 dígitos.
Introduce un numero entre 0 y 999 (0 finaliza):899
Tiene 3 dígitos.
Introduce un numero entre 0 y 999 (0 finaliza):1
Tiene 1 dígito.
Introduce un numero entre 0 y 999 (0 finaliza):22
Tiene 2 dígitos.
Introduce un numero entre 0 y 999 (0 finaliza):0
Tiene 1 dígito.
```

Problema 2:

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

Cuando la finalización depende de algún valor introducido por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores)

Diagrama de flujo:



Es importante analizar este diagrama de flujo.

Definimos un contador cant que cuenta la cantidad de valores ingresados por el operador (no lo incrementa si ingresamos 0).

El valor 0 no es parte de la serie de valores que se deben sumar.

Definimos el acumulador suma que almacena todos los valores ingresados por teclado.

La estructura repetitiva do while se repite hasta que ingresamos el valor 0. Con dicho valor la condición del ciclo retorna falso y continúa con el flujo del diagrama.

Disponemos por último una estructura condicional para el caso que el operador cargue únicamente un 0 y por lo tanto no podemos calcular el promedio ya que no existe la división por 0.

En caso que el contador cant tenga un valor distinto a 0 el promedio se obtiene dividiendo el acumulador suma por el contador cant que tiene la cantidad de valores introducidos antes de introducir el 0.

Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaDoWhile2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int suma,cant,valor,promedio;
        suma=0;
        cant=0;
        do {
            System.out.print("Introduce un numero (0 para finalizar):");
            valor=teclado.nextInt();
            if (valor!=0) {
                suma=suma+valor;
                cant++;
            }
        } while (valor!=0);
        if (cant!=0) {
            promedio=suma/cant;
            System.out.print("El promedio de los numeros introducidos es:");
            System.out.print(promedio);
        } else {
            System.out.print("No se introdujeron numeros.");
        }
    }
}
```

*EstructuraRepetitivaDoWhile2.java X

```
1
2 import java.util.Scanner;
3 public class EstructuraRepetitivaDoWhile2 {
4     public static void main(String[] ar) {
5         Scanner teclado=new Scanner(System.in);
6         int suma,cant,valor,promedio;
7         suma=0;
8         cant=0;
9         do {
10             System.out.print("Introduce un numero (0 para finalizar):");
11             valor=teclado.nextInt();
12             if (valor!=0) {
13                 suma=suma+valor;
14                 cant++;
15             }
16         } while (valor!=0);
17         if (cant!=0) {
18             promedio=suma/cant;
19             System.out.print("El promedio de los numeros introducidos es:");
20             System.out.print(promedio);
21         } else {
22             System.out.print("No se introdujeron numeros.");
23         }
24     }
25 }
```

Problems @ Javadoc Declaration Console X

<terminated> EstructuraRepetitivaDoWhile2 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe

```
Introduce un numero (0 para finalizar):58
Introduce un numero (0 para finalizar):63
Introduce un numero (0 para finalizar):54
Introduce un numero (0 para finalizar):25
Introduce un numero (0 para finalizar):0
El promedio de los valores ingresados es:50
```

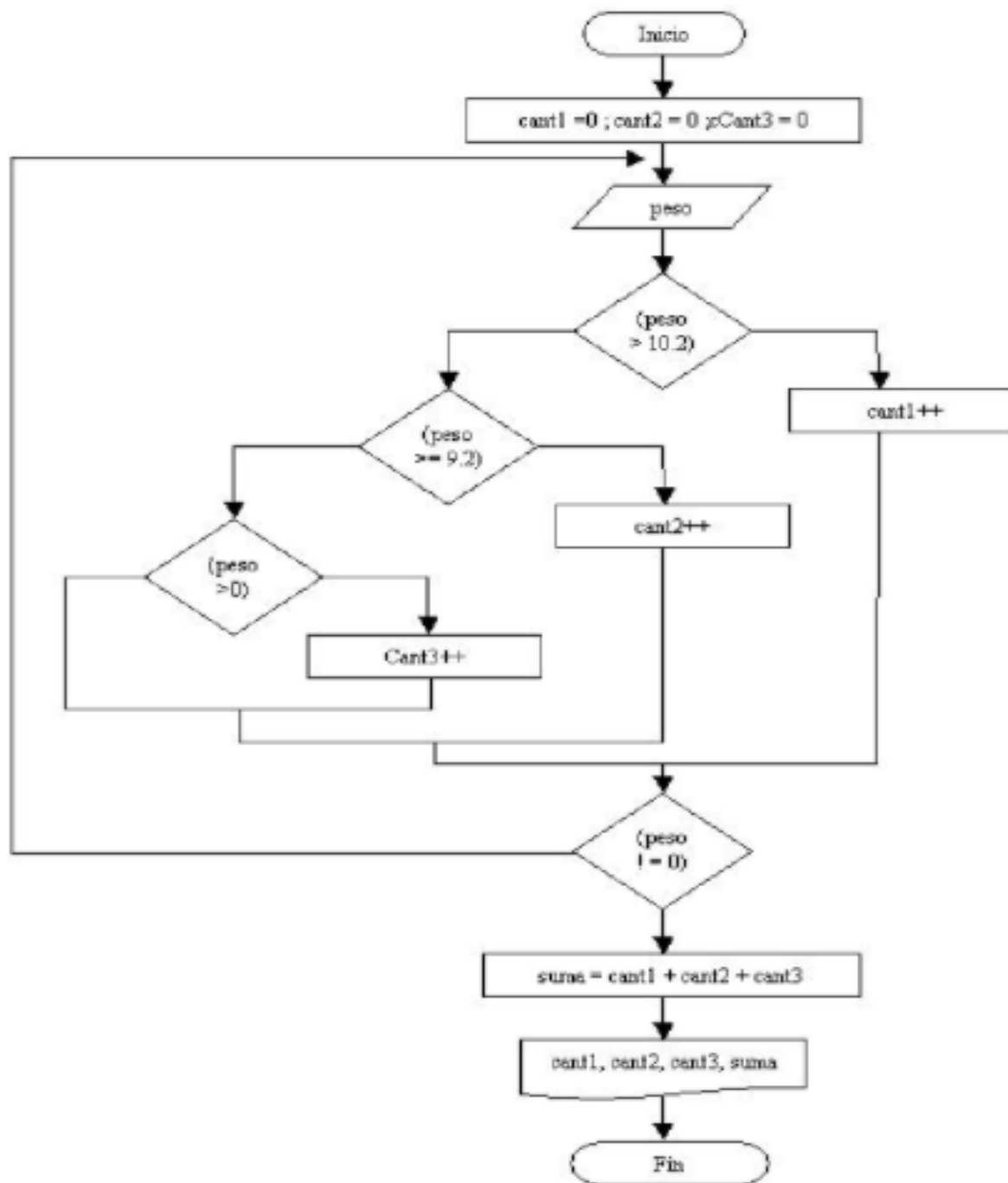
El contador cant DEBE inicializarse antes del ciclo, lo mismo que el acumulador suma. El promedio se calcula siempre y cuando el contador cant sea distinto a 0.

Problema 3:

Realizar un programa que permita introducir el peso (en kilogramos) de piezas. El proceso termina cuando introducimos el valor 0. Se debe informar:

- a) ¿Cuántas piezas tienen un peso entre 9.8 Kg. y 10.2 Kg.?, ¿cuántas con más de 10.2 Kg.? y ¿cuántas con menos de 9.8 Kg.?
- b) La cantidad total de piezas procesadas.

Diagrama de flujo:



Los tres contadores cont1, cont2, y cont3 se inicializan en 0 antes de entrar a la estructura repetitiva.

A la variable suma no se la inicializa en 0 porque no es un acumulador, sino que guarda la suma del contenido de las variables cont1, cont2 y cont3.

La estructura se repite hasta que se ingresa el valor 0 en la variable peso. Este valor no se lo considera un peso menor a 9.8 Kg., sino que indica que ha finalizado la carga de valores por teclado.

```
import java.util.Scanner;
```

Programa:

```
public class EstructuraRepetitivaDoWhile3 {  
    public static void main(String[] ar) {  
        Scanner teclado=new Scanner(System.in);  
        int cant1,cant2,cant3,suma;  
        float peso;  
        cant1=0;  
        cant2=0;  
        cant3=0;  
        do {  
            System.out.print("Introduce el peso de la pieza (0 pera finalizar):");  
            peso=teclado.nextFloat();  
            if (peso>10.2) {  
                cant1++;  
            } else {  
                if (peso>=9.8) {  
                    cant2++;  
                } else {  
                    if (peso>0) {  
                        cant3++;  
                    }  
                }  
            }  
        }  
    }  
}
```



```
    } while(peso!=0);  
    suma=cant1+cant2+cant3;  
    System.out.print("Piezas aptas:");  
    System.out.println(cant2);  
    System.out.print("Piezas con un peso superior a 10.2:");  
    System.out.println(cant1);  
    System.out.print("Piezas con un peso inferior a 9.8:");  
    System.out.println(cant3);  
    }  
}
```

```
1
2 import java.util.Scanner;
3
4 public class EstructuraRepetitivaDoWhile3 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         int cant1,cant2,cant3,suma;
8         float peso;
9         cant1=0;
10        cant2=0;
11        cant3=0;
12        do {
13            System.out.print("Introduce el peso de la pieza (0 pera finalizar):");
14            peso=teclado.nextFloat();
15            if (peso>10.2) {
16                cant1++;
17            } else {
18                if (peso>=9.8) {
19                    cant2++;
20                } else {
21                    if (peso>0) {
22                        cant3++;
23                    }
24                }
25            }
26        } while(peso!=0);
27        suma=cant1+cant2+cant3;
```

<terminated> EstructuraRepetitivaDoWhile3 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (5 ene

```
Introduce el peso de la pieza (0 pera finalizar):10
Introduce el peso de la pieza (0 pera finalizar):12
Introduce el peso de la pieza (0 pera finalizar):9,8
Introduce el peso de la pieza (0 pera finalizar):0
Piezas aptas:2
Piezas con un peso superior a 10.2:1
Piezas con un peso inferior a 9.8:0
```

Problemas propuestos

1.Realizar un programa que acumule (sume) valores ingresados por teclado hasta introducir el 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.

2.En un banco se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: número de cuenta y saldo actual. El ingreso de datos debe finalizar al introducir un valor negativo en el número de cuenta. Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:

a)De cada cuenta: número de cuenta y estado de la cuenta según su saldo, sabiendo que:

| | |
|---------------------|----------------------------------|
| Estado de la cuenta | 'Acreedor' si el saldo es >0 . |
| | 'Deudor' si el saldo es <0 . |
| | 'Nulo' si el saldo es $=0$. |

b) La suma total de los saldos acreedores.

13 - Cadenas de caracteres en Java

En Java hemos visto que cuando queremos almacenar un valor entero definimos una variable de tipo `int`, si queremos almacenar un valor con decimales definimos una variable de tipo `float`. Ahora si queremos almacenar una cadena de caracteres (por ejemplo un nombre de una persona) debemos definir **un objeto de la clase `String`**.

Más adelante veremos en profundidad y detenimiento los conceptos de CLASE y OBJETO, por ahora solo nos interesa la mecánica para trabajar con cadenas de caracteres.

Problema 1:

Solicitar el nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad.

Programa:

```
import java.util.Scanner;

public class CadenaDeCaracteres1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String nombre1,nombre2;
        int edad1,edad2;
        System.out.print("Introduce el nombre:");
        nombre1=teclado.next();
        System.out.print("Introduce edad:");
        edad1=teclado.nextInt();
        System.out.print("Introduce el nombre:");
        nombre2=teclado.next();
        System.out.print("Introduce edad:");
        edad2=teclado.nextInt();
        System.out.print("La persona de mayor edad es:");
        if (edad1>edad2) {
            System.out.print(nombre1);
        } else {
            System.out.print(nombre2);
        }
    }
}
```

```
*CadenaDeCaracteres1.java X
2 import java.util.Scanner;
3
4 public class CadenaDeCaracteres1 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         String nombre1,nombre2;
8         int edad1,edad2;
9         System.out.print("Introduce el nombre:");
10        nombre1=teclado.next();
11        System.out.print("Introduce edad:");
12        edad1=teclado.nextInt();
13        System.out.print("Introduce el nombre:");
14        nombre2=teclado.next();
15        System.out.print("Introduce edad:");
16        edad2=teclado.nextInt();
17        System.out.print("La persona de mayor edad es:");
18        if (edad1>edad2) {
19            System.out.print(nombre1);
20        } else {
21            System.out.print(nombre2);
22        }
23    }
24 }
25
```

Problems @ Javadoc Declaration Console X

<terminated> CadenaDeCaracteres1 [Java Application] C:\Program Files\Java\jdk-17

Introduce el nombre:juan
Introduce edad:45
Introduce el nombre:Maria
Introduce edad:28
La persona de mayor edad es:juan

Para almacenar un nombre debemos definir una **variable de tipo String** y su **introducción por teclado** se hace llamando al **método next()** del objeto teclado:

```
nombre1=teclado.next();
```

La primera salvedad que tenemos que hacer cuando utilizamos el método next() es que **solo** nos **permite introducir una cadena de caracteres con la excepción del espacio en blanco** (es decir debemos introducir un nombre de persona y **no** su nombre y apellido separado por un espacio en blanco)

Veamos que existe otro método llamado **nextLine()** que nos **permite cargar espacios en blanco** pero para su uso se complica cuando cargamos otros valores de tipo distinto a String (por ejemplo int, float etc.)

Problema 2:

Solicitar el apellido, nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad. Realizar la carga del apellido y nombre en una variable de tipo String.

```
import java.util.Scanner;
```

Programa:

```
public class CadenaDeCaracteres2 {  
    public static void main(String[] ar) {  
        Scanner teclado=new Scanner(System.in);  
        String apenom1,apenom2;  
        int edad1,edad2;  
        System.out.print("Introduce el apellido y el nombre:");  
        apenom1=teclado.nextLine();  
        System.out.print("Introduce edad:");  
        edad1=teclado.nextInt();  
        System.out.print("Introduce el apellido y el nombre:");  
        teclado.nextLine();  
        apenom2=teclado.nextLine();  
        System.out.print("Introduce edad:");  
        edad2=teclado.nextInt();  
        System.out.print("La persona de mayor edad es:");  
        if (edad1>edad2) {  
            System.out.print(apenom1);  
        } else {  
            System.out.print(apenom2);  
        }  
    }  
}
```


*CadenaDeCaracteres2.java X

```
3
4 public class CadenaDeCaracteres2 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         String apenom1,apenom2;
8         int edad1,edad2;
9         System.out.print("Introduce el apellido y el nombre:");
10        apenom1=teclado.nextLine();
11        System.out.print("Introduce edad:");
12        edad1=teclado.nextInt();
13        System.out.print("Introduce el apellido y el nombre:");
14        teclado.nextLine();
15        apenom2=teclado.nextLine();
16        System.out.print("Introduce edad:");
17        edad2=teclado.nextInt();
18        System.out.print("La persona de mayor edad es:");
19        if (edad1>edad2) {
20            System.out.print(apenom1);
21        } else {
22            System.out.print(apenom2);
23        }
24    }
25 }
26
```

Problems @ Javadoc Declaration Console X

<terminated> CadenaDeCaracteres2 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin

Introduce el apellido y el nombre:Garcia Luis

Introduce edad:35

Introduce el apellido y el nombre:Gomez Laura

Introduce edad:25

La persona de mayor edad es:Garcia Luis

Cuando se introduce una **cadena con caracteres en blanco** debemos tener en cuenta en llamar al **método nextLine()**

Existe una dificultad si llamamos al método nextLine() y previamente hemos llamado al método nextInt(), esto es debido a que después de ejecutar el método nextInt() queda almacenado en el objeto de la clase Scanner el carácter "Enter" y si llamamos inmediatamente al método nextLine() este almacena dicho valor de tecla y continúa con el flujo del programa. Para solucionar este problema debemos generar un código similar a:

```
System.out.print("Introduce edad:");  
edad1=teclado.nextInt();  
System.out.print("Introduce el apellido y el nombre:");  
teclado.nextLine();  
apenom2=teclado.nextLine();
```

Como vemos **llamamos al método nextLine() dos veces, la primera retorna la tecla "Enter" y la segunda se queda esperando que introduzcamos el apellido y nombre** (tener en cuenta que esto es necesario solo si previamente se llamó al método nextInt() o nextFloat()).

Problema 3:

Solicitar dos apellidos. Mostrar un mensaje si son iguales o distintos.

Programa:

```
import java.util.Scanner;

public class CadenaDeCaracteres3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String apellido1,apellido2;
        System.out.print("Introduce el primer apellido:");
        apellido1=teclado.next();
        System.out.print("Introduce el segundo apellido:");
        apellido2=teclado.next();
        if (apellido1.equals(apellido2)) {
            System.out.print("Los apellidos son iguales");
        } else {
            System.out.print("Los apellidos son distintos");
        }
    }
}
```

```
1
2 import java.util.Scanner;
3
4 public class CadenaDeCaracteres3 {
5     public static void main(String[] ar) {
6         Scanner teclado=new Scanner(System.in);
7         String apellido1,apellido2;
8         System.out.print("Introduce el primer apellido:");
9         apellido1=teclado.next();
10        System.out.print("Introduce el segundo apellido:");
11        apellido2=teclado.next();
12        if (apellido1.equals(apellido2)) {
13            System.out.print("Los apellidos son iguales");
14        } else {
15            System.out.print("Los apellidos son distintos");
16        }
17    }
18 }
```

<terminated> CadenaDeCaracteres3 [Java Application] C:\Program Files\Java\jdk-17.0.1\

Introduce el primer apellido:Garcia

Introduce el segundo apellido:Perez

Los apellidos son distintos

Para comparar si el contenido de dos String son iguales no podemos utilizar el operador `==`. **Debemos utilizar un método de la clase String llamado equals** y pasar como parámetro el String con el que queremos compararlo:

```
if (apellido1.equals(apellido2)) {
```

El método equals retorna verdadero si los contenidos de los dos String son exactamente iguales, esto hace que se ejecute el bloque del verdadero.

Recordemos que hemos utilizado el método `next()` para la carga de los String, luego esto hace que no podamos introducir un apellido con espacios en blanco (podemos probar que si introducimos por ejemplo "Rodriguez Rodriguez" en el primer apellido, se carga la cadena "Rodriguez" en la variable `apellido1` y "Rodriguez" en la variable `apellido2` (con esto hacemos notar que cada vez que introducimos un espacio en blanco cuando utilizamos el método `next()` los caracteres que siguen al espacio en blanco son recuperados en la siguiente llamada al método `next()`)

El método `equals` retorna verdadero si los contenidos de los dos String son exactamente iguales, es decir si cargamos "Martinez" en `apellido1` y "martinez" en `apellido2` luego el método `equals` retorna falso ya que no es lo mismo la "M" mayúscula y la "m" minúscula.

En el caso que necesitemos considerar igual caracteres mayúsculas y minúsculas podemos utilizar el método equalsIgnoreCase:

```
if (apellido1.equalsIgnoreCase(apellido2)) {  
    System.out.print("Los apellidos son iguales sin tener en cuenta mayúsculas y  
    minúsculas");  
} else {  
    System.out.print("Los apellidos son distintos sin tener en cuenta mayúsculas y  
    minúsculas");  
}
```

14 - Declaración de una clase y definición de objetos.

La **programación orientada a objetos** se basa en la **programación de clases**; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones).

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

La estructura de una clase es:

```
class [nombre de la clase] {  
    [atributos o variables de la clase]  
    [métodos o funciones de la clase]  
    [main]  
}
```

Problema 1:

Confeccionar una clase que permita carga el nombre y la edad de una persona. Mostrar los datos cargados. Imprimir un mensaje si es mayor de edad (edad \geq 18)

Programa:

```
import java.util.Scanner;
public class Persona {
    private Scanner teclado;
    private String nombre;
    private int edad;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Introduce nombre:");
        nombre=teclado.next();
        System.out.print("Introduce edad:");
        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:"+nombre);
        System.out.println("Edad:"+edad);
    }
}
```

```
public void esMayorEdad() {  
    if (edad >= 18) {  
        System.out.print(nombre + " es mayor de edad.");  
    } else {  
        System.out.print(nombre + " no es mayor de edad.");  
    }  
}
```

```
public static void main(String[] ar) {  
    Persona persona1;  
    persona1 = new Persona();  
    persona1.inicializar();  
    persona1.imprimir();  
    persona1.esMayorEdad();  
}
```

```
1
2 import java.util.Scanner;
3 public class Persona {
4     private Scanner teclado;
5     private String nombre;
6     private int edad;
7
8     public void inicializar() {
9         teclado=new Scanner(System.in);
10        System.out.print("Introduce nombre:");
11        nombre=teclado.next();
12        System.out.print("Introduce edad:");
13        edad=teclado.nextInt();
14    }
15
16    public void imprimir() {
17        System.out.println("Nombre:"+nombre);
18        System.out.println("Edad:"+edad);
19    }
20
21    public void esMayorEdad() {
22        if (edad>=18) {
23            System.out.print(nombre+" es mayor de edad.");
24        } else {
25            System.out.print(nombre+" no es mayor de edad.");
26        }
27    }
28
29    public static void main(String[] ar) {
```

<terminated> Persona [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (

Introduce nombre:laura

Introduce edad:31

Nombre:laura

Edad:31

laura es mayor de edad.

El nombre de la clase debe hacer referencia al concepto (en este caso la hemos llamado Persona):

```
public class Persona {
```

Los atributos los definimos dentro de la clase pero fuera de la main:

```
    private Scanner teclado;
```

```
    private String nombre;
```

```
    private int edad;
```

Veremos más adelante que un atributo es normalmente definido con la cláusula **private** (con esto no permitimos el acceso al atributo desde otras clases)

A los atributos se tiene acceso desde cualquier función o método de la clase (salvo la main)

Después de definir los atributos de la clase debemos declarar los métodos o funciones de la clase. La sintaxis es parecida a la main (sin la cláusula static):

```
public void inicializar() {  
    teclado=new Scanner(System.in);  
    System.out.print("Introduce nombre:");  
    nombre=teclado.next();  
    System.out.print("Introduce edad:");  
    edad=teclado.nextInt();  
}
```

En el método inicializar (que será el primero que deberemos llamar desde la main) creamos el objeto de la clase Scanner y cargamos por teclado los atributos nombre y edad. Como podemos ver el método inicializar puede hacer acceso a los tres atributos de la clase Persona.

El segundo método tiene por objetivo imprimir el contenido de los atributos nombre y edad (los datos de los atributos se cargaron al ejecutarse previamente el método inicializar:

```
public void imprimir() {  
    System.out.println("Nombre:"+nombre);  
    System.out.println("Edad:"+edad);  
}
```

El tercer método tiene por objetivo mostrar un mensaje si la persona es mayor o no de edad:

```
public void esMayorEdad() {  
    if (edad>=18) {  
        System.out.print(nombre+" es mayor de edad.");  
    } else {  
        System.out.print(nombre+" no es mayor de edad.");  
    }  
}
```

Por último en la main declaramos un objeto de la clase Persona y llamamos a los métodos en un orden adecuado:

```
public static void main(String[] ar) {  
    Persona persona1;  
    persona1=new Persona();  
    persona1.inicializar();  
    persona1.imprimir();  
    persona1.esMayorEdad();  
}
```

Persona persona1; //Declaración del objeto
persona1=new Persona(); //Creación del objeto
persona1.inicializar(); //Llamada de un método

Problema 2:

Desarrollar un programa que cargue los lados de un triángulo e implemente los siguientes métodos: inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no.

Programa:

```
import java.util.Scanner;
public class Triangulo {
    private Scanner teclado;
    private int lado1,lado2,lado3;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Medida lado 1:");
        lado1=teclado.nextInt();
        System.out.print("Medida lado 2:");
        lado2=teclado.nextInt();
        System.out.print("Medida lado 3:");
        lado3=teclado.nextInt();
    }

    public void ladoMayor() {
        System.out.print("Lado mayor:");
        if (lado1>lado2 && lado1>lado3) {
            System.out.println(lado1);
        } else {
            if (lado2>lado3) {
                System.out.println(lado2);
            } else {
                System.out.println(lado3);
            }
        }
    }
}
```



```
    }  
    }  
}  
public void esEquilatero() {  
    if (lado1==lado2 && lado1==lado3) {  
        System.out.print("Es un triángulo equilátero");  
    } else {  
        System.out.print("No es un triángulo equilátero");  
    }  
}  
  
public static void main(String []ar) {  
    Triangulo triangulo1=new Triangulo();  
    triangulo1.inicializar();  
    triangulo1.ladoMayor();  
    triangulo1.esEquilatero();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class Triangulo {
4     private Scanner teclado;
5     private int lado1,lado2,lado3;
6
7     public void inicializar() {
8         teclado=new Scanner(System.in);
9         System.out.print("Medida lado 1:");
10        lado1=teclado.nextInt();
11        System.out.print("Medida lado 2:");
12        lado2=teclado.nextInt();
13        System.out.print("Medida lado 3:");
14        lado3=teclado.nextInt();
15    }
16
17    public void ladoMayor() {
18        System.out.print("Lado mayor:");
19        if (lado1>lado2 && lado1>lado3) {
20            System.out.println(lado1);
21        } else {
22            if (lado2>lado3) {
23                System.out.println(lado2);
24            } else {
25                System.out.println(lado3);
26            }
27        }
28    }
29
```

<terminated> Triangulo [Java Application] C:\Program Files\Java\jdk1

Medida lado 1:25

Medida lado 2:32

Medida lado 3:47

Lado mayor:47

No es un triángulo equilátero

Todos los problemas que requieran la entrada de datos por teclado debemos definir un atributo de la clase Scanner:

```
private Scanner teclado;
```

Este problema requiere definir tres atributos de tipo entero donde almacenamos los valores de los lados del triángulo:

```
private int lado1,lado2,lado3;
```

El primer método que deberá llamarse desde la main es el inicializar donde creamos el objeto de la clase Scanner y cargamos los tres atributos por teclado:

```
public void inicializar() {  
    teclado=new Scanner(System.in);  
    System.out.print("Medida lado 1:");  
    lado1=teclado.nextInt();  
    System.out.print("Medida lado 2:");  
    lado2=teclado.nextInt();  
    System.out.print("Medida lado 3:");  
    lado3=teclado.nextInt();  
}
```

El método ladoMayor muestra el valor mayor de los tres enteros introducidos:

```
public void ladoMayor() {  
    System.out.print("Lado mayor:");  
    if (lado1>lado2 && lado1>lado3) {  
        System.out.println(lado1);  
    } else {  
        if (lado2>lado3) {  
            System.out.println(lado2);  
        } else {  
            System.out.println(lado3);  
        }  
    }  
}
```

Como podemos observar cuando un problema se vuelve más complejo es más fácil y ordenado separar los distintos algoritmos en varios métodos y no codificar todo en la main.

El último método de esta clase verifica si los tres enteros introducidos son iguales:

```
public void esEquilatero() {  
    if (lado1==lado2 && lado1==lado3) {  
        System.out.print("Es un triángulo equilátero");  
    } else {  
        System.out.print("No es un triángulo equilátero");  
    }  
}
```

En la main creamos un objeto de la clase Triangulo y llamamos los métodos respectivos:

```
public static void main(String []ar) {  
    Triangulo triangulo1=new Triangulo();  
    triangulo1.inicializar();  
    triangulo1.ladoMayor();  
    triangulo1.esEquilatero();  
}
```

Problema 3:

Desarrollar una clase que represente un punto en el plano y tenga los siguientes métodos: cargar los valores de x e y, imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante si x e y son positivas, si $x < 0$ e $y > 0$ segundo cuadrante, etc.)

Programa:

```
import java.util.Scanner;

public class Punto {
    private Scanner teclado;
    int x,y;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Introduce coordenada x :");
        x=teclado.nextInt();
        System.out.print("Introduce coordenada y :");
        y=teclado.nextInt();
    }

    void imprimirCuadrante() {
        if (x>0 && y>0) {
            System.out.print("Se encuentra en el primer cuadrante.");
        } else {
            if (x<0 && y>0) {
                System.out.print("Se encuentra en el segundo cuadrante.");
            } else {
                if (x<0 && y<0) {
                    System.out.print("Se encuentra en el tercer cuadrante.");
                } else {
```

```
        if (x>0 && y<0) {  
            System.out.print("Se encuentra en el cuarto cuadrante.");  
        } else {  
            System.out.print("El punto no está en un cuadrante.");  
        }  
    }  
}  
}
```

```
public static void main(String[] ar) {  
    Punto punto1;  
    punto1=new Punto();  
    punto1.inicializar();  
    punto1.imprimirCuadrante();  
}  
}
```



```
1
2 import java.util.Scanner;
3 public class Punto {
4     private Scanner teclado;
5     int x,y;
6
7     public void inicializar() {
8         teclado=new Scanner(System.in);
9         System.out.print("Introduce coordenada x :");
10        x=teclado.nextInt();
11        System.out.print("Introduce coordenada y :");
12        y=teclado.nextInt();
13    }
14
15    void imprimirCuadrante() {
16        if (x>0 && y>0) {
17            System.out.print("Se encuentra en el primer cuadrante.");
18        } else {
19            if (x<0 && y>0) {
20                System.out.print("Se encuentra en el segundo cuadrante.");
21            } else {
22                if (x<0 && y<0) {
23                    System.out.print("Se encuentra en el tercer cuadrante.");
24                } else {
25                    if (x>0 && y<0) {
26                        System.out.print("Se encuentra en el cuarto cuadrante.");
27                    } else {
28                        System.out.print("El punto no está en un cuadrante.");
29                    }
30                }
31            }
32        }
33    }
34 }
```

Definimos tres atributos (el objeto de la clase Scanner y los dos enteros donde almacenamos las coordenadas x e y del punto:

```
private Scanner teclado;  
int x,y;
```

El método inicializar crea el objeto de la clase Scanner y pide cargar las coordenadas x e y:

```
public void inicializar() {  
    teclado = new Scanner(System.in);  
    System.out.print("Introduce coordenada x :");  
    x = teclado.nextInt();  
    System.out.print("Introduce coordenada y :");  
    y = teclado.nextInt();  
}
```

El segundo método mediante un conjunto de if verificamos en que cuadrante se encuentra el punto introducido:

```
void imprimirCuadrante() {  
    if (x>0 && y>0) {  
        System.out.print("Se encuentra en el primer cuadrante.");  
    } else {  
        if (x<0 && y>0) {  
            System.out.print("Se encuentra en el segundo cuadrante.");  
        } else {  
            if (x<0 && y<0) {  
                System.out.print("Se encuentra en el tercer cuadrante.");  
            } else {  
                if (x>0 && y<0) {  
                    System.out.print("Se encuentra en el cuarto cuadrante.");  
                } else {  
                    System.out.print("El punto no está en un cuadrante.");  
                }  
            }  
        }  
    }  
}
```

El main no tiene grandes diferencias con los problemas realizados anteriormente, declaramos un objeto de la clase Punto, creamos el objeto mediante el operador new y seguidamente llamamos a los métodos inicializar e imprimirCuadrante en ese orden:

```
public static void main(String[] ar) {  
    Punto punto1;  
    punto1=new Punto();  
    punto1.inicializar();  
    punto1.imprimirCuadrante();  
}
```

Problema 4:

Desarrollar una clase que represente un Cuadrado y tenga los siguientes métodos: cargar el valor de su lado, imprimir su perímetro y su superficie.

Programa:

```
import java.util.Scanner;

public class Cuadrado {
    private Scanner teclado;
    int lado;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Introduce valor del lado :");
        lado=teclado.nextInt();
    }

    public void imprimirPerimetro() {
        int perimetro;
        perimetro=lado*4;
        System.out.println("El perímetro es:"+perimetro);
    }

    public void imprimirSuperficie() {
        int superficie;
        superficie=lado*lado;
        System.out.println("La superficie es:"+superficie);
    }
}
```

```
public static void main(String[] ar) {  
    Cuadrado cuadrado1;  
    cuadrado1=new Cuadrado();  
    cuadrado1.inicializar();  
    cuadrado1.imprimirPerimetro();  
    cuadrado1.imprimirSuperficie();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class Cuadrado {
4     private Scanner teclado;
5     int lado;
6
7     public void inicializar() {
8         teclado=new Scanner(System.in);
9         System.out.print("Introduce valor del lado :");
10        lado=teclado.nextInt();
11    }
12
13    public void imprimirPerimetro() {
14        int perimetro;
15        perimetro=lado*4;
16        System.out.println("El perímetro es:"+perimetro);
17    }
18
19    public void imprimirSuperficie() {
20        int superficie;
21        superficie=lado*lado;
22        System.out.println("La superficie es:"+superficie);
23    }
24
25    public static void main(String[] ar) {
26        Cuadrado cuadrado1;
27        cuadrado1=new Cuadrado();
28        cuadrado1.inicializar();
29        cuadrado1.imprimirPerimetro();
30        cuadrado1.imprimirSuperficie();
31    }
```

En este problema es interesante ver como no definimos dos atributos donde se almacenan la superficie y el perímetro del cuadrado, esto es debido a que solo estos datos son requeridos en el método donde se imprimen:

```
public void imprimirPerimetro() {  
    int perimetro;  
    perimetro=lado*4;  
    System.out.println("El perímetro es:"+perimetro);  
}
```

Esto significa que la variable perímetro es una variable local al método imprimirPerimetro. Esta variable es local a dicho método y solo se la puede acceder dentro del método. La diferencia fundamental entre una variable local y un atributo de la clase es que al atributo se le puede acceder desde cualquier método de la clase y la variable local solo existe mientras se está ejecutando el método.

Problemas propuestos

1. Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. Confeccionar los métodos para la carga, otro para imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)
2. Implementar la clase operaciones. Se deben cargar dos valores enteros, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

15 - Declaración de métodos.

Cuando uno plantea una clase en lugar de especificar todo el algoritmo en un único método (lo que hicimos en los primeros pasos de este tutorial) es dividir todas las responsabilidades de la clase en un conjunto de métodos.

Un método hemos visto que tiene la siguiente sintaxis:

```
public void [nombre del método]() {  
    [algoritmo]  
}
```

Veremos que hay varios tipos de métodos:

Métodos con parámetros.

Un método puede tener parámetros:

```
public void [nombre del método]([parámetros]) {  
    [algoritmo]  
}
```

Los parámetros los podemos imaginar como variables locales al método, pero su valor se inicializa con datos que llegan cuando lo llamamos.

Problema 1:

Confeccionar una clase que permita introducir valores enteros por teclado y nos muestre la tabla de multiplicar de dicho valor. Finalizar el programa al introducir el -1.

Programa:

```
import java.util.Scanner;
public class TablaMultiplicar {
    public void cargarValor() {
        Scanner teclado=new Scanner(System.in);
        int valor;
        do {
            System.out.print("Introduce valor:");
            valor=teclado.nextInt();
            if (valor!=-1) {
                calcular(valor);
            }
        } while (valor!=-1);
    }

    public void calcular(int v) {
        for(int f=v;f<=v*10;f=f+v) {
            System.out.print(f+"-");
        }
    }
}
```

```
public static void main(String[] ar) {  
    TablaMultiplicar tabla;  
    tabla=new TablaMultiplicar();  
    tabla.cargarValor();  
}  
}
```

```
TablaMultiplicar.java X
1
2 import java.util.Scanner;
3 public class TablaMultiplicar {
4     public void cargarValor() {
5         Scanner teclado=new Scanner(System.in);
6         int valor;
7         do {
8             System.out.print("Introduce valor:");
9             valor=teclado.nextInt();
10            if (valor!=-1) {
11                calcular(valor);
12            }
13        } while (valor!=-1);
14    }
15
16    public void calcular(int v) {
17        for(int f=v;f<=v*10;f=f+v) {
18            System.out.print(f+"-");
19        }
20    }
21
22    public static void main(String[] ar) {
23        TablaMultiplicar tabla;
24        tabla=new TablaMultiplicar();
25        tabla.cargarValor();
26    }
27 }
```

Problems @ Javadoc Declaration Console X

<terminated> TablaMultiplicar [Java Application] C:\Program Files\Java\j
Introduce valor:8
8-16-24-32-40-48-56-64-72-80-Introduce valor:5
5-10-15-20-25-30-35-40-45-50-Introduce valor:-1

En esta clase no hemos definido ningún atributo, ya que el objeto de la clase Scanner lo requerimos en un solo método, por ello lo definimos como una variable local.

El método calcular recibe un parámetro de tipo entero, luego lo utilizamos dentro del método para mostrar la tabla de multiplicar de dicho valor, para esto inicializamos la variable f con el valor que llega en el parámetro. Luego de cada ejecución del for incrementamos el contador f con el valor de v.

```
public void calcular(int v) {  
    for(int f=v;f<=v*10;f=f+v) {  
        System.out.print(f+"-");  
    }  
}
```

Un método puede no tener parámetros como hemos visto en problemas anteriores o puede tener uno o más parámetros (en caso de tener más de un parámetro los mismos se separan por coma)

El método cargarValores no tiene parámetros y tiene por objetivo cargar un valor entero por teclado y llamar al método calcular para que muestre la tabla de multiplicar del valor que le pasamos por teclado:

```
public void cargarValor() {  
    Scanner teclado=new Scanner(System.in);  
    int valor;  
    do {  
        System.out.print("Introduce valor:");  
        valor=teclado.nextInt();  
        if (valor!=-1) {  
            calcular(valor);  
        }  
    } while (valor!=-1);  
}
```

Como vemos al método calcular lo llamamos por su nombre y entre paréntesis le pasamos el dato a enviar (debe ser un valor o variable entera)

En este problema en la main solo llamamos al método cargarValor, ya que el método calcular luego es llamado por el método cargarValor:

```
public static void main(String[] ar) {  
    TablaMultiplicar tabla;  
    tabla=new TablaMultiplicar();  
    tabla.cargarValor();  
}
```

Métodos que retornan un dato.

Un método puede retornar un dato:

```
public [tipo de dato] [nombre del método]([parámetros]) {  
    [algoritmo]  
    return [tipo de dato]  
}
```

Cuando un método retorna un dato en vez de indicar la palabra clave void previo al nombre del método indicamos el tipo de dato que retorna. Luego dentro del algoritmo en el momento que queremos que finalice el mismo y retorne el dato empleamos la palabra clave return con el valor respectivo.

Problema 2:

Confeccionar una clase que permita introducir tres valores por teclado. Luego mostrar el mayor y el menor.

Programa:

```
import java.util.Scanner;
public class MayorMenor {
    public void cargarValores() {
        Scanner teclado=new Scanner(System.in);
        System.out.print("Introduce primer valor:");
        int valor1=teclado.nextInt();
        System.out.print("Introduce segundo valor:");
        int valor2=teclado.nextInt();
        System.out.print("Introduce tercer valor:");
        int valor3=teclado.nextInt();
        int mayor,menor;
        mayor=calcularMayor(valor1,valor2,valor3);
        menor=calcularMenor(valor1,valor2,valor3);
        System.out.println("El valor mayor de los tres es:"+mayor);
        System.out.println("El valor menor de los tres es:"+menor);
    }
}
```

```
public int calcularMayor(int v1,int v2,int v3) {  
    int m;  
    if(v1>v2 && v1>v3) {  
        m=v1;  
    } else {  
        if(v2>v3) {  
            m=v2;  
        } else {  
            m=v3;  
        }  
    }  
    return m;  
}
```

```
public int calcularMenor(int v1,int v2,int v3) {  
    int m;  
    if(v1<v2 && v1<v3) {  
        m=v1;  
    } else {  
        if(v2<v3) {  
            m=v2;  
        } else {  
            m=v3;  
        }  
    }  
    return m;  
}
```

```
public static void main(String[] ar) {  
    MayorMenor maymen=new MayorMenor();  
    maymen.cargarValores();  
}  
}
```

```
MayorMenor.java X
19 public int calcularMayor(int v1,int v2,int v3) {
20     int m;
21     if(v1>v2 && v1>v3) {
22         m=v1;
23     } else {
24         if(v2>v3) {
25             m=v2;
26         } else {
27             m=v3;
28         }
29     }
30     return m;
31 }
32
33 public int calcularMenor(int v1,int v2,int v3) {
34     int m;
35     if(v1<v2 && v1<v3) {
36         m=v1;
37     } else {
38         if(v2<v3) {
39             m=v2;
40         } else {
41             m=v3;
42         }
43     }
44     return m;
45 }
46
47 public static void main(String[] ar) {
```

Problems @ Javadoc Declaration Console X

<terminated> MayorMenor [Java Application] C:\Program Files\Java\jdk-17.0.1

```
Introduce primer valor:28
Introduce segundo valor:40
Introduce tercer valor:54
El valor mayor de los tres es:54
El valor menor de los tres es:28
```

Si vemos la sintaxis que calcula el mayor de tres valores enteros es similar al algoritmo visto en conceptos anteriores:

```
public int calcularMayor(int v1,int v2,int v3) {  
    int m;  
    if(v1>v2 && v1>v3) {  
        m=v1;  
    } else {  
        if(v2>v3) {  
            m=v2;  
        } else {  
            m=v3;  
        }  
    }  
    return m;  
}
```

Lo primero que podemos observar que el método retorna un entero y recibe tres parámetros:

```
public int calcularMayor(int v1,int v2,int v3) {
```

Dentro del método verificamos cual de los tres parámetros almacena un valor mayor, a este valor lo almacenamos en una variable local llamada "m", al valor almacenado en esta variable lo retornamos al final con un return.

La llamada al método calcularMayor lo hacemos desde dentro del método cargarCalores:

```
mayor=calcularMayor(valor1,valor2,valor3);
```

Debemos asignar a una variable el valor devuelto por el método calcularMayor. Luego el contenido de la variable mayor lo mostramos:

```
System.out.println("El valor mayor de los tres es:"+mayor);
```

La lógica es similar para el cálculo del menor.

16 - Estructura de datos tipo vector.

Hemos empleado variables de distinto tipo para el almacenamiento de datos (variables int, float, String) En esta sección veremos otros tipos de variables que permiten almacenar un conjunto de datos en una única variable.

Un vector es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente)

Problema 1:

Se desea guardar los sueldos de 5 operarios.

Según lo conocido deberíamos definir 5 variables si queremos tener en un cierto momento los 5 sueldos almacenados en memoria.

Empleando un vector solo se requiere definir un único nombre y accedemos a cada elemento por medio del subíndice.

| sueldos | | | | |
|------------|------------|------------|------------|------------|
| 1200 | 750 | 820 | 550 | 490 |
| sueldos[0] | sueldos[1] | sueldos[2] | sueldos[3] | sueldos[4] |

Programa:

```
import java.util.Scanner;
public class PruebaVector1 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar()
    {
        teclado = new Scanner(System.in);
        sueldos = new int[5];
        for(int f = 0;f < 5;f++) {
            System.out.print("Introduce valor de la componente:");
            sueldos[f] = teclado.nextInt();
        }
    }

    public void imprimir() {
        for(int f = 0;f < 5;f++) {
            System.out.println(sueldos[f]);
        }
    }
}
```

```
public static void main(String[] ar) {  
    PruebaVector1 pv = new PruebaVector1();  
    pv.cargar();  
    pv.imprimir();  
}  
}
```

```

PruebaVector1.java ×
1
2 import java.util.Scanner;
3 public class PruebaVector1 {
4     private Scanner teclado;
5     private int[] sueldos;
6
7     public void cargar()
8     {
9         teclado=new Scanner(System.in);
10        sueldos=new int[5];
11        for(int f=0;f<5;f++) {
12            System.out.print("Introduce el valor del componente:");
13            sueldos[f]=teclado.nextInt();
14        }
15    }
16
17    public void imprimir() {
18        for(int f=0;f<5;f++) {
19            System.out.println(sueldos[f]);
20        }
21    }
22
23    public static void main(String[] ar) {
24        PruebaVector1 pv=new PruebaVector1();
25        pv.cargar();

```

Problems @ Javadoc Declaration Console ×

<terminated> PruebaVector1 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (

```

Introduce el valor del componente:78
Introduce el valor del componente:54
Introduce el valor del componente:63
Introduce el valor del componente:21
Introduce el valor del componente:0
78
54
63
21
0

```

Para la declaración de un vector le antecedemos al nombre los corchetes abiertos y cerrados:

```
private int[] sueldos;
```

Lo definimos como atributo de la clase ya que lo utilizaremos en los dos métodos.

En el método de cargar lo primero que hacemos es crear el vector (en java los vectores son objetos por lo que es necesario proceder a su creación mediante el operador new):

```
sueldos = new int[5];
```

Cuando creamos el vector indicamos entre corchetes la cantidad de elementos que se pueden almacenar posteriormente en el mismo.

Para cargar cada componente debemos indicar entre corchetes que elemento del vector estamos accediendo:

```
for(int f=0;f<5;f++) {  
    System.out.print("Introduce el valor del componente:");  
    sueldos[f]=teclado.nextInt();  
}
```

La estructura de programación que más se adapta para cargar en forma completa las componentes de un vector es un for, ya que sabemos de antemano la cantidad de valores a cargar.

Cuando f vale cero estamos accediendo a la primer componente del vector (en nuestro caso sería):

```
sueldos[0]=teclado.nextInt();
```

Lo mas común es utilizar una estructura repetitiva for para recorrer cada componente del vector.

Utilizar el for nos reduce la cantidad de código, si no utilizo un for debería en forma secuencial implementar el siguiente código:

```
System.out.print("Introduce valor de la componente:");
sueldos[0]=teclado.nextInt();
System.out.print("Introduce valor de la componente:");
sueldos[1]=teclado.nextInt();
System.out.print("Introduce valor de la componente:");
sueldos[2]=teclado.nextInt();
System.out.print("Introduce valor de la componente:");
sueldos[3]=teclado.nextInt();
System.out.print("Introduce valor de la componente:");
sueldos[4]=teclado.nextInt();
```

La impresión de las componentes del vector lo hacemos en el otro método:

```
public void imprimir() {
    for(int f=0;f<5;f++) {
        System.out.println(sueldos[f]);
    }
}
```

Siempre que queremos acceder a una componente del vector debemos indicar entre corchetes la componente, dicho valor comienza a numerarse en cero y continua hasta un número menos del tamaño del vector, en nuestro caso creamos el vector con 5 elementos:

```
sueldos=new int[5];
```

Por último en este programa creamos un objeto en el main y llamamos a los métodos de cargar e imprimir el vector:

```
public static void main(String[] ar) {  
    PruebaVector1 pv=new PruebaVector1();  
    pv.cargar();  
    pv.imprimir();  
}
```

Problema 2:

Definir un vector de 5 componentes de tipo float que representen las alturas de 5 personas.

Obtener el promedio de las mismas. Contar cuántas personas son más altas que el promedio y cuántas más bajas.

Programa:

```
import java.util.Scanner;

public class PruebaVector2 {
    private Scanner teclado;
    private float[] alturas;
    private float promedio;

    public void cargar() {
        teclado=new Scanner(System.in);
        alturas=new float[5];
        for(int f=0;f<5;f++) {
            System.out.print("Introduce la altura de la persona:");
            alturas[f]=teclado.nextFloat();
        }
    }

    public void calcularPromedio() {
        float suma;
        suma=0;
        for(int f = 0;f < 5;f++) {
            suma=suma+alturas[f];
        }
        promedio=suma/5;
        System.out.println("Promedio de alturas:"+promedio);
    }
}
```

```
public void mayoresMenores() {  
    int may,men;  
    may=0;  
    men=0;  
    for(int f = 0;f < 5;f++) {  
        if (alturas[f]>promedio) {  
            may++;  
        } else {  
            if (alturas[f] < promedio) {  
                men++;  
            }  
        }  
    }  
}  
System.out.println("Cantidad de personas mayores al promedio:"+may);  
System.out.println("Cantidad de personas menores al promedio:"+men);  
}
```

```
public static void main(String[] ar) {  
    PruebaVector2 pv2=new PruebaVector2();  
    pv2.cargar();  
    pv2.calcularPromedio();  
    pv2.mayoresMenores();  
}  
}
```

```

PruebaVector2.java ×
23     promedio=suma/5;
24     System.out.println("Promedio de alturas:"+promedio);
25 }
26
27 public void mayoresMenores() {
28     int may,men;
29     may=0;
30     men=0;
31     for(int f=0;f<5;f++) {
32         if (alturas[f]>promedio) {
33             may++;
34         } else {
35             if (alturas[f]<promedio) {
36                 men++;
37             }
38         }
39     }
40     System.out.println("Cantidad de personas mayores al promedio:"+may);
41     System.out.println("Cantidad de personas menores al promedio:"+men);
42 }
43
44 public static void main(String[] ar) {
45     PruebaVector2 pv2=new PruebaVector2();
46     pv2.cargar();
47     pv2.calcularPromedio();

```

Problems @ Javadoc Declaration Console ×

<terminated> PruebaVector2 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (7 ene 2024)

```

Introduce la altura de la persona:1,85
Introduce la altura de la persona:1,78
Introduce la altura de la persona:2,05
Introduce la altura de la persona:1,76
Introduce la altura de la persona:1,92
Promedio de alturas:1.8720001
Cantidad de personas mayores al promedio:2
Cantidad de personas menores al promedio:3

```

Definimos como atributo un vector donde almacenaremos las alturas:

```
private float[] alturas;
```

En la carga creamos el vector indicando que reserve espacio para 5 componentes:

```
alturas=new float[5];
```

Procedemos seguidamente a cargar todos sus elementos:

```
for(int f = 0;f < 5;f++) {  
    System.out.print("Introduce la altura de la persona:");  
    alturas[f] = teclado.nextFloat();  
}
```

En otro método procedemos a sumar todas sus componentes y obtener el promedio. El promedio lo almacenamos en un atributo de la clase ya que lo necesitamos en otro método:

```
public void calcularPromedio() {  
    float suma;  
    suma=0;  
    for(int f = 0;f < 5;f++) {  
        suma=suma+alturas[f];  
    }  
    promedio=suma/5;  
    System.out.println("Promedio de alturas:"+promedio);  
}
```

Por último en un tercer método comparamos cada componente del vector con el atributo promedio, si el valor almacenado supera al promedio incrementamos un contador en caso que sea menor al promedio incrementamos otro contador:

```
public void mayoresMenores() {  
    int may,men;  
    may=0;  
    men=0;  
    for(int f = 0;f < 5;f++) {  
        if (alturas[f] > promedio) {  
            may++;  
        } else {  
            if (alturas[f] < promedio) {  
                men++;  
            }  
        }  
    }  
    System.out.println("Cantidad de personas mayores al promedio:"+may);  
    System.out.println("Cantidad de personas menores al promedio:"+men);  
}
```

Importante:

En este problema podemos observar una ventaja de tener almacenadas todas las alturas de las personas. Si no conociéramos los vectores tendríamos que cargar otra vez las alturas por teclado para compararlas con el promedio.

Mientras el programa está en ejecución tenemos el vector alturas a nuestra disposición. Es importante tener en cuenta que cuando finaliza la ejecución del programa se pierde el contenido de todas las variables (simples y vectores)

Problema 3:

Una empresa tiene dos turnos (mañana y tarde) en los que trabajan 8 empleados (4 por la mañana y 4 por la tarde).

Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados por turno.

Imprimir los gastos en sueldos de cada turno.

Programa:

```
import java.util.Scanner;
public class PruebaVector3 {
    private Scanner teclado;
    private float[] turnoMan;
    private float[] turnoTar;

    public void cargar() {
        teclado=new Scanner(System.in);
        turnoMan=new float[4];
        turnoTar=new float[4];
        System.out.println("Sueldos de empleados del turno de la mañana.");
        for(int f=0;f<4;f++) {
            System.out.print("Introduce sueldo:");
            turnoMan[f]=teclado.nextFloat();
        }
        System.out.println("Sueldos de empleados del turno de la tarde.");
        for(int f=0;f<4;f++) {
            System.out.print("Introduce sueldo:");
            turnoTar[f]=teclado.nextFloat();
        }
    }
}
```



```
public void calcularGastos() {  
    float man = 0;  
    float tar = 0;  
    for(int f = 0;f < 4;f++){  
        man = man+turnoMan[f];  
        tar = tar+turnoTar[f];  
    }  
    System.out.println("Total de gastos del turno de la mañana:"+man);  
    System.out.println("Total de gastos del turno de la tarde:"+tar);  
}
```

```
public static void main(String[] ar){  
    PruebaVector3 pv=new PruebaVector3();  
    pv.cargar();  
    pv.calcularGastos();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class PruebaVector3 {
4     private Scanner teclado;
5     private float[] turnoMan;
6     private float[] turnoTar;
7
8     public void cargar() {
9         teclado=new Scanner(System.in);
10        turnoMan=new float[4];
11        turnoTar=new float[4];
12        System.out.println("Sueldos de empleados del turno de la mañana.");
13        for(int f=0;f<4;f++) {
14            System.out.print("Introduce sueldo:");
15            turnoMan[f]=teclado.nextFloat();
16        }
17        System.out.println("Sueldos de empleados del turno de la tarde.");
18        for(int f=0;f<4;f++) {
19            System.out.print("Introduce sueldo:");
20            turnoTar[f]=teclado.nextFloat();
21        }
22    }
```

<terminated> PruebaVector3 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (7 ene 2022)

Sueldos de empleados del turno de la mañana.

Introduce sueldo:1500

Introduce sueldo:1800

Introduce sueldo:2000

Introduce sueldo:1400

Sueldos de empleados del turno de la tarde.

Introduce sueldo:2000

Introduce sueldo:1500

Introduce sueldo:1800

Introduce sueldo:1700

Total de gastos del turno de la mañana:6700.0

Total de gastos del turno de la tarde:7000.0

Definimos dos atributos de tipo vector donde almacenaremos los sueldos de los empleados de cada turno:

```
private float[] turnoMan;  
private float[] turnoTar;
```

Creamos los vectores con cuatro elementos cada uno:

```
turnoMan=new float[4];  
turnoTar=new float[4];
```

Mediante dos estructuras repetitivas procedemos a cargar cada vector:

```
System.out.println("Sueldos de empleados del turno de la mañana.");  
for(int f=0;f<4;f++) {  
    System.out.print("Introduce sueldo:");  
    turnoMan[f]=teclado.nextFloat();  
}  
System.out.println("Sueldos de empleados del turno de la tarde.");  
for(int f=0;f<4;f++) {  
    System.out.print("Introduce sueldo:");  
    turnoTar[f]=teclado.nextFloat();  
}
```

En otro método procedemos a sumar las componentes de cada vector y mostrar dichos acumuladores:

```
float man=0;
float tar=0;
for(int f=0;f<4;f++){
    man=man+turnoMan[f];
    tar=tar+turnoTar[f];
}
```

```
System.out.println("Total de gastos del turno de la mañana:"+man);
System.out.println("Total de gastos del turno de la tarde:"+tar);
```

Problemas propuestos

1. Desarrollar un programa que permita ingresar un vector de 8 elementos, e informe:

El valor acumulado de todos los elementos del vector.

El valor acumulado de los elementos del vector que sean mayores a 36.

Cantidad de valores mayores a 50.

2. Realizar un programa que pida la carga de dos vectores numéricos enteros de 4 elementos. Obtener la suma de los dos vectores, dicho resultado guardarlo en un tercer vector del mismo tamaño. Sumar componente a componente.
3. Se tienen las notas del primer parcial de los alumnos de dos cursos, el curso A y el curso B, cada curso cuenta con 5 alumnos.

Realizar un programa que muestre el curso que obtuvo el mayor promedio general.

4. Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenado de menor a mayor.

17 - Vector (Tamaño de un vector)

Como hemos visto cuando se crea un vector indicamos entre corchetes su tamaño:

```
sueldos=new int[5];
```

Luego cuando tenemos que recorrer dicho vector disponemos una estructura repetitiva for:

```
for(int f=0;f<5;f++) {  
    System.out.print("Introduce valor de la componente:");  
    sueldos[f]=teclado.nextInt();  
}
```

Como vemos el for se repite mientras el contador f vale menos de 5. Esta estructura repetitiva es idéntica cada vez que recorremos el vector.

Que pasa ahora si cambiamos el tamaño del vector cuando lo creamos:

```
sueldos=new int[7];
```

Con esto tenemos que cambiar todos los for que recorren dicho vector. Ahora veremos que un vector al ser un objeto tiene un atributo llamado length que almacena su tamaño. Luego podemos modificar todos los for con la siguiente sintaxis:

```
for(int f=0;f<sueldos.length;f++) {  
    System.out.print("Ingrese valor de la componente:");  
    sueldos[f]=teclado.nextInt();  
}
```

También podemos pedir al usuario que indique el tamaño del vector en tiempo de ejecución, en estos casos se hace imprescindible el empleo del atributo length.

Problema 1:

Se desea almacenar los sueldos de operarios. Cuando se ejecuta el programa se debe pedir la cantidad de sueldos a ingresar. Luego crear un vector con dicho tamaño.

Programa:

```
import java.util.Scanner;

public class PruebaVector8 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar()
    {
        teclado=new Scanner(System.in);
        System.out.print("Cuantos sueldos introducirás:");
        int cant;
        cant=teclado.nextInt();
        sueldos=new int[cant];
        for(int f=0;f<sueldos.length;f++) {
            System.out.print("Introduce sueldo:");
            sueldos[f]=teclado.nextInt();
        }
    }

    public void imprimir() {
        for(int f=0;f<sueldos.length;f++) {
            System.out.println(sueldos[f]);
        }
    }
}
```



```
public static void main(String[] ar) {  
    PruebaVector8 pv=new PruebaVector8();  
    pv.cargar();  
    pv.imprimir();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class PruebaVector8 {
4     private Scanner teclado;
5     private int[] sueldos;
6
7     public void cargar()
8     {
9         teclado=new Scanner(System.in);
10        System.out.print("Cuantos sueldos introduciras:");
11        int cant;
12        cant=teclado.nextInt();
13        sueldos=new int[cant];
14        for(int f=0;f<sueldos.length;f++) {
15            System.out.print("Introduce sueldo:");
16            sueldos[f]=teclado.nextInt();
17        }
18    }
19
20    public void imprimir() {
21        for(int f=0;f<sueldos.length;f++) {
22            System.out.println(sueldos[f]);
23        }
24    }
25
26    public static void main(String[] ar) {
27        PruebaVector8 pv=new PruebaVector8();
28        pv.cargar();
29        pv.imprimir();
30    }
```

<terminated> PruebaVector8 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\

Cuantos sueldos introduciras:2

Introduce sueldo:1450

Introduce sueldo:1750

1450

1750

La definición del vector no varía:

```
private int[] sueldos;
```

Luego para la creación del mismo ingresamos una variable entera y la utilizamos como subíndice en el momento de la creación del vector:

```
System.out.print("Cuantos sueldos cargará:");  
int cant;  
cant=teclado.nextInt();  
sueldos=new int[cant];
```

Las estructuras repetitivas las acotamos accediendo al atributo length del vector:

```
for(int f=0;f<sueldos.length;f++) {  
    System.out.print("Introduce sueldo:");  
    sueldos[f]=teclado.nextInt();  
}
```

Problema propuesto

1. Desarrollar un programa que permita introducir un vector de n elementos, introducir n por teclado. Luego imprimir la suma de todos sus elementos.

18 - Vectores paralelos

Este concepto se da cuando hay una relación entre las componentes de **igual subíndice (misma posición) de un vector y otro.**

| | | | | | |
|----------------|------|-----|--------|-------|-------|
| <i>nombres</i> | Juan | Ana | Marcos | Pablo | Laura |
| <i>edades</i> | 12 | 21 | 27 | 14 | 21 |

Si tenemos dos vectores de 5 elementos cada uno. En uno se almacenan los nombres de personas en el otro las edades de dichas personas.

Decimos que el **vector nombres es paralelo al vector edades** si en la componente 0 de cada vector se almacena información relacionada a una persona (Juan - 12 años)

Es decir hay una relación entre cada componente de los dos vectores.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en las estructuras de datos.

Problema 1:

Desarrollar un programa que permita cargar 5 nombres de personas y sus edades respectivas. Luego de realizar la carga por teclado de todos los datos imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años).

Programa:

```
import java.util.Scanner;
public class PruebaVector10 {
    private Scanner teclado;
    private String[] nombres;
    private int[] edades;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        edades=new int[5];
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Introduce nombre:");
            nombres[f]=teclado.next();
            System.out.print("Introduce edad:");
            edades[f]=teclado.nextInt();
        }
    }
}
```

```
public void mayoresEdad() {  
    System.out.println("Personas mayores de edad.");  
    for(int f=0;f<nombres.length;f++) {  
        if (edades[f]>=18) {  
            System.out.println(nombres[f]);  
        }  
    }  
}
```

```
public static void main(String[] ar) {  
    PruebaVector10 pv=new PruebaVector10();  
    pv.cargar();  
    pv.mayoresEdad();  
}
```


PruebaVector10.java X

```
1
2 import java.util.Scanner;
3 public class PruebaVector10 {
4     private Scanner teclado;
5     private String[] nombres;
6     private int[] edades;
7
8     public void cargar() {
9         teclado=new Scanner(System.in);
10        nombres=new String[5];
11        edades=new int[5];
12        for(int f=0;f<nombres.length;f++) {
13            System.out.print("Introduce nombre:");
14            nombres[f]=teclado.next();
15            System.out.print("Introduce edad:");
16            edades[f]=teclado.nextInt();
17        }
18    }
19 }
```

Problems @ Javadoc Declaration Console X

<terminated> PruebaVector10 [Java Application] C:\Program Files\Java\jdk-1

```
Introduce nombre:maria
Introduce edad:26
Introduce nombre:luís
Introduce edad:31
Introduce nombre:javier
Introduce edad:24
Introduce nombre:laura
Introduce edad:30
Introduce nombre:jose
Introduce edad:28
Personas mayores de edad.
maria
luís
javier
laura
jose
```

Definimos los dos vectores:

```
private String[] nombres;  
private int[] edades;
```

Creamos los dos vectores con 5 elementos cada uno:

```
nombres = new String[5];  
edades = new int[5];
```

Mediante un for procedemos a la carga de los elementos de los vectores:

```
for(int f=0;f<nombres.length;f++) {  
    System.out.print("Introduce nombre:");  
    nombres[f] = teclado.next();  
    System.out.print("Introduce edad:");  
    edades[f] = teclado.nextInt();  
}
```

Podemos utilizar el length de cualquiera de los dos vectores, ya que tienen el mismo tamaño.

Para imprimir los nombres de las personas mayores de edad verificamos cada componente del vector de edades, en caso que sea igual o mayor o 18 procedemos a mostrar el elemento de la misma posición del otro vector:

```
    for(int f = 0;f < nombres.length;f++) {  
    if (edades[f]>=18) {  
        System.out.println(nombres[f]);  
    }  
    }
```

19 - Vectores (mayor y menor elemento)

Es una actividad común la búsqueda del mayor y menor elemento de un vector, lo mismo que su posición.

sueldos

| | | | | |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 120 | 750 | 820 | 550 | 490 |
| <code>sueldos[0]</code> | <code>sueldos[1]</code> | <code>sueldos[2]</code> | <code>sueldos[3]</code> | <code>sueldos[4]</code> |

El mayor elemento es el 820 y se encuentra en la posición n° 2.

Problema 1:

Confeccionar un programa que permita cargar los nombres de 5 operarios y sus sueldos respectivos. Mostrar el sueldo mayor y el nombre del operario.

Programa:

```
import java.util.Scanner;

public class PruebaVector11 {
    private Scanner teclado;
    private String[] nombres;
    private float[] sueldos;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        sueldos=new float[5];
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Introduce el nombre del empleado:");
            nombres[f]=teclado.next();
            System.out.print("Introduce el sueldo:");
            sueldos[f]=teclado.nextFloat();
        }
    }

    public void mayorSueldo() {
        float mayor;
        int pos;
        mayor=sueldos[0];
        pos=0;
```

```
for(int f=1;f<nombres.length;f++) {  
    if (sueldos[f]>mayor) {  
        mayor=sueldos[f];  
        pos=f;  
    }  
}
```

```
System.out.println("El empleado con sueldo mayor es "+nombres[pos]);  
System.out.println("Tiene un sueldo:"+mayor);  
}
```

```
public static void main(String[] ar) {  
    PruebaVector11 pv=new PruebaVector11();  
    pv.cargar();  
    pv.mayorSueldo();  
}  
}
```

```
PruebaVector11.java X
1
2 import java.util.Scanner;
3 public class PruebaVector11 {
4     private Scanner teclado;
5     private String[] nombres;
6     private float[] sueldos;
7
8     public void cargar() {
9         teclado=new Scanner(System.in);
10        nombres=new String[5];
11        sueldos=new float[5];
12        for(int f=0;f<nombres.length;f++) {
13            System.out.print("Introduce el nombre del empleado:");
14            nombres[f]=teclado.next();
15            System.out.print("Introduce el sueldo:");
16            sueldos[f]=teclado.nextFloat();
17        }
18    }
19
20    public void mayorSueldo() {
21        float mayor;
22        int pos;
23        mayor=sueldos[0];
```

Problems @ Javadoc Declaration Console X

<terminated> PruebaVector11 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe

Introduce el nombre del empleado:juan
Introduce el sueldo:1500
Introduce el nombre del empleado:eva
Introduce el sueldo:1800
Introduce el nombre del empleado:luis
Introduce el sueldo:1200
Introduce el nombre del empleado:lidia
Introduce el sueldo:1500
Introduce el nombre del empleado:pepe
Introduce el sueldo:2000
El empleado con sueldo mayor es pepe
Tiene un sueldo:2000.0

Definimos los dos vectores paralelos donde almacenaremos los nombres y los sueldos de los operarios:

```
private String[] nombres;  
private float[] sueldos;
```

Creamos los dos vectores y procedemos a cargar sus elementos:

```
nombres=new String[5];  
sueldos=new float[5];  
for(int f=0;f<nombres.length;f++) {  
    System.out.print("Introduce el nombre del empleado:");  
    nombres[f]=teclado.next();  
    System.out.print("Introduce el sueldo:");  
    sueldos[f]=teclado.nextFloat();  
}
```


Para obtener el mayor sueldo y el nombre del operario realizar los siguientes pasos:

Inicializamos una variable mayor con la primer componente del vector sueldos:

```
mayor = sueldos[0];
```

Inicializamos una variable pos con el valor 0, ya que decimos primeramente que el mayor es la primer componente del vector:

```
pos = 0;
```

Recorremos las componentes del vector que faltan analizar, o sea, de la 1 a la 4:

```
for(int f = 1;f < nombres.length;f++) {
```

Accedemos a cada componente para controlar si supera lo que tiene la variable mayor:

```
if (sueldos[f]>mayor) {
```

En caso de ser verdadera la condición asignamos a la variable mayor este nuevo valor sueldos[f]

```
mayor = sueldos[f];
```

y a la variable pos le cargamos la variable f que indica la componente que estamos analizando:

```
pos = f
```

Cuando salimos de la estructura repetitiva imprimimos la variable mayor que contiene el mayor sueldo y para imprimir el nombre del operario conociendo la posición del mayor sueldo imprimimos el elemento que ocupa la posición que indica la variable pos en el vector paralelo:

```
System.out.println("El empleado con sueldo mayor es "+nombres[pos]);  
System.out.println("Tiene un sueldo:"+mayor);
```

Problema propuesto

Cargar un vector de n elementos. imprimir el menor y un mensaje si se repite dentro del vector.

20 - Vectores (ordenamiento)

El ordenamiento de un vector se logra intercambiando las componentes de manera que:

$\text{vec}[0] \leq \text{vec}[1] \leq \text{vec}[2]$ etc.

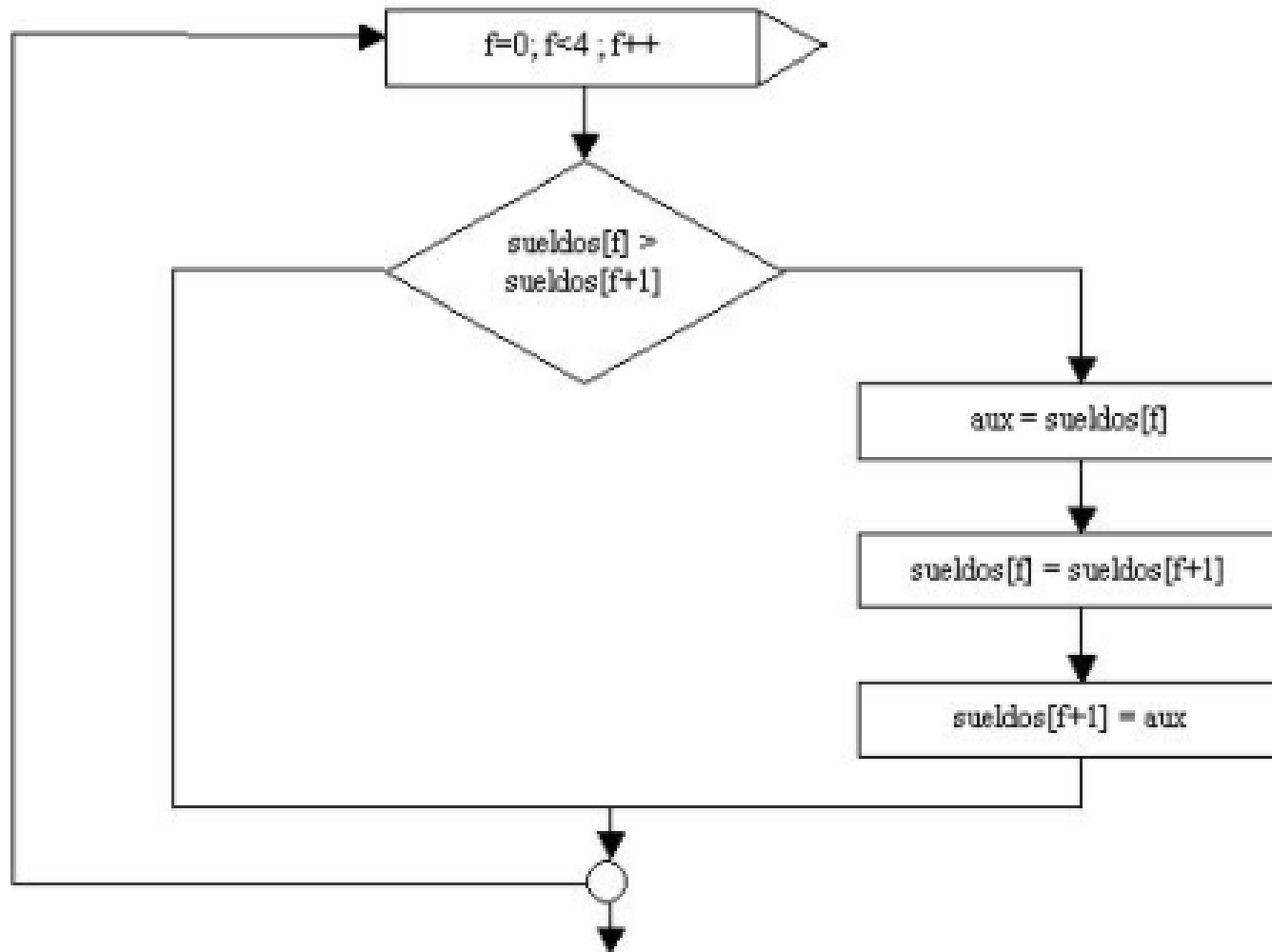
El contenido de la componente $\text{vec}[0]$ sea menor o igual al contenido de la componente $\text{vec}[1]$ y así sucesivamente.

Si se cumple lo dicho anteriormente decimos que el vector está ordenado de menor a mayor. Igualmente podemos ordenar un vector de mayor a menor.

Se puede ordenar tanto vectores con componentes de **tipo int, float como String**. En este último caso el ordenamiento es alfabético.

Problema 1:

Se debe crear un vector donde almacenar 5 sueldos. Ordenar el vector sueldos de menor a mayor.



Esta primera aproximación tiene por objetivo analizar los intercambios de elementos dentro del vector.

El algoritmo consiste en comparar si la primera componente es mayor a la segunda, en caso que la condición sea verdadera, intercambiamos los contenidos de las componentes.

Vamos a suponer que se introducen los siguientes valores por teclado:

1200

750

820

550

490

En este ejemplo: ¿es 1200 mayor a 750? La respuesta es verdadera, por lo tanto intercambiamos el contenido de la componente 0 con el de la componente 1.

Luego comparamos el contenido de la componente 1 con el de la componente 2: ¿Es 1200 mayor a 820?

La respuesta es verdadera entonces intercambiamos.

Si hay 5 componentes hay que hacer 4 comparaciones, por eso el for se repite 4 veces.

Generalizando: si el vector tiene N componentes hay que hacer $N-1$ comparaciones.

| Cuando | $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|--------|---------|---------|---------|---------|
| | 750 | 750 | 750 | 750 |
| | 1200 | 820 | 820 | 820 |
| | 820 | 1200 | 550 | 550 |
| | 550 | 550 | 1200 | 490 |
| | 490 | 490 | 490 | 1200 |

Podemos ver cómo el valor más grande del vector desciende a la última componente. Empleamos una variable auxiliar (aux) para el proceso de intercambio:

```
aux=sueldos[f];  
sueldos[f]=sueldos[f+1];  
sueldos[f+1]=aux;
```

Al salir del for en este ejemplo el contenido del vector es el siguiente:

750
820
550
490
1200

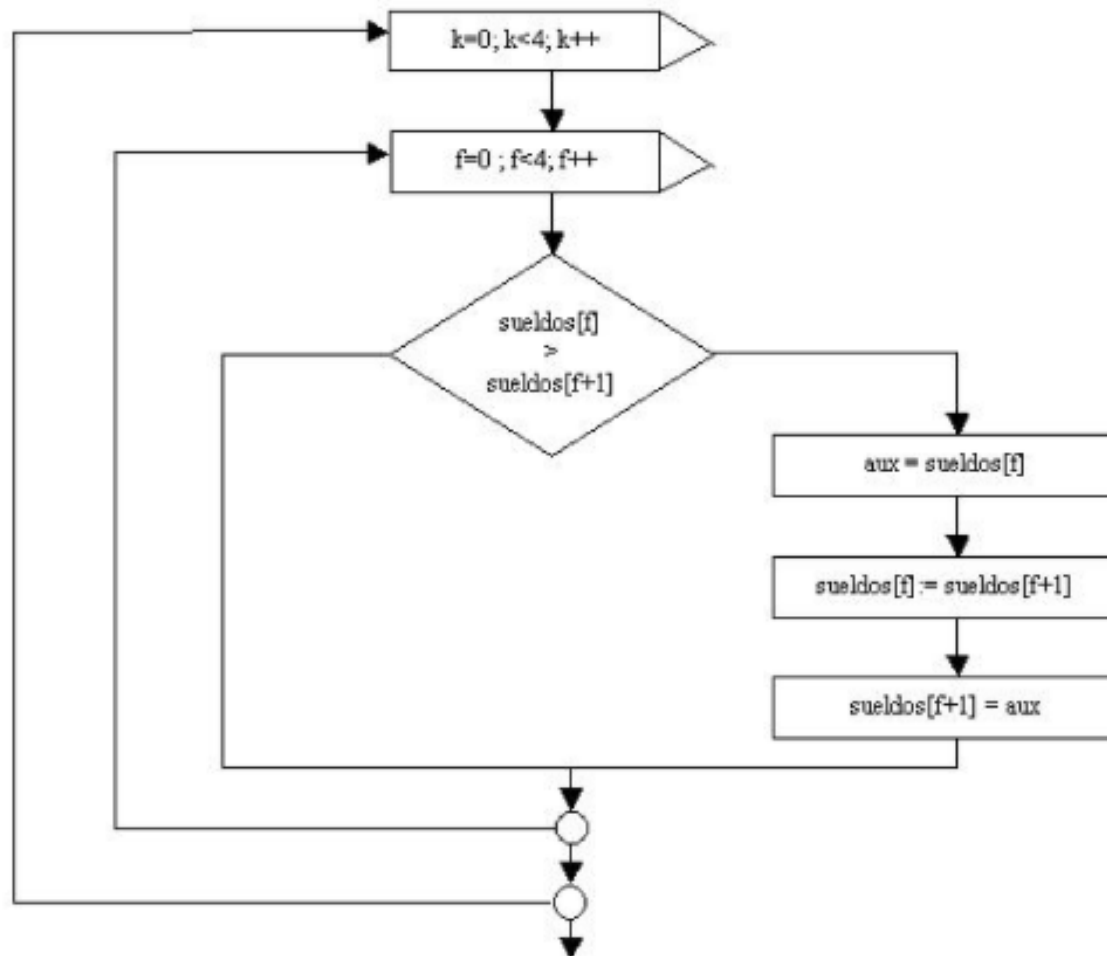
Analizando el algoritmo podemos comprobar que el elemento mayor del vector se ubica ahora en el último lugar.

Podemos definir otros vectores con distintos valores y comprobar que siempre el elemento mayor queda al final.

Pero todavía con este algoritmo no se ordena un vector. Solamente está ordenado el último elemento del vector.

Ahora bien, con los 4 elementos que nos quedan podemos hacer el mismo proceso visto anteriormente, con lo cual quedará ordenado otro elemento del vector. Este proceso lo repetiremos hasta que quede ordenado por completo el vector.

Como debemos repetir el mismo algoritmo podemos englobar todo el bloque en otra estructura repetitiva.



Realicemos una prueba del siguiente algoritmo:

Cuando $k = 0$

| $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|---------|---------|---------|---------|
| 750 | 750 | 750 | 750 |
| 1200 | 820 | 820 | 820 |
| 820 | 1200 | 550 | 550 |
| 550 | 550 | 1200 | 490 |
| 490 | 490 | 490 | 1200 |

Cuando $k = 1$

| $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|---------|---------|---------|---------|
| 750 | 750 | 750 | 750 |
| 820 | 550 | 550 | 550 |
| 550 | 820 | 490 | 490 |
| 490 | 490 | 820 | 820 |
| 1200 | 1200 | 1200 | 1200 |

Cuando $k = 2$

| $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|---------|---------|---------|---------|
| 550 | 550 | 550 | 550 |
| 750 | 490 | 490 | 490 |
| 490 | 750 | 750 | 750 |
| 820 | 820 | 820 | 820 |
| 1200 | 1200 | 1200 | 1200 |

Cuando $k = 3$

| $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|---------|---------|---------|---------|
| 490 | 490 | 490 | 490 |
| 550 | 550 | 550 | 550 |
| 750 | 750 | 750 | 750 |
| 820 | 820 | 820 | 820 |
| 1200 | 1200 | 1200 | 1200 |

¿Porque repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenada una componente del vector. A primera vista diríamos que deberíamos repetir el for externo la cantidad de componentes del vector, en este ejemplo el vector sueldos tiene 5 componentes.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos queda el otro automáticamente ordenado (podemos imaginar que si tenemos un vector con 2 elementos no se requiere el for externo, porque este debería repetirse una única vez)

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando, continuamos comparándolos.

Ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 del vector. En la segunda ejecución comparamos si el 820 es mayor a 1200, lo cual seguramente será falso.

Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en uno la cantidad de comparaciones.

Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura: (f=0 ; f<4-k; f++)

Es decir restarle el valor del contador del for externo.

Programa:

```
import java.util.Scanner;
public class PruebaVector13 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar() {
        teclado=new Scanner(System.in);
        sueldos=new int[5];
        for(int f=0;f<sueldos.length;f++) {
            System.out.print("Introduce el sueldo:");
            sueldos[f]=teclado.nextInt();
        }
    }

    public void ordenar() {
        for(int k=0;k<4;k++) {
            for(int f=0;f<4-k;f++) {
                if (sueldos[f]>sueldos[f+1]) {
                    int aux;
                    aux=sueldos[f];
                    sueldos[f]=sueldos[f+1];
                    sueldos[f+1]=aux;
                }
            }
        }
    }
}
```

```
    }  
}  
public void imprimir() {  
    System.out.println("Sueldos ordenados de menor a mayor.");  
    for(int f=0;f<sueldos.length;f++) {  
        System.out.println(sueldos[f]);  
    }  
}  
  
public static void main(String[] ar) {  
    PruebaVector13 pv=new PruebaVector13();  
    pv.cargar();  
    pv.ordenar();  
    pv.imprimir();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class PruebaVector13 {
4     private Scanner teclado;
5     private int[] sueldos;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         sueldos=new int[5];
10        for(int f=0;f<sueldos.length;f++) {
11            System.out.print("Introduce el sueldo:");
12            sueldos[f]=teclado.nextInt();
13        }
14    }
15
16    public void ordenar() {
17        for(int k=0;k<4;k++) {
18            for(int f=0;f<4-k;f++) {
19                if (sueldos[f]>sueldos[f+1]) {
20                    int aux;
21                    aux=sueldos[f];
22                    sueldos[f]=sueldos[f+1];
23                    sueldos[f+1]=aux;
24                }
25            }
26        }
27    }
28 }
```

<terminated> PruebaVector13 [Java Application] C:\Program Files\Java\jdk-17.

```
Introduce el sueldo:1500
Introduce el sueldo:1200
Introduce el sueldo:800
Introduce el sueldo:1000
Introduce el sueldo:1400
Sueldos ordenados de menor a mayor.
800
1000
1200
1400
1500
```


También podemos ordenar vectores cuyas componentes sean de tipo String. Para esto no podemos utilizar el operador `>` sino debemos utilizar un método de la clase String:

```
String cad1="juan";  
String cad2="analia";  
if (cad1.compareTo(cad2)>0)  
{  
    System.out.println(cad1 + " es mayor alfabéticamente que " + cad2);  
}
```

El método `compareTo` retorna un valor mayor a cero si `cad1` es mayor alfabéticamente. En este ejemplo `cad1` tiene un valor alfabéticamente mayor a `cad2`, luego el `compareTo` retorna un valor mayor a cero.

Si los dos String son exactamente iguales el método `compareTo` retorna un cero, y finalmente si `cad1` es menor alfabeticamente retorna un valor menor a cero.

Problema 2:

Definir un vector donde almacenar los nombres de 5 países. Confeccionar el algoritmo de ordenamiento alfabético.

Programa:

```
import java.util.Scanner;

public class PruebaVector14 {
    private Scanner teclado;
    private String[] paises;

    public void cargar() {
        teclado=new Scanner(System.in);
        paises=new String[5];
        for(int f=0;f<paises.length;f++) {
            System.out.print("Introduce el nombre del país:");
            paises[f]=teclado.next();
        }
    }

    public void ordenar() {
        for(int k=0;k<4;k++) {
            for(int f=0;f<4-k;f++) {
                if (paises[f].compareTo(paises[f+1])>0) {
                    String aux;
                    aux=paises[f];
                    paises[f]=paises[f+1];
                    paises[f+1]=aux;
                }
            }
        }
    }
}
```

```
    }  
}  
public void imprimir() {  
    System.out.println("Países ordenados en forma alfabética:");  
    for(int f=0;f<países.length;f++) {  
        System.out.println(países[f]);  
    }  
}  
  
public static void main(String[] ar) {  
    PruebaVector14 pv=new PruebaVector14();  
    pv.cargar();  
    pv.ordenar();  
    pv.imprimir();  
}  
}
```

```
1 import java.util.Scanner;
2 public class PruebaVector14 {
3     private Scanner teclado;
4     private String[] paises;
5
6     public void cargar() {
7         teclado=new Scanner(System.in);
8         paises=new String[5];
9         for(int f=0;f<paises.length;f++) {
10             System.out.print("Introduce el nombre del pais:");
11             paises[f]=teclado.next();
12         }
13     }
14
15     public void ordenar() {
16         for(int k=0;k<4;k++) {
17             for(int f=0;f<4-k;f++) {
18                 if (paises[f].compareTo(paises[f+1])>0) {
19                     String aux;
20                     aux=paises[f];
21                     paises[f]=paises[f+1];
22                     paises[f+1]=aux;
23                 }
24             }
25         }
26     }
27 }
```

<terminated> PruebaVector14 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java

Introduce el nombre del pais:Alemania

Introduce el nombre del pais:Irlanda

Introduce el nombre del pais:Italia

Introduce el nombre del pais:Suiza

Países ordenados en forma alfabética:

Alemania

España

Irlanda

Italia

Suiza

Definimos un vector de tipo String:

```
private String[] paises;
```

Lo creamos indicando que almacenará cinco elementos:

```
paises=new String[5];
```

Procedemos a cargar el vector:

```
for(int f=0;f<paises.length;f++) {  
    System.out.print("Ingrese el nombre del pais:");  
    paises[f]=teclado.next();  
}
```

Para el ordenamiento utilizamos el método `compareTo` para verificar si tenemos que intercambiar las componentes:

```
if (paises[f].compareTo(paises[f+1])>0) {
```

En el caso que si tenemos que intercambiarla utilizamos un auxiliar de tipo String:

```
String aux;  
aux=paises[f];  
paises[f]=paises[f+1];  
paises[f+1]=aux;
```

Problema propuesto

1. Cargar un vector de n elementos de tipo entero. Ordenar posteriormente el vector.

21 - Vectores (ordenamiento con vectores paralelos)

Cuando se tienen vectores paralelos y se ordena uno de ellos hay que tener la precaución de intercambiar los elementos de los vectores paralelos.

Problema 1:

Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas. Luego ordenar las notas de mayor a menor. Imprimir las notas y los nombres de los alumnos.

Programa:

```
import java.util.Scanner;
public class PruebaVector16 {
    private Scanner teclado;
    private String[] nombres;
    private int[] notas;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        notas=new int[5];
        System.out.println("Carga de nombres y notas");
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Introduce el nombre del alumno:");
            nombres[f]=teclado.next();
            System.out.print("Introduce la nota del alumno:");
            notas[f]=teclado.nextInt();
        }
    }
}
```

```
public void ordenar() {  
    for(int k=0;k<notas.length;k++) {  
        for(int f=0;f<notas.length-1-k;f++) {  
            if (notas[f]<notas[f+1]) {  
                int auxnota;  
                auxnota=notas[f];  
                notas[f]=notas[f+1];  
                notas[f+1]=auxnota;  
                String auxnombre;  
                auxnombre=nombres[f];  
                nombres[f]=nombres[f+1];  
                nombres[f+1]=auxnombre;  
            }  
        }  
    }  
}
```

```
public void imprimir() {  
    System.out.println("Nombres de alumnos y notas de mayor a menor");  
    for(int f=0;f<notas.length;f++) {  
        System.out.println(nombres[f] + " - " + notas[f]);  
    }  
}
```

```
public static void main(String[] ar) {  
    PruebaVector16 pv=new PruebaVector16();  
    pv.cargar();  
    pv.ordenar();  
    pv.imprimir();  
}  
}
```

PruebaVector16.java ×

```
1
2 import java.util.Scanner;
3 public class PruebaVector16 {
4     private Scanner teclado;
5     private String[] nombres;
6     private int[] notas;
7
8     public void cargar() {
9         teclado=new Scanner(System.in);
10        nombres=new String[5];
11        notas=new int[5];
12        System.out.println("Carga de nombres y notas");
13        for(int f=0;f<nombres.length;f++) {
14            System.out.print("Introduce el nombre del alumno:");
15            nombres[f]=teclado.next();
16            System.out.print("Introduce la nota del alumno:");
17            notas[f]=teclado.nextInt();
18        }
19    }
```

Problems @ Javadoc Declaration Console ×

<terminated> PruebaVector16 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.

Carga de nombres y notas

Introduce el nombre del alumno:juan

Introduce la nota del alumno:8

Introduce el nombre del alumno:begoña

Introduce la nota del alumno:3

Introduce el nombre del alumno:luís

Introduce la nota del alumno:5

Introduce el nombre del alumno:maría

Introduce la nota del alumno:9

Introduce el nombre del alumno:pedro

Introduce la nota del alumno:7

Nombres de alumnos y notas de mayor a menor

maría - 9

juan - 8

pedro - 7

luís - 5

begoña - 3

Definimos los dos vectores:

```
private String[] nombres;  
private int[] notas;
```

Creamos los dos vectores paralelos con cinco elementos cada uno:

```
nombres=new String[5];  
notas=new int[5];
```

En el proceso de ordenamiento dentro de los dos for verificamos si debemos intercambiar los elementos del vector notas:

```
for(int k=0;k<notas.length;k++) {  
    for(int f=0;f<notas.length-1-k;f++) {  
        if (notas[f]<notas[f+1]) {
```

En el caso que la nota de la posición 'f' sea menor a de la posición siguiente 'f+1' procedemos a intercambiar las notas:

```
int auxnota;  
auxnota=notas[f];  
notas[f]=notas[f+1];  
notas[f+1]=auxnota;
```

y simultáneamente procedemos a intercambiar los elementos del vector paralelo (con esto logramos que los dos vectores continuen siendo vectores paralelos):

```
String auxnombre;  
auxnombre=nombres[f];  
nombres[f]=nombres[f+1];  
nombres[f+1]=auxnombre;
```

Como vemos utilizamos dos auxiliares distintos porque los elementos de los dos vectores son de distinto tipo (int y String)

Si deseamos ordenar alfabéticamente la condición dependerá del vector nombres.

Problema propuesto

Cargar en un vector los nombres de 5 países y en otro vector paralelo la cantidad de habitantes del mismo. Ordenar alfabéticamente e imprimir los resultados. Por último ordenar con respecto a la cantidad de habitantes (de mayor a menor) e imprimir nuevamente.

22 - Estructura de datos tipo matriz

Una matriz es una estructura de datos que permite almacenar un **CONJUNTO de datos del MISMO tipo**.

Con un único nombre se define la matriz y por medio de **DOS subíndices** hacemos referencia a cada elemento de la misma (componente)

mat

Columnas

| | | | | | |
|----|----|----|----|-----|-----|
| | 50 | 5 | 27 | 400 | 7 |
| 0 | | 67 | 90 | 6 | 97 |
| 30 | | 14 | 23 | 251 | 490 |

Filas

Hemos graficado una matriz de 3 filas y 5 columnas. Para hacer referencia a cada elemento debemos indicar primero la fila y luego la columna, por ejemplo en la componente 1,4 se almacena el valor 97.

En este ejemplo almacenamos valores enteros. Todos los elementos de la matriz deben ser del mismo tipo (int, float, String etc.)

Las filas y columnas comienzan a numerarse a partir de cero, similar a los vectores.

Una matriz se la puede representar por un conjunto de vectores.

Problema 1:

Crear una matriz de 3 filas por 5 columnas con elementos de tipo int, cargar sus componentes y luego imprimirlas.

Programa:

```
import java.util.Scanner;
public class Matriz1 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        mat=new int[3][5];
        for(int f=0;f<3;f++) {
            for(int c=0;c<5;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void imprimir() {
        for(int f=0;f<3;f++) {
            for(int c=0;c<5;c++) {
                System.out.print(mat[f][c]+" ");
            }
            System.out.println();
        }
    }
}
```

```
public static void main(String[] ar) {  
    Matriz1 ma=new Matriz1();  
    ma.cargar();  
    ma.imprimir();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class Matriz1 {
4     private Scanner teclado;
5     private int[][] mat;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         mat=new int[3][5];
10        for(int f=0;f<3;f++) {
11            for(int c=0;c<5;c++) {
12                System.out.print("Introduce componente:");
13                mat[f][c]=teclado.nextInt();
14            }
15        }
16    }
17
18    public void imprimir() {
19        for(int f=0;f<3;f++) {
20            for(int c=0;c<5;c++) {
21                System.out.print(mat[f][c]+" ");
22            }
23            System.out.println();
24        }
25    }
26
27    public static void main(String[] ar) {
28        Matriz1 ma=new Matriz1();
29        ma.cargar();
30        ma.imprimir();
31    }
32 }
```



Problems



@ Javadoc



De

<terminated> Matriz1 [Java Applica

Introduce componente:15

Introduce componente:2

Introduce componente:65

Introduce componente:958

Introduce componente:7

Introduce componente:0

Introduce componente:58

Introduce componente:612

Introduce componente:14

Introduce componente:326

Introduce componente:2

Introduce componente:548

Introduce componente:4

Introduce componente:7

Introduce componente:8

15 2 65 958 7

0 58 612 14 326

2 548 4 7 8

Para definir una matriz debemos antecederle los corchetes abiertos y cerrados dos veces:

```
private int[][] mat;
```

De esta forma el compilador de Java puede diferenciar los vectores de las matrices.

Para crear la matriz, es decir hacer la reserva de espacio de todas sus componentes debemos utilizar el operador new y mediante dos subíndices indicamos la cantidad de filas y columnas que tendrá la matriz:

```
mat=new int[3][5];
```

Después pasamos a cargar sus 15 componentes (cada fila almacena 5 componentes y tenemos 3 filas)

Lo más cómodo es utilizar un for anidado, el primer for que incrementa el contador f lo utilizamos para recorrer las filas y el contador interno llamado c lo utilizamos para recorrer las columnas.

Cada vez que se repite en forma completa el for interno se carga una fila completa, primero se carga la fila cero en forma completa, luego la fila uno y finalmente la fila 2.

Siempre que accedemos a una posición de la matriz debemos disponer dos subíndices que hagan referencia a la fila y columna `mat[f][c]`):

```
for(int f=0;f<3;f++) {  
    for(int c=0;c<5;c++) {  
        System.out.print("Ingresa componente:");  
        mat[f][c]=teclado.nextInt();  
    }  
}
```

Para imprimir la matriz de forma similar utilizamos dos for para acceder a cada elemento de la matriz:

```
for(int f=0;f<3;f++) {  
    for(int c=0;c<5;c++) {  
        System.out.print(mat[f][c]+" ");  
    }  
    System.out.println();  
}
```


Cada vez que se ejecuta todas las vueltas del for interno tenemos en pantalla una fila completa de la matriz, por eso pasamos a ejecutar un salto de línea (con esto logramos que en pantalla los datos aparezcan en forma matricial):

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

Problema 2:

Crear y cargar una matriz de 4 filas por 4 columnas. Imprimir la diagonal principal.

| | | | |
|---|---|---|---|
| X | - | - | - |
| - | X | - | - |
| - | - | X | - |
| - | - | - | X |

Programa:

```
import java.util.Scanner;
public class Matriz2 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        mat=new int[4][4];
        for(int f=0;f<4;f++) {
            for(int c=0;c<4;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void imprimirDiagonalPrincipal() {
        for(int k=0;k<4;k++) {
            System.out.print(mat[k][k]+" ");
        }
    }
}
```

```
public static void main(String[] ar) {  
    Matriz2 ma=new Matriz2();  
    ma.cargar();  
    ma.imprimirDiagonalPrincipal();  
}  
}
```

```
1
2 import java.util.Scanner;
3 public class Matriz2 {
4     private Scanner teclado;
5     private int[][] mat;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         mat=new int[4][4];
10        for(int f=0;f<4;f++) {
11            for(int c=0;c<4;c++) {
12                System.out.print("Introduce componente:");
13                mat[f][c]=teclado.nextInt();
14            }
15        }
16    }
17
18    public void imprimirDiagonalPrincipal() {
19        for(int k=0;k<4;k++) {
20            System.out.print(mat[k][k]+" ");
21        }
22    }
23
24    public static void main(String[] ar) {
25        Matriz2 ma=new Matriz2();
26        ma.cargar();
27        ma.imprimirDiagonalPrincipal();
28    }
29 }
30
```

<terminated> Matriz2 [Java Applica

Introduce componente:48

Introduce componente:5

Introduce componente:65

Introduce componente:30

Introduce componente:2

Introduce componente:4

Introduce componente:85

Introduce componente:123

Introduce componente:369

Introduce componente:548

Introduce componente:456

Introduce componente:5

Introduce componente:23

Introduce componente:5

Introduce componente:75

Introduce componente:4

|48 4 456 4

La definición, creación y carga de la matriz no varían con el ejemplo anterior.

Para imprimir la diagonal principal de la matriz lo más conveniente es utilizar un for que se repita 4 veces y disponer como subíndice dicho contador (los elementos de la diagonal principal coinciden los valores de la fila y columna):

```
for(int k=0;k<4;k++) {  
    System.out.print(mat[k][k]+" ");  
}
```

Problema 3:

Crear y cargar una matriz de 3 filas por 4 columnas. Imprimir la primera fila. Imprimir la última fila e imprimir la primer columna.

Programa:

```
import java.util.Scanner;
public class Matriz3 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        mat=new int[3][4];
        for(int f=0;f<3;f++) {
            for(int c=0;c<4;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void primerFila() {
        System.out.println("Primer fila de la matriz:");
        for(int c=0;c<4;c++) {
            System.out.println(mat[0][c]);
        }
    }
}
```

```
public void ultimaFila() {  
    System.out.println("Ultima fila de la matriz:");  
    for(int c=0;c<4;c++) {  
        System.out.println(mat[2][c]);  
    }  
}
```

```
public void primerColumna() {  
    System.out.println("Primer columna:");  
    for(int f=0;f<3;f++) {  
        System.out.println(mat[f][0]);  
    }  
}
```

```
public static void main(String[] ar) {  
    Matriz3 ma=new Matriz3();  
    ma.cargar();  
    ma.primerFila();  
    ma.ultimaFila();  
    ma.primerColumna();  
}
```



```
1
2 import java.util.Scanner;
3 public class Matriz3 {
4     private Scanner teclado;
5     private int[][] mat;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         mat=new int[3][4];
10        for(int f=0;f<3;f++) {
11            for(int c=0;c<4;c++) {
12                System.out.print("Introduce componente:");
13                mat[f][c]=teclado.nextInt();
14            }
15        }
16    }
17
18    public void primerFila() {
19        System.out.println("Primer fila de la matriz:");
20        for(int c=0;c<4;c++) {
21            System.out.println(mat[0][c]);
22        }
23    }
24
25    public void ultimaFila() {
26        System.out.println("Ultima fila de la matriz:");
27        for(int c=0;c<4;c++) {
28            System.out.println(mat[2][c]);
29        }
30    }
31
32    public void primerColumna() {
33        System.out.println("Primer columna:");
34        for(int f=0;f<3;f++) {
35            System.out.println(mat[f][0]);
```

<terminated> Matriz3 [Java Applicat

```
Introduce componente:47
Introduce componente:5
Introduce componente:6
Introduce componente:3
Introduce componente:2
Introduce componente:0
Introduce componente:78
Introduce componente:36
Introduce componente:25
Introduce componente:34
Introduce componente:91
Introduce componente:20
Primer fila de la matriz:
47
5
6
3
Ultima fila de la matriz:
25
34
91
20
Primer columna:
47
2
25
```

Creamos una matriz de 3 filas y 4 columnas:

```
mat=new int[3][4];
```

Luego de cargarla el primer método que codificamos es el que imprime la primer fila. Disponemos un for para recorrer las columnas, ya que la fila siempre será la cero. Como son cuatro los elementos de la primera fila el for se repite esta cantidad de veces:

```
System.out.println("Primer fila de la matriz:");  
for(int c=0;c<4;c++) {  
    System.out.println(mat[0][c]);  
}
```

Para imprimir la última fila el algoritmo es similar, disponemos un for que se repita 4 veces y en el subíndice de la fila disponemos el valor 2 (ya que la matriz tiene 3 filas):

```
System.out.println("Ultima fila de la matriz:");  
for(int c=0;c<4;c++) {  
    System.out.println(mat[2][c]);  
}
```

Para imprimir la primer columna el for debe repetirse 3 veces ya que la matriz tiene 3 filas. Dejamos constante el subíndice de la columna con el valor cero:

```
        System.out.println("Primera columna:");  
for(int f=0;f<3;f++) {  
    System.out.println(mat[f][0]);  
}
```

Problema propuesto

Crear una matriz de 2 filas y 5 columnas. Realizar la carga de componentes por columna (es decir primero introducir toda la primer columna, luego la segunda columna y así sucesivamente)

Imprimir luego la matriz.

23 - Matrices (cantidad de filas y columnas)

Como hemos visto para definir y crear la matriz utilizamos la siguiente sintaxis:

```
int[][] mat;
```

Creación:

```
mat=new int[3][4];
```

Como **las matrices son objetos en Java** disponemos por un lado del **atributo length** que **almacena la cantidad de filas de la matriz**:

```
System.out.println("Cantidad de filas de la matriz:" + mat.length);
```

También podemos preguntarle a cada fila de la matriz la cantidad de elementos que almacena:

```
System.out.println("Cantidad de elementos de la primer fila:" + mat[0].length);
```

Problema 1:

Crear una matriz de $n * m$ filas (cargar n y m por teclado) Imprimir la matriz completa y la última fila.

Programa:

```
import java.util.Scanner;
public class Matriz5 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        System.out.print("Cuántas filas tiene la matriz:");
        int filas=teclado.nextInt();
        System.out.print("Cuántas columnas tiene la matriz:");
        int columnas=teclado.nextInt();
        mat=new int[filas][columnas];
        for(int f=0;f<mat.length;f++) {
            for(int c=0;c<mat[f].length;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }
}
```

```
public void imprimir() {  
    for(int f=0;f<mat.length;f++) {  
        for(int c=0;c<mat[f].length;c++) {  
            System.out.print(mat[f][c]+" ");  
        }  
        System.out.println();  
    }  
}
```

```
public void imprimirUltimaFila() {  
    System.out.println("Ultima fila");  
    for(int c=0;c<mat[mat.length-1].length;c++) {  
        System.out.print(mat[mat.length-1][c]+" ");  
    }  
}
```

```
public static void main(String[] ar) {  
    Matriz5 ma=new Matriz5();  
    ma.cargar();  
    ma.imprimir();  
    ma.imprimirUltimaFila();  
}
```



```
1
2 import java.util.Scanner;
3 public class Matriz5 {
4     private Scanner teclado;
5     private int[][] mat;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         System.out.print("Cuántas filas tiene la matriz:");
10        int filas=teclado.nextInt();
11        System.out.print("Cuántas columnas tiene la matriz:");
12        int columnas=teclado.nextInt();
13        mat=new int[filas][columnas];
14        for(int f=0;f<mat.length;f++) {
15            for(int c=0;c<mat[f].length;c++) {
16                System.out.print("Introduce componente:");
17                mat[f][c]=teclado.nextInt();
18            }
19        }
20    }
21
22    public void imprimir() {
23        for(int f=0;f<mat.length;f++) {
24            for(int c=0;c<mat[f].length;c++) {
25                System.out.print(mat[f][c]+" ");
26            }
27            System.out.println();
28        }
29    }
30
31    public void imprimirUltimaFila() {
32        System.out.println("Ultima fila");
33        for(int c=0;c<mat[mat.length-1].length;c++) {
34            System.out.print(mat[mat.length-1][c]+" ");
35        }
36    }
37
38    public static void main(String[] ar) {
39        Matriz5 ma=new Matriz5();
```



Problems

@ Javadoc



Declaration

<terminated> Matriz5 [Java Application] C:\Pro

Cuántas filas tiene la matriz:3

Cuántas columnas tiene la matriz:3

Introduce componente:7

Introduce componente:8

Introduce componente:3

Introduce componente:25

Introduce componente:65

Introduce componente:39

Introduce componente:51

Introduce componente:84

Introduce componente:52

7 8 3

25 65 39

51 84 52

Última fila

51 84 52

En este ejemplo cada vez que se ejecute el programa el tamaño de la matriz lo define el usuario, para ello introducimos por teclado dos enteros y seguidamente procedemos a crear la matriz con dichos valores:

```
System.out.print("Cuántas filas tiene la matriz:");  
int filas = teclado.nextInt();  
System.out.print("Cuántas columnas tiene la matriz:");  
int columnas = teclado.nextInt();  
mat = new int[filas][columnas];
```

Ahora las estructuras repetitivas las acotamos preguntando a la misma matriz la cantidad de filas y la cantidad de elementos de cada **fila**(**mat.length** almacena la cantidad de filas de la matriz y **mat[f].length** cuando **f** vale cero accedemos a la cantidad de elementos de la fila cero y así sucesivamente para cada valor de **f**):

```
for(int f=0;f<mat.length;f++) {  
    for(int c=0;c<mat[f].length;c++) {  
        System.out.print("Introduce componente:");  
        mat[f][c]=teclado.nextInt();  
    }  
}
```

El algoritmo de impresión es idéntico al visto anteriormente con la modificación de las condiciones de los for:

```
public void imprimir() {  
    for(int f=0;f<mat.length;f++) {  
        for(int c=0;c<mat[f].length;c++) {  
            System.out.print(mat[f][c]+" ");  
        }  
        System.out.println();  
    }  
}
```

Para imprimir la última fila debemos disponer un valor fijo en el subíndice de la fila (en este caso no podemos disponer un número fijo sino preguntarle a la misma matriz la cantidad de filas y restarle uno ya que las filas comienzan a numerarse a partir de cero: `mat[mat.length-1][c]`)

También la condición del for debemos acceder al atributo `length` de la última fila `mat[mat.length-1].length`

```
for(int c=0;c<mat[mat.length-1].length;c++) {  
    System.out.print(mat[mat.length-1][c]+" ");  
}
```

Problema 2:

Crear una matriz de $n * m$ filas (cargar n y m por teclado) Imprimir el mayor elemento y la fila y columna donde se almacena.

Programa:

```
import java.util.Scanner;
public class Matriz6 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        System.out.print("Cuántas fila tiene la matriz:");
        int filas=teclado.nextInt();
        System.out.print("Cuántas columnas tiene la matriz:");
        int columnas=teclado.nextInt();
        mat=new int[filas][columnas];
        for(int f=0;f<mat.length;f++) {
            for(int c=0;c<mat[f].length;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }
}
```

```
public void imprimirMayor() {  
    int mayor=mat[0][0];  
    int filamay=0;  
    int columnamay=0;  
    for(int f=0;f<mat.length;f++) {  
        for(int c=0;c<mat[f].length;c++) {  
            if (mat[f][c]>mayor) {  
                mayor=mat[f][c];  
                filamay=f;  
                columnamay=c;  
            }  
        }  
    }  
    System.out.println("El elemento mayor es:"+mayor);  
    System.out.println("Se encuentra en la fila:"+filamay+ " y en la columna:  
"+columnamay);  
}
```

```
public static void main(String[] ar) {  
    Matriz6 ma=new Matriz6();  
    ma.cargar();  
    ma.imprimirMayor();  
}
```

```
1
2 import java.util.Scanner;
3 public class Matriz6 {
4     private Scanner teclado;
5     private int[][] mat;
6
7     public void cargar() {
8         teclado=new Scanner(System.in);
9         System.out.print("Cuantas fila tiene la matriz:");
10        int filas=teclado.nextInt();
11        System.out.print("Cuantas columnas tiene la matriz:");
12        int columnas=teclado.nextInt();
13        mat=new int[filas][columnas];
14        for(int f=0;f<mat.length;f++) {
15            for(int c=0;c<mat[f].length;c++) {
16                System.out.print("Introduce componente:");
17                mat[f][c]=teclado.nextInt();
18            }
19        }
20    }
21
22    public void imprimirMayor() {
23        int mayor=mat[0][0];
24        int filamay=0;
25        int columnamay=0;
26        for(int f=0;f<mat.length;f++) {
27            for(int c=0;c<mat[f].length;c++) {
28                if (mat[f][c]>mayor) {
29                    mayor=mat[f][c];
30                    filamay=f;
31                    columnamay=c;
32                }
33            }
34        }
35        System.out.println("El elemento mayor es:"+mayor);
36        System.out.println("Se encuentra en la fila:"+filamay+ " y en la columna: "+columnamay);
37    }
38
39    public static void main(String[] ar) {
```




Problems



@ Javadoc



Declaration



Console



<terminated> Matriz6 [Java Application] C:\Program Files\Java

Cuántas fila tiene la matriz:3

Cuántas columnas tiene la matriz:3

Introduce componente:4

Introduce componente:85

Introduce componente:74

Introduce componente:21

Introduce componente:36

Introduce componente:59

Introduce componente:51

Introduce componente:42

Introduce componente:36

El elemento mayor es:85

Se encuentra en la fila:0 y en la columna: 1

Para obtener el mayor elemento de la matriz y la fila y columna donde se ubica debemos inicializar una variable mayor con el elemento de la fila cero y columna cero (esto lo hacemos suponiendo que en dicha posición se almacena el mayor):

```
int mayor=mat[0][0];  
int filamay=0;  
int columnamay=0;
```

Luego mediante dos for recorreremos todos los elementos de la matriz y cada vez que encontramos un elemento mayor al actual procedemos a actualizar la variable mayor y la posición donde se almacena:

```
for(int f=0;f<mat.length;f++) {  
    for(int c=0;c<mat[f].length;c++) {  
        if (mat[f][c]>mayor) {  
            mayor=mat[f][c];  
            filamay=f;  
            columnamay=c;  
        }  
    }  
}
```

Problemas propuestos

1. Crear una matriz de $n * m$ filas (cargar n y m por teclado) Intercambiar la primer fila con la segunda. Imprimir luego la matriz.
2. Crear una matriz de $n * m$ filas (cargar n y m por teclado) Imprimir los cuatro valores que se encuentran en los vértices de la misma (`mat[0][0]` etc.)

24 - Matrices y vectores paralelos

Dependiendo de la complejidad del problema podemos necesitar el empleo de vectores y matrices paralelos.

Problema 1:

Se tiene la siguiente información:

- Nombres de 4 empleados.
- Ingresos en concepto de sueldo, cobrado por cada empleado, en los últimos 3 meses.

Confeccionar el programa para:

- Realizar la carga de la información mencionada.
- Generar un vector que contenga el ingreso acumulado en sueldos en los últimos 3 meses para cada empleado.
- Mostrar por pantalla el total pagado en sueldos a todos los empleados en los últimos 3 meses
- Obtener el nombre del empleado que tuvo el mayor ingreso acumulado

| empleados | sueldos | | | sueldostot |
|-----------|---------|-----|-----|------------|
| Marcos | 540 | 540 | 760 | |
| Ana | 200 | 220 | 250 | |
| Luis | 760 | 760 | 760 | |
| María | 605 | 799 | 810 | |

Programa:

```
import java.util.Scanner;
public class Matriz9 {
    private Scanner teclado;
    private String[] empleados;
    private int[][] sueldos;
    private int[] sueldostot;

    public void cargar() {
        teclado=new Scanner(System.in);
        empleados=new String[4];
        sueldos=new int[4][3];
        for(int f=0;f<empleados.length;f++){
            System.out.print("Introduce el nombre del empleado:");
            empleados[f]=teclado.next();
            for(int c=0;c<sueldos[f].length;c++) {
                System.out.print("Introduce sueldo:");
                sueldos[f][c]=teclado.nextInt();
            }
        }
    }
}
```

```
public void calcularSumaSuealdos() {  
    suealdostot=new int[4];  
    for(int f=0;f<suealdos.length;f++) {  
        int suma=0;  
        for(int c=0;c<suealdos[f].length;c++) {  
            suma=suma+suealdos[f][c];  
        }  
        suealdostot[f]=suma;  
    }  
}
```

```
public void imprimirTotalPagado() {  
    System.out.println("Total de suealdos pagados por empleado.");  
    for(int f=0;f<suealdostot.length;f++) {  
        System.out.println(empleados[f]+" - "+suealdostot[f]);  
    }  
}
```

```
public void empleadoMayorSueldo() {  
    int may=sueldostot[0];  
    String nom=empleados[0];  
    for(int f=0;f<sueldostot.length;f++) {  
        if (sueldostot[f]>may) {  
            may=sueldostot[f];  
            nom=empleados[f];  
        }  
    }  
    System.out.println("El empleado con mayor sueldo es "+ nom + " que tiene un sueldo  
de "+may);  
}
```

```
public static void main(String[] ar){  
    Matriz9 ma=new Matriz9();  
    ma.cargar();  
    ma.calcularSumaSueldos();  
    ma.imprimirTotalPagado();  
    ma.empleadoMayorSueldo();  
}  
}
```


Para resolver este problema lo primero que hacemos es definir una matriz donde se almacenarán los sueldos mensuales de cada empleado, un vector de tipo String donde almacenaremos los nombre de cada empleado y finalmente definimos un vector paralelo a la matriz donde almacenaremos la suma de cada fila de la matriz:

```
private String[] empleados;  
private int[][] sueldos;  
private int[] sueldostot;
```

En el método de cargar inicializamos el vector con los nombres de los empleados y la matriz paralela donde se almacenan los últimos tres sueldos (previo a cargar procedemos a crear el vector y la matriz):

```
empleados=new String[4];  
sueldos=new int[4][3];  
for(int f=0;f<empleados.length;f++){  
    System.out.print("Ingrese el nombre del empleado:");  
    empleados[f]=teclado.next();  
    for(int c=0;c<sueldos[f].length;c++) {  
        System.out.print("Ingrese sueldo:");  
        sueldos[f][c]=teclado.nextInt();  
    }  
}
```

El método sumar sueldos crea el vector donde se almacenará la suma de cada fila de la matriz. Mediante dos for recorreremos toda la matriz y sumamos cada fila:

```
        sueldostot=new int[4];
for(int f=0;f<sueldos.length;f++) {
    int suma=0;
    for(int c=0;c<sueldos[f].length;c++) {
        suma=suma+sueldos[f][c];
    }
    sueldostot[f]=suma;
}
```

El método imprimirTotalPagado tiene por objetivo mostrar los dos vectores (el de nombre de los empleados y el que almacena la suma de cada fila de la matriz):

```
for(int f=0;f<sueldostot.length;f++) {
    System.out.println(empleados[f]+" - "+sueldostot[f]);
}
```

Por último para obtener el nombre del empleado con mayor sueldo acumulado debemos inicializar dos variables auxiliares con el primer elemento del vector de empleados y en otra auxiliar guardamos la primer componente del vector sueldostot:

```
int may=sueldostot[0];
String nom=empleados[0];
for(int f=0;f<sueldostot.length;f++) {
    if (sueldostot[f]>may) {
        may=sueldostot[f];
        nom=empleados[f];
    }
}
System.out.println("El empleado con mayor sueldo es "+ nom + " que tiene un
sueldo de "+may);
```

Problema propuesto

Se desea saber la temperatura media trimestral de cuatro países. Para ello se tiene como dato las temperaturas medias mensuales de dichos países.

Se debe introducir el nombre del país y seguidamente las tres temperaturas medias mensuales.

Seleccionar las estructuras de datos adecuadas para el almacenamiento de los datos en memoria.

a - Cargar por teclado los nombres de los países y las temperaturas medias mensuales.

b - Imprimir los nombres de los países y las temperaturas medias mensuales de las mismas.

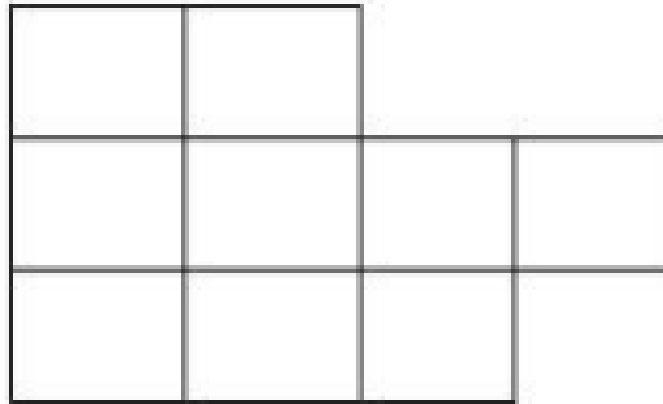
c - Calcular la temperatura media trimestral de cada país.

d - Imprimir los nombres de los países y las temperaturas medias trimestrales.

e - Imprimir el nombre del país con la temperatura media trimestral mayor.

25 - Matrices irregulares

Java nos permite crear matrices irregulares. Se dice que una **matriz es irregular si la cantidad de elementos de cada fila varía**. Luego podemos imaginar una matriz irregular:



Como podemos ver la fila cero tiene reservado dos espacios, la fila uno reserva cuatro espacios y la última fila reserva espacio para tres componentes.

Para crear la matriz irregular del gráfico:

La declaración es la misma que para matrices regulares:

```
int [][] mat;
```

Primero creamos la cantidad de filas dejando vacío el espacio que indica la cantidad de columnas:

```
mat=new int[3][];
```

Luego debemos ir creando cada fila de la matriz indicando la cantidad de elementos de la respectiva fila:

```
mat[0]=new int[2];  
mat[1]=new int[4];  
mat[2]=new int[3];
```

Luego la forma para acceder a sus componentes es similar a las matrices regulares, siempre teniendo en cuenta y validando que exista dicha componente:

```
mat[0][0]=120;
```

Dará un error si queremos cargar el tercer componente de la fila cero (esto debido a que no existe):

```
mat[0][2]=230;
```

Luego si queremos saber la cantidad de filas que tiene la matriz:

```
Sytem.out.println(mat.length);
```

Si queremos saber la cantidad de elementos de una determinada fila:

```
Sytem.out.println("Cantidad de elementos de la fila 0:"+mat[0].length);
```

```
Sytem.out.println("Cantidad de elementos de la fila 1:"+mat[1].length);
```

```
Sytem.out.println("Cantidad de elementos de la fila 2:"+mat[2].length);
```

Problema 1:

Confeccionaremos un programa que permita crear una matriz irregular y luego imprimir la matriz en forma completa.

Programa:

```
import java.util.Scanner;
public class MatrizIrregular1 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        System.out.print("Cuantas fila tiene la matriz:");
        int filas=teclado.nextInt();
        mat=new int[filas][];
        for(int f=0;f<mat.length;f++) {
            System.out.print("Cuantas elementos tiene la fila " + f + ":");
            int elementos=teclado.nextInt();
            mat[f]=new int[elementos];
            for(int c=0;c<mat[f].length;c++) {
                System.out.print("Introduce componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }
}
```



```
public void imprimir() {  
    for(int f=0;f<mat.length;f++) {  
        for(int c=0;c<mat[f].length;c++) {  
            System.out.print(mat[f][c]+" ");  
        }  
        System.out.println();  
    }  
}
```

```
public static void main(String[] ar) {  
    MatrizIrregular1 ma=new MatrizIrregular1();  
    ma.cargar();  
    ma.imprimir();  
}
```

```
1  import java.util.Scanner;
2  public class MatrizIrregular1 {
3      private Scanner teclado;
4      private int[][] mat;
5
6  public void cargar() {
7      teclado=new Scanner(System.in);
8      System.out.print("Cuantas fila tiene la matriz:");
9      int filas=teclado.nextInt();
10     mat=new int[filas][];
11     for(int f=0;f<mat.length;f++) {
12         System.out.print("Cuantas elementos tiene la fila " + f + ":");
13         int elementos=teclado.nextInt();
14         mat[f]=new int[elementos];
15         for(int c=0;c<mat[f].length;c++) {
16             System.out.print("Introduce componente:");
17             mat[f][c]=teclado.nextInt();
18         }
19     }
20 }
21
22 public void imprimir() {
23     for(int f=0;f<mat.length;f++) {
24         for(int c=0;c<mat[f].length;c++) {
25             System.out.print(mat[f][c]+" ");
26         }
27         System.out.println();
28     }
29 }
30
31 public static void main(String[] ar) {
32     MatrizIrregular1 ma=new MatrizIrregular1();
33     ma.cargar();
34     ma.imprimir();
35 }
36 }
37
```



Problems

@ Javadoc



Declaration



<terminated> MatrizIrregular1 [Java Application]

Cuantas fila tiene la matriz:4

Cuantas elementos tiene la fila 0:3

Introduce componente:1

Introduce componente:2

Introduce componente:3

Cuantas elementos tiene la fila 1:2

Introduce componente:6

Introduce componente:5

Cuantas elementos tiene la fila 2:5

Introduce componente:4

Introduce componente:3

Introduce componente:6

Introduce componente:5

Introduce componente:9

Cuantas elementos tiene la fila 3:4

Introduce componente:7

Introduce componente:6

Introduce componente:2

Introduce componente:5

1 2 3

6 5

4 3 6 5 9

7 6 2 5

Primero creamos la cantidad de filas que tendrá la matriz (en los corchetes para las columnas no disponemos valor):

```
System.out.print("Cuántas fila tiene la matriz:");  
int filas=teclado.nextInt();  
mat=new int[filas][];
```

Dentro del primer for pedimos que se introduzca la cantidad de elementos que tendrá cada fila y utilizamos el operador new nuevamente, pero en este caso se están creando cada fila de la matriz (Java trata a cada fila como un vector):

```
for(int f=0;f<mat.length;f++) {  
    System.out.print("Cuántos elementos tiene la fila " + f + ":");  
    int elementos=teclado.nextInt();  
    mat[f]=new int[elementos];
```

Dentro del for interno hacemos la carga de las componentes propiamente dicho de la matriz (podemos ir cargando cada fila a medida que las vamos creando):

```
    for(int c=0;c<mat[f].length;c++) {  
        System.out.print("Introduce componente:");  
        mat[f][c]=teclado.nextInt();  
    }
```

Luego imprimimos la matriz en forma completa teniendo cuidado las condiciones que disponemos en cada for.

El primer for se repite tantas veces como filas tiene la matriz: `f<mat.length` y el for interno se repite tantas veces como elementos tiene la fila que estamos procesando `c<mat [f].length`:

```
for(int f=0;f<mat.length;f++) {  
    for(int c=0;c<mat[f].length;c++) {  
        System.out.print(mat[f][c]+" ");  
    }  
    System.out.println();  
}
```

Problemas propuestos

1. Confeccionar una clase para administrar una matriz irregular de 5 filas y 1 columna la primer fila, 2 columnas la segunda fila y así sucesivamente hasta 5 columnas la última fila (crearla sin la intervención del operador)
Realizar la carga por teclado e imprimir posteriormente.
2. Confeccionar una clase para administrar los días que han faltado los 3 empleados de una empresa.

Definir un vector de 3 elementos de tipo String para cargar los nombres y una matriz irregular para cargar los días que han faltado cada empleado (cargar el número de día que faltó).

Cada fila de la matriz representan los días de cada empleado.
Mostrar los empleados con la cantidad de inasistencias.
Cuál empleado faltó menos días.

26 - Constructor de la clase

En Java **podemos definir un método que se ejecute inicialmente y de forma automática. Este método se llama constructor.**

El constructor tiene las siguientes **características:**

Tiene el mismo nombre de la clase.

Es el primer método que se ejecuta.

Se ejecuta en forma automática.

No puede retornar datos.

Se ejecuta una única vez.

Un constructor tiene por **objetivo inicializar atributos.**

Problema 1:

Se desea guardar los sueldos de 5 operarios en un vector. Realizar la creación y carga del vector en el constructor.

Programa:

```
import java.util.Scanner;
public class Operarios {
    private Scanner teclado;
    private int[] sueldos;

    public Operarios()
    {
        teclado=new Scanner(System.in);
        sueldos=new int[5];
        for(int f=0;f<5;f++) {
            System.out.print("Introduce valor del componente:");
            sueldos[f]=teclado.nextInt();
        }
    }

    public void imprimir() {
        for(int f=0;f<5;f++) {
            System.out.println(sueldos[f]);
        }
    }
}
```



```
public static void main(String[] ar) {  
    Operarios op=new Operarios();  
    op.imprimir();  
}  
}
```

```
1  import java.util.Scanner;
2  public class Operarios {
3      private Scanner teclado;
4      private int[] sueldos;
5
6      public Operarios()
7      {
8          teclado=new Scanner(System.in);
9          sueldos=new int[5];
10         for(int f=0;f<5;f++) {
11             System.out.print("Introduce valor del componente:");
12             sueldos[f]=teclado.nextInt();
13         }
14     }
15
16     public void imprimir() {
17         for(int f=0;f<5;f++) {
18             System.out.println(sueldos[f]);
19         }
20     }
21
22     public static void main(String[] ar) {
23         Operarios op=new Operarios();
24         op.imprimir();
25     }
26 }
```



Problems



@ Javadoc



Declaration



<terminated> Operarios [Java Application] C:\Pro

Introduce valor del componente:1500

Introduce valor del componente:2300

Introduce valor del componente:1000

Introduce valor del componente:2100

Introduce valor del componente:900

1500

2300

1000

2100

900

Como podemos ver es el mismo problema que resolvimos cuando vimos vectores. La diferencia es que hemos sustituido el método cargar con el constructor:

```
public Operarios()  
{  
    teclado=new Scanner(System.in);  
    sueldos=new int[5];  
    for(int f=0;f<5;f++) {  
        System.out.print("Introduce el valor de la componente:");  
        sueldos[f]=teclado.nextInt();  
    }  
}
```

Como la clase se llama Operarios el constructor tiene el mismo nombre, no disponemos la palabra clave void ya que el constructor no puede retornar datos.

La ventaja de plantear un constructor en lugar de definir un método con cualquier nombre **es que se llamará en forma automática cuando se crea un objeto de esta clase:**

```
public static void main(String[] ar) {  
    Operarios op=new Operarios();
```

Cuando se crea el objeto op se llama al método constructor.

Finalmente llamamos al método imprimir:

```
    op.imprimir();
```

Problema 2:

Plantear una clase llamada Alumno y definir como atributos su nombre y su edad. En el constructor realizar la carga de datos. Definir otros dos métodos para imprimir los datos introducidos y un mensaje si es mayor o no de edad (edad >=18)

Programa:

```
import java.util.Scanner;
public class Alumno {
    private Scanner teclado;
    private String nombre;
    private int edad;

    public Alumno() {
        teclado=new Scanner(System.in);
        System.out.print("Introduce nombre:");
        nombre=teclado.next();
        System.out.print("Introduce edad:");
        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:"+nombre);
        System.out.println("Edad:"+edad);
    }
}
```

```
public void esMayorEdad() {  
    if (edad >= 18) {  
        System.out.print(nombre + " es mayor de edad.");  
    } else {  
        System.out.print(nombre + " no es mayor de edad.");  
    }  
}
```

```
public static void main(String[] ar) {  
    Alumno alumno1 = new Alumno();  
    alumno1.imprimir();  
    alumno1.esMayorEdad();  
}
```

```
1 import java.util.Scanner;
2 public class Alumno {
3     private Scanner teclado;
4     private String nombre;
5     private int edad;
6
7     public Alumno() {
8         teclado=new Scanner(System.in);
9         System.out.print("Introduce nombre:");
10        nombre=teclado.next();
11        System.out.print("Introduce edad:");
12        edad=teclado.nextInt();
13    }
14
15    public void imprimir() {
16        System.out.println("Nombre:"+nombre);
17        System.out.println("Edad:"+edad);
18    }
19
20    public void esMayorEdad() {
21        if (edad>=18) {
22            System.out.print(nombre+" es mayor de edad.");
23        } else {
24            System.out.print(nombre+" no es mayor de edad.");
25        }
26    }
27
28    public static void main(String[] ar) {
29        Alumno alumno1=new Alumno();
30        alumno1.imprimir();
31        alumno1.esMayorEdad();
32    }
33 }
```




Problems @ Javadoc



<terminated> Alumno [Java App

Introduce nombre:luis

Introduce edad:19

Nombre:luis

Edad:19

luis es mayor de edad.

Declaramos la clase Persona, sus tres atributos y definimos el constructor con el mismo nombre de la clase:

```
public class Alumno {  
    private Scanner teclado;  
    private String nombre;  
    private int edad;  
  
    public Alumno() {  
        teclado=new Scanner(System.in);  
        System.out.print("Introduce nombre:");  
        nombre=teclado.next();  
        System.out.print("Introduce edad:");  
        edad=teclado.nextInt();  
    }  
}
```

En la main el constructor se llama en forma automática cuando creamos un objeto de la clase Alumno:

```
public static void main(String[] ar) {  
    Alumno alumno1=new Alumno();  
}
```

Los otros dos métodos deben llamarse por su nombre y en el orden que necesitamos:

```
alumno1.imprimir();  
alumno1.esMayorEdad();
```

Problema 3:

Plantear una clase TablaMultiplicar. Definir dos constructores uno con un parámetro que lleve un entero indicando que tabla queremos ver y otro con dos enteros que indique el primero que tabla queremos ver y el segundo parámetro indica cuantos términos mostrar.

Si no llega la cantidad de términos a mostrar inicializar en 10 los términos a mostrar

Programa:

```
public class TablaMultiplicar {  
    private int tabla;  
    private int terminos;  
  
    public TablaMultiplicar(int ta,int ter) {  
        tabla=ta;  
        terminos=ter;  
    }  
  
    public TablaMultiplicar(int ta) {  
        tabla=ta;  
        terminos=10;  
    }  
  
    public void imprimir() {  
        System.out.println("Tabla de multiplicar del "+tabla);  
        for(int f=1;f<=terminos;f++) {  
            int resultado=f*tabla;  
            System.out.println(tabla + "*" + f + " = " + resultado);  
        }  
    }  
}
```

```
public static void main(String[] ar) {  
    TablaMultiplicar tabla1=new TablaMultiplicar(5);  
    tabla1.imprimir();  
    TablaMultiplicar tabla2=new TablaMultiplicar(3,5);  
    tabla2.imprimir();  
}  
}
```

```
1 public class TablaMultiplicar {
2     private int tabla;
3     private int terminos;
4
5     public TablaMultiplicar(int ta,int ter) {
6         tabla=ta;
7         terminos=ter;
8     }
9
10    public TablaMultiplicar(int ta) {
11        tabla=ta;
12        terminos=10;
13    }
14
15    public void imprimir() {
16        System.out.println("Tabla de multiplicar del "+tabla);
17        for(int f=1;f<=terminos;f++) {
18            int resultado=f*tabla;
19            System.out.println(tabla + "*" + f + " = " + resultado);
20        }
21    }
22
23    public static void main(String[] ar) {
24        TablaMultiplicar tabla1=new TablaMultiplicar(5);
25        tabla1.imprimir();
26        TablaMultiplicar tabla2=new TablaMultiplicar(3,5);
27        tabla2.imprimir();
28    }
29 }
```

<terminated> TablaMultiplicar (1) [Jav

Tabla de multiplicar del 5

5*1 = 5

5*2 = 10

5*3 = 15

5*4 = 20

5*5 = 25

5*6 = 30

5*7 = 35

5*8 = 40

5*9 = 45

5*10 = 50

Tabla de multiplicar del 3

3*1 = 3

3*2 = 6

3*3 = 9

3*4 = 12

3*5 = 15

Declaramos la clase TablaMultiplicar, con dos atributos:

```
public class TablaMultiplicar {  
    private int tabla;  
    private int terminos;
```

Definimos dos constructores en la clase TablaMultiplicar:

```
    public TablaMultiplicar(int ta,int ter) {  
        tabla=ta;  
        terminos=ter;  
    }
```

```
    public TablaMultiplicar(int ta) {  
        tabla=ta;  
        terminos=10;  
    }
```

Si cuando creamos un objeto de la clase TablaMultiplicar le pasamos dos enteros, se ejecuta el primer constructor y se inicializan los atributos con los datos que llegan:

```
TablaMultiplicar tabla2=new TablaMultiplicar(3,5);
```


En cambio si llamamos al constructor que tiene un solo parámetro se ejecuta el segundo constructor:

```
TablaMultiplicar tabla1=new TablaMultiplicar(5);
```

Tengamos en cuenta que es un problema que no requiere que se carguen valores por teclado, luego por eso no definimos un objeto de la clase Scanner.

En la main definimos dos objetos de la clase TablaMultiplicar para comprobar los resultados de llamar a uno u otro constructor:

```
public static void main(String[] ar) {  
    TablaMultiplicar tabla1=new TablaMultiplicar(5);  
    tabla1.imprimir();  
    TablaMultiplicar tabla2=new TablaMultiplicar(3,5);  
    tabla2.imprimir();  
}
```

La definición de múltiples constructores hace que nuestra clase se adapte a más situaciones.

Problemas propuestos

1. Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. En el constructor cargar los atributos y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)
2. Implementar la clase operaciones. Se deben cargar dos valores enteros en el constructor, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

27 - Clase String

La clase **String** está orientada a manejar cadenas de caracteres. Hasta este momento hemos utilizado algunos métodos de la clase String (equals, compareTo)

Ahora veremos otro conjunto de métodos de uso común de la clase String:

Métodos

- boolean equals(String s1)

Como vimos el método **equals** retorna true si el contenido de caracteres del parámetro **s1** es exactamente igual a la cadena de caracteres del objeto que llama al método **equals**.

- boolean equalsIgnoreCase(String s1)

El funcionamiento es casi exactamente **igual** que el método **equals** con la **diferencia** que **no tiene en cuenta mayúsculas y minúsculas** (si comparamos 'Ana' y 'ana' luego el método **equalsIgnoreCase** retorna true)

- `int compareTo(String s1)`

Este método retorna un 0 si el contenido de s1 es exactamente igual al String contenido por el objeto que llama al método compareTo. Retorna un valor >0 si el contenido del String que llama al método compareTo es mayor alfabéticamente al parámetro s1.

- `char charAt(int pos)`

Retorna un caracter del String, llega al método la posición del caracter a extraer.

- `int length()`

Retorna la cantidad de caracteres almacenados en el String.

- `String substring(int pos1,int pos2)`

Retorna un substring a partir de la posición indicada en el parámetro pos1 hasta la posición pos2 sin incluir dicha posición.

- `int indexOf(String s1)`

Retorna -1 si el String que le pasamos como parámetro no está contenida en la cadena del objeto que llama al método. En caso que se encuentre contenido el String s1 retorna la posición donde comienza a repetirse.

- `String toUpperCase()`

Retorna un String con el contenido convertido todo a mayúsculas.

- `String toLowerCase()`

Retorna un String con el contenido convertido todo a minúsculas.

Problema 1:

Confeccionar una clase que solicite introducir dos String y luego emplee los métodos más comunes de la clase String.

Programa:

```
import java.util.Scanner;
public class Cadena1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String cad1;
        String cad2;
        System.out.print("Introduce la primer cadena:");
        cad1=teclado.nextLine();
        System.out.print("Introduce la segunda cadena:");
        cad2=teclado.nextLine();
        if (cad1.equals(cad2)==true) {
            System.out.println(cad1+" es exactamente igual a "+cad2);
        } else {
            System.out.println(cad1+" no es exactamente igual a "+cad2);
        }
        if (cad1.equalsIgnoreCase(cad2)==true) {
            System.out.println(cad1+" es igual a "+cad2+" sin tener en cuenta mayúsculas/minúsculas");
        } else {
            System.out.println(cad1+" no es igual a "+cad2+" sin tener en cuenta mayúsculas/minúsculas");
        }
    }
}
```

```
if (cad1.compareTo(cad2)==0) {  
    System.out.println(cad1+" es exactamente igual a "+cad2);  
} else {  
    if (cad1.compareTo(cad2)>0) {  
        System.out.println(cad1+ " es mayor alfabéticamente que "+cad2);  
    } else {  
        System.out.println(cad2+ " es mayor alfabéticamente que "+cad1);  
    }  
}
```

```
char carac1=cad1.charAt(0);
    System.out.println("El primer caracter de "+cad1+" es "+carac1);
    int largo=cad1.length();
    System.out.println("El largo del String "+cad1+" es "+largo);
    String cad3=cad1.substring(0,3);
    System.out.println("Los primeros tres caracteres de "+cad1+" son "+cad3);
    int posi=cad1.indexOf(cad2);
    if (posi== -1) {
        System.out.println(cad2+" no está contenido en "+cad1);
    } else {
        System.out.println(cad2+" está contenido en "+cad1+" a partir de la posición "+posi);
    }
    System.out.println(cad1+" convertido a mayúsculas es "+cad1.toUpperCase());
    System.out.println(cad1+" convertido a minúsculas es "+cad1.toLowerCase());
}
```



```
1 import java.util.Scanner;
2 public class Cadena1 {
3     public static void main(String[] ar) {
4         Scanner teclado=new Scanner(System.in);
5         String cad1;
6         String cad2;
7         System.out.print("Introduce la primer cadena:");
8         cad1=teclado.nextLine();
9         System.out.print("Introduce la segunda cadena:");
10        cad2=teclado.nextLine();
11        if (cad1.equals(cad2)==true) {
12            System.out.println(cad1+" es exactamente igual a "+cad2);
13        } else {
14            System.out.println(cad1+" no es exactamente igual a "+cad2);
15        }
16        if (cad1.equalsIgnoreCase(cad2)==true) {
17            System.out.println(cad1+" es igual a "+cad2+" sin tener en cuenta mayúsculas/minúsculas");
18        } else {
19            System.out.println(cad1+" no es igual a "+cad2+" sin tener en cuenta mayúsculas/minúsculas");
20        }
21        if (cad1.compareTo(cad2)==0) {
22            System.out.println(cad1+" es exactamente igual a "+cad2);
23        } else {
24            if (cad1.compareTo(cad2)>0) {
25                System.out.println(cad1+" es mayor alfabéticamente que "+cad2);
26            } else {
27                System.out.println(cad2+" es mayor alfabéticamente que "+cad1);
28            }
29        }
30        char carac1=cad1.charAt(0);
31        System.out.println("El primer caracter de "+cad1+" es "+carac1);
32        int largo=cad1.length();
33        System.out.println("El largo del String "+cad1+" es "+largo);
34        String cad3=cad1.substring(0,3);
35        System.out.println("Los primeros tres caracteres de "+cad1+" son "+cad3);
36        int posi=cad1.indexOf(cad2);
37        if (posi==-1) {
38            System.out.println(cad2+" no está contenido en "+cad1);
39        } else {
```

Problems @ Javadoc Declaration Console X

<terminated> Cadena1 [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (8 ene 2022 17:20:42 – 17:21:26)

Introduce la primer cadena:Estudia y aprobaras

Introduce la segunda cadena:No pain no gain

Estudia y aprobaras no es exactamente igual a No pain no gain

Estudia y aprobaras no es igual a No pain no gain sin tener en cuenta mayúsculas/minúsculas

No pain no gain es mayor alfabéticamente que Estudia y aprobaras

El primer caracter de Estudia y aprobaras es E

El largo del String Estudia y aprobaras es 19

Los primeros tres caracteres de Estudia y aprobaras son Est

No pain no gain no está contenido en Estudia y aprobaras

Estudia y aprobaras convertido a mayúsculas es ESTUDIA Y APROBARAS

Estudia y aprobaras convertido a minúsculas es estudia y aprobaras

Para cargar los dos String utilizamos en este caso el método `nextLine` para permitir introducir espacios en blanco:

```
System.out.print("Introduce la primer cadena:");  
cad1=teclado.nextLine();  
System.out.print("Introduce la segunda cadena:");  
cad2=teclado.nextLine();
```

Problemas propuestos

1. Realizar una clase, que permita cargar una dirección de mail en el constructor, luego en otro método mostrar un mensaje si contiene el caracter '@'.

2. Cargar un String por teclado e implementar los siguientes métodos:

a) Imprimir la primera mitad de los caracteres de la cadena.

b) Imprimir el último caracter.

c) Imprimirlo en forma inversa.

d) Imprimir cada caracter del String separado con un guión.

e) Imprimir la cantidad de vocales almacenadas.

f) Implementar un método que verifique si la cadena se lee igual de izquierda a derecha tanto como de derecha a izquierda (ej. neuquen se lee igual en las dos direcciones)

3. Desarrollar un programa que solicite la carga de una clave. La clase debe tener dos métodos uno para la carga y otro que muestre si la clave es la correcta (la clave a comparar es "123abc")
4. Confeccionar un programa que permita cargar los nombres de 5 personas y sus mail, luego implementar los siguientes métodos:
 - a) Mostrar por pantalla los datos.
 - b) Consulta del mail introduciendo su nombre.
 - c) Mostrar los mail que no tienen el carácter @.

5. Codifique un programa que permita cargar una oración por teclado, luego mostrar cada palabra introducida en una línea distinta.

Por ejemplo si cargo: La mañana está fría.

Debe aparecer:

La
mañana
está
fría.

28 - Colaboración de clases

Normalmente **un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.**

Plantearemos un problema separando las actividades en dos clases.

Problema 1:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositada.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Debemos definir los atributos y los métodos de cada clase:

Cliente

atributos

nombre

monto

métodos

constructor

depositar

extraer

retornarMonto

Banco

atributos

3 Cliente (3 objetos de la clase Cliente)

métodos

constructor

operar

depositosTotales

Creamos un proyecto en Eclipse llamado: Proyecto1 y dentro del proyecto creamos dos clases llamadas: Cliente y Banco.

Programa:

```
public class Cliente {  
    private String nombre;  
    private int monto;  
  
    public Cliente(String nom) {  
        nombre=nom;  
        monto=0;  
    }  
  
    public void depositar(int m) {  
        monto=monto+m;  
    }  
  
    public void extraer(int m) {  
        monto=monto-m;  
    }  
  
    public int retornarMonto() {  
        return monto;  
    }  
}
```

```
public void imprimir() {  
    System.out.println(nombre+" tiene depositado la suma de "+monto);  
}  
}
```

```
public class Banco {  
    private Cliente cliente1,cliente2,cliente3;  
  
    public Banco() {  
        cliente1=new Cliente("Juan");  
        cliente2=new Cliente("Ana");  
        cliente3=new Cliente("Pedro");  
    }  
  
    public void operar() {  
        cliente1.depositar (100);  
        cliente2.depositar (150);  
        cliente3.depositar (200);  
        cliente3.extraer (150);  
    }  
}
```

```
public void depositosTotales ()
{
    int t = cliente1.retornarMonto () + cliente2.retornarMonto () +
cliente3.retornarMonto ();
    System.out.println ("El total de dinero en el banco es:" + t);
    cliente1.imprimir();
    cliente2.imprimir();
    cliente3.imprimir();
}

public static void main(String[] ar) {
    Banco banco1=new Banco();
    banco1.operar();
    banco1.depositosTotales();
}
}
```

```
1 public class Cliente {
2     private String nombre;
3     private int monto;
4
5     public Cliente(String nom) {
6         nombre=nom;
7         monto=0;
8     }
9
10    public void depositar(int m) {
11        monto=monto+m;
12    }
13
14    public void extraer(int m) {
15        monto=monto-m;
16    }
17
18    public int retornarMonto() {
19        return monto;
20    }
21
22    public void imprimir() {
23        System.out.println(nombre+" tiene depositado la suma de "+monto);
24    }
25 }
```

```
1 public class Banco {
2     private Cliente cliente1,cliente2,cliente3;
3
4     public Banco() {
5         cliente1=new Cliente("Juan");
6         cliente2=new Cliente("Ana");
7         cliente3=new Cliente("Pedro");
8     }
9
10    public void operar() {
11        cliente1.depositar (100);
12        cliente2.depositar (150);
13        cliente3.depositar (200);
14        cliente3.extraer (150);
15    }
16
17    public void depositosTotales ()
18    {
19        int t = cliente1.retornarMonto () + cliente2.retornarMonto () + cliente3.retornarMonto ();
20        System.out.println ("El total de dinero en el banco es:" + t);
21        cliente1.imprimir();
22        cliente2.imprimir();
23        cliente3.imprimir();
24    }
25
26    public static void main(String[] ar) {
27        Banco banco1=new Banco();
28        banco1.operar();
29        banco1.depositosTotales();
30    }
31 }
```



Problems



Javadoc



Declaration



Console

<terminated> Banco [Java Application] C:\Program Files\Java\jdk-9.0.4\bin\java.exe

El total de dinero en el banco es:300

Juan tiene depositado la suma de 100

Ana tiene depositado la suma de 150

Pedro tiene depositado la suma de 50

Analicemos la implementación del problema.

Los atributos de una clase normalmente son privados para que no se tenga acceso directamente desde otra clase, los atributos son modificados por los métodos de la misma clase:

```
private String nombre;  
private int monto;
```

El constructor recibe como parámetro el nombre del cliente y lo almacena en el atributo respectivo e inicializa el atributo monto en cero:

```
public Cliente(String nom) {  
    nombre=nom;  
    monto=0;  
}
```

Los métodos depositar y extraer actualizan el atributo monto con el dinero que llega como parámetro (para simplificar el problema no hemos validado que cuando se extrae dinero el atributo monto quede con un valor negativo):

```
public void depositar(int m) {  
    monto=monto+m;  
}
```

```
public void extraer(int m) {  
    monto=monto-m;  
}
```

El método retornarMonto tiene por objetivo comunicar al Banco la cantidad de dinero que tiene el cliente (recordemos que como el atributo monto es privado de la clase, debemos tener un método que lo retorne):

```
public int retornarMonto() {  
    return monto;  
}
```


Por último el método imprimir muestra nombre y el monto de dinero del cliente:

```
public void imprimir() {  
    System.out.println(nombre+" tiene depositado la suma de "+monto);  
}
```

Como podemos observar la clase Cliente no tiene función main. Entonces ¿donde definimos objetos de la clase Cliente?

La respuesta a esta pregunta es que en la clase Banco definimos tres objetos de la clase Cliente.

Veamos ahora la clase Banco que requiere la colaboración de la clase Cliente. Primero definimos tres atributos de tipo Cliente:

```
public class Banco {  
    private Cliente cliente1,cliente2,cliente3;
```

En el constructor creamos los tres objetos (cada vez que creamos un objeto de la clase Cliente debemos pasar a su constructor el nombre del cliente, recordemos que su monto de depósito se inicializa con cero):

```
public Banco() {  
    cliente1=new Cliente("Juan");  
    cliente2=new Cliente("Ana");  
    cliente3=new Cliente("Pedro");  
}
```

El método operar del banco (llamamos a los métodos depositar y extraer de los clientes):

```
public void operar() {  
    cliente1.depositar (100);  
    cliente2.depositar (150);  
    cliente3.depositar (200);  
    cliente3.extraer (150);  
}
```

El método `depositosTotales` obtiene el monto depositado de cada uno de los tres clientes, procede a mostrarlos y llama al método `imprimir` de cada cliente para poder mostrar el nombre y depósito:

```
public void depositosTotales ()
{
    int t = cliente1.retornarMonto () + cliente2.retornarMonto () + cliente3.retornarMonto ();
    System.out.println ("El total de dinero en el banco es:" + t);
    cliente1.imprimir();
    cliente2.imprimir();
    cliente3.imprimir();
}
```

Por último en la main definimos un objeto de la clase `Banco` (la clase `Banco` es la clase principal en nuestro problema):

```
public static void main(String[] ar) {
    Banco banco1=new Banco();
    banco1.operar();
    banco1.depositosTotales();
}
```

Problema 2:

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió".

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Dado y la clase JuegoDeDados.

Luego los atributos y los métodos de cada clase:

Dado

- atributos

 - valor

- métodos

 - tirar

 - imprimir

 - retornarValor

JuegoDeDados

atributos

3 Dado (3 objetos de la clase Dado)

métodos

constructor

jugar

Creamos un proyecto en Eclipse llamado: Proyecto2 y dentro del proyecto creamos dos clases llamadas: Dado y JuegoDeDados.

Programa:

```
public class Dado {  
    private int valor;  
  
    public void tirar() {  
        valor=1+(int)(Math.random()*6);  
    }  
  
    public void imprimir() {  
        System.out.println("El valor del dado es:"+valor);  
    }  
  
    public int retornarValor() {  
        return valor;  
    }  
}
```

```
public class JuegoDeDados {  
    private Dado dado1,dado2,dado3;  
  
    public JuegoDeDados() {  
        dado1=new Dado();  
        dado2=new Dado();  
        dado3=new Dado();  
    }  
  
    public void jugar() {  
        dado1.tirar();  
        dado1.imprimir();  
        dado2.tirar();  
        dado2.imprimir();  
        dado3.tirar();  
        dado3.imprimir();  
        if (dado1.retornarValor()==dado2.retornarValor() &&  
            dado1.retornarValor()==dado3.retornarValor()) {  
            System.out.println("Ganó");  
        } else {  
            System.out.println("Perdió");  
        }  
    }  
}
```

```
public static void main(String[] ar){  
    JuegoDeDados j=new JuegoDeDados();  
    j.jugar();  
}  
}
```


*Dado.java × JuegoDeDados.java

```
1 public class Dado {
2     private int valor;
3
4     public void tirar() {
5         valor=1+(int)(Math.random()*6);
6     }
7
8     public void imprimir() {
9         System.out.println("El valor del dado es:"+valor);
10    }
11
12    public int retornarValor() {
13        return valor;
14    }
15 }
```

```
1 public class JuegoDeDados {
2     private Dado dado1,dado2,dado3;
3
4     public JuegoDeDados() {
5         dado1=new Dado();
6         dado2=new Dado();
7         dado3=new Dado();
8     }
9
10    public void jugar() {
11        dado1.tirar();
12        dado1.imprimir();
13        dado2.tirar();
14        dado2.imprimir();
15        dado3.tirar();
16        dado3.imprimir();
17        if (dado1.retornarValor()==dado2.retornarValor() &&
18            dado1.retornarValor()==dado3.retornarValor()) {
19            System.out.println("Ganó");
20        } else {
21            System.out.println("Perdió");
22        }
23    }
24
25    public static void main(String[] ar){
26        JuegoDeDados j=new JuegoDeDados();
27        j.jugar();
28    }
29 }
```



Problems @ Javadoc



<terminated> JuegoDeDados [Ja

El valor del dado es:5

El valor del dado es:5

El valor del dado es:2

Perdió

La clase dado define el atributo "valor" donde almacenamos un valor aleatorio que representa el número que sale al tirarlo.

```
public class Dado {  
    private int valor;
```

El método tirar almacena el valor aleatorio (para generar un valor aleatorio utilizamos el método random de la clase Math, el mismo genera un valor real comprendido entre 0 y 1, pero nunca 0 o 1. Puede ser un valor tan pequeño como 0.0001 o tan grande como 0.9999. Luego este valor generado multiplicado por 6 y antecediendo (int) obtenemos la parte entera de dicho producto):

```
    public void tirar() {  
        valor=1+(int)(Math.random()*6);  
    }
```

Como vemos le sumamos uno ya que el producto del valor aleatorio con seis puede generar números enteros entre 0 y 5.

El método imprimir de la clase Dado muestra por pantalla el valor del dado:

```
public void imprimir() {  
    System.out.println("El valor del dado es:"+valor);  
}
```

Por último el método que retorna el valor del dado (se utiliza en la otra clase para ver si los tres dados generaron el mismo valor):

```
public int retornarValor() {  
    return valor;  
}
```

La clase JuegoDeDatos define tres atributos de la clase Dado (con esto decimos que la clase Dado colabora con la clase JuegoDeDatos):

```
public class JuegoDeDatos {  
    private Dado dado1,dado2,dado3;
```

En el constructor procedemos a crear los tres objetos de la clase Dado:

```
public JuegoDeDados() {  
    dado1=new Dado();  
    dado2=new Dado();  
    dado3=new Dado();  
}
```

En el método jugar llamamos al método tirar de cada dado, pedimos que se imprima el valor generado y finalmente procedemos a verificar si se ganó o no:

```
public void jugar() {  
    dado1.tirar();  
    dado1.imprimir();  
    dado2.tirar();  
    dado2.imprimir();  
    dado3.tirar();  
    dado3.imprimir();  
    if (dado1.retornarValor()==dado2.retornarValor() &&  
        dado1.retornarValor()==dado3.retornarValor()) {  
        System.out.println("Ganó");  
    } else {  
        System.out.println("Perdió");  
    }  
}
```

En la main creamos solo un objeto de la clase principal (en este caso la clase principal es el JuegoDeDados):

```
public static void main(String[] ar){  
    JuegoDeDados j=new JuegoDeDados();  
    j.jugar();  
}
```


Problemas propuestos

1. Plantear una clase Club y otra clase Socio.

La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). En el constructor pedir la carga del nombre y su antigüedad. La clase Club debe tener como atributos 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

29 - Herencia

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la Herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todos los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

clase padre

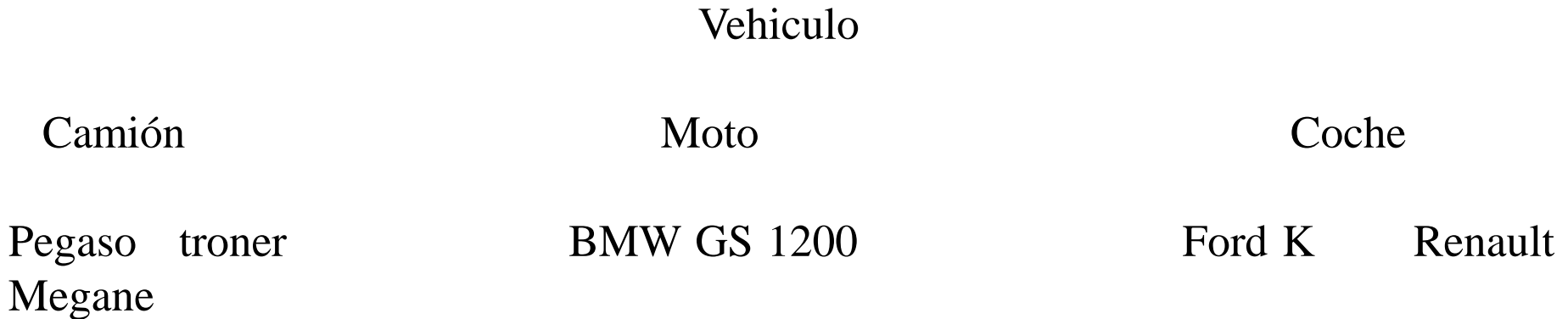
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la la clase padre.

Subclase

Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase Vehículo. ¿Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos).

2) Imaginemos la clase Software. ¿Qué clases podrían derivar de ella?

Software

DeAplicacion

DeBase

ProcesadorTexto PlanillaDeCalculo

SistemaOperativo

Word WordPerfect Excel Lotus123

Linux Windows

El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo.

Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relación de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clase responden a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

Auto "es un" Vehiculo

Circulo "es una" Figura

Mouse "es un" DispositivoEntrada

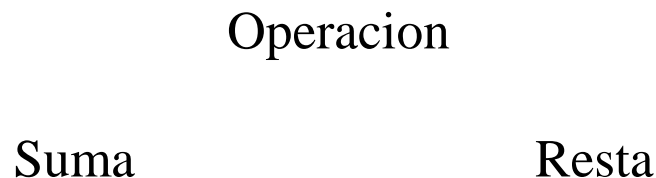
Suma "es una" Operacion

Problema 1:

Ahora plantearemos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado. Los métodos a definir son cargar1 (que inicializa el atributo valor1), carga2 (que inicializa el atributo valor2), operar (que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2, y otro método mostrarResultado.

Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agrupe dichos atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:



Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion), luego los métodos cargar1, cargar2 y mostrarResultado son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operación.

Crear un proyecto y luego crear cuatro clases llamadas: Operación, Suma, Resta y Prueba

Programa:

```
import java.util.Scanner;

public class Operacion {
    protected Scanner teclado;
    protected int valor1;
    protected int valor2;
    protected int resultado;
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
        System.out.print("Introduce el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Introduce el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
}
```

```
public class Suma extends Operacion{  
    void operar() {  
        resultado=valor1+valor2;  
    }  
}
```

```
public class Resta extends Operacion {  
    public void operar(){  
        resultado=valor1-valor2;  
    }  
}
```

```
public class Prueba {  
    public static void main(String[] ar) {  
        Suma suma1=new Suma();  
        suma1.cargar1();  
        suma1.cargar2();  
        suma1.operar();  
        System.out.print("El resultado de la suma es:");  
        suma1.mostrarResultado();  
        Resta resta1=new Resta();  
        resta1.cargar1();  
        resta1.cargar2();  
        resta1.operar();  
        System.out.print("El resultado de la resta es:");  
        resta1.mostrarResultado();  
    }  
}
```

*Operacion.java ✖

*Suma.java

*Resta.java

*Prueba.java

```
1 import java.util.Scanner;
2 public class Operacion {
3     protected Scanner teclado;
4     protected int valor1;
5     protected int valor2;
6     protected int resultado;
7     public Operacion() {
8         teclado=new Scanner(System.in);
9     }
10
11     public void cargar1() {
12         System.out.print("Introduce el primer valor:");
13         valor1=teclado.nextInt();
14     }
15
16     public void cargar2() {
17         System.out.print("Introduce el segundo valor:");
18         valor2=teclado.nextInt();
19     }
20
21     public void mostrarResultado() {
22         System.out.println(resultado);
23     }
24 }
```

 *Operacion.java

 *Suma.java ×

 *Resta.java





```
1 public class Suma extends Operacion{
2     void operar() {
3         resultado=valor1+valor2;
4     }
5 }
```

 *Operacion.java

 *Suma.java

 *Resta.java 

```
1 public class Resta extends Operacion {  
2     public void operar(){  
3         resultado=valor1-valor2;  
4     }  
5 }  
6
```

 *Operacion.java  *Suma.java  *Resta.java  *Prueba.java ✕

```
1  public class Prueba {  
2      public static void main(String[] ar) {  
3          Suma suma1=new Suma();  
4          suma1.cargar1();  
5          suma1.cargar2();  
6          suma1.operar();  
7          System.out.print("El resultado de la suma es:");  
8          suma1.mostrarResultado();  
9          Resta resta1=new Resta();  
10         resta1.cargar1();  
11         resta1.cargar2();  
12         resta1.operar();  
13         System.out.print("El resultado de la resta es:");  
14         resta1.mostrarResultado();  
15     }  
16 }
```



Problems



Javadoc



Declaration



Console



<terminated> Club [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java

Introduce el nombre del socio:luis

Introduce la antigüedad:3

Introduce el nombre del socio:juan

Introduce la antigüedad:6

Introduce el nombre del socio:laura

Introduce la antigüedad:3

Socio con mayor antigüedad:juan tiene una antigüedad de 6

La clase Operación define los cuatro atributos:

```
import java.util.Scanner;  
public class Operacion {  
    protected Scanner teclado;  
    protected int valor1;  
    protected int valor2;  
    protected int resultado;
```

Ya veremos que definimos los atributos con este nuevo modificador de acceso (protected) para que la subclase tenga acceso a dichos atributos. Si los definimos private las subclases no pueden acceder a dichos atributos.

Los métodos de la clase Operacion son:

```
public Operacion() {  
    teclado=new Scanner(System.in);  
}
```

```
public void cargar1() {  
    System.out.print("Ingrese el primer valor:");  
    valor1=teclado.nextInt();  
}
```

```
public void cargar2() {  
    System.out.print("Ingrese el segundo valor:");  
    valor2=teclado.nextInt();  
}
```

```
public void mostrarResultado() {  
    System.out.println(resultado);  
}
```

Ahora veamos como es la sintaxis para indicar que una clase hereda de otra:

```
public class Suma extends Operacion{
```

Utilizamos **la palabra clave extends** y seguidamente el **nombre de la clase padre** (con esto estamos indicando que todos los métodos y atributos de la **clase Operación** son también métodos de la **clase Suma**).

Luego, la característica que añade la clase Suma es el siguiente método:

```
void operar() {  
    resultado=valor1+valor2;  
}
```

El método operar puede acceder a los atributos heredados (siempre y cuando los mismos se declaren protected, en caso que sean private si bien lo hereda de la clase padre solo los pueden modificar métodos de dicha clase padre.

Ahora podemos decir que la clase Suma tiene cinco métodos (cuatro heredados y uno propio) y 3 atributos (todos heredados)

Luego en otra clase creamos un objeto de la clase Suma:

```
public class Prueba {  
    public static void main(String[] ar) {  
        Suma suma1=new Suma();  
        suma1.cargar1();  
        suma1.cargar2();  
        suma1.operar();  
        System.out.print("El resultado de la suma es:");  
        suma1.mostrarResultado();  
        Resta resta1=new Resta();  
        resta1.cargar1();  
        resta1.cargar2();  
        resta1.operar();  
        System.out.print("El resultado de la resta es:");  
        resta1.mostrarResultado();  
    }  
}
```

Podemos llamar tanto al método propio de la clase Suma "operar()" como a los métodos heredados. Quien utilice la clase Suma solo debe conocer que métodos públicos tiene (independientemente que pertenezcan a la clase Suma o a una clase superior)

La lógica es similar para declarar la clase Resta.

La clase Operación agrupa en este caso un conjunto de atributos y métodos comunes a un conjunto de subclases (Suma, Resta). No tiene sentido definir objetos de la clase Operación.

El planteo de jerarquías de clases es una tarea compleja que requiere un perfecto entendimiento de todas las clases que intervienen en un problema, cuales son sus atributos y responsabilidades.

Problema propuesto

1. Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir como responsabilidades un método que cargue los datos personales y otro que los imprima.

Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo.

Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.