



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

<https://www.dropbox.com/s/d3e57mbxq243m0e/Cursojava2015.pdf?dl=0>

# Curso de Java



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## INDICE

- Java es un lenguaje de programación orientado a objetos desarrollado por SUN cuya sintaxis está basada en C++
- Java no es sólo un lenguaje de programación, Java es además un sistema de tiempo de ejecución, un juego de herramientas de desarrollo y una interfaz de programación de aplicaciones (API). Las características son:
  - **Orientado a objetos:** es un paradigma de la programación que nos permite descomponer el código en pequeñas unidades lógicas llamadas objetos.
  - **Multiplataforma:** se pueden ejecutar en cualquier sistema operativo.
  - **Arquitectura neutral.** El compilador genera un fichero objeto con un formato independiente de la arquitectura. Esto permite que el código compilado sea ejecutable en todos los procesadores para los que exista un run time system Java (llamado Maquina Virtual Java).
  - Es muy **robusto**, ya que el desarrollador no tendrá que manejar direcciones de memoria RAM.
  - Es **seguro**, Java detecta los códigos maliciosos.
  - **Orientado a la red:** incorpora diferentes protocolos de red



E.T.S.I.S.I.

UNIVERSIDAD

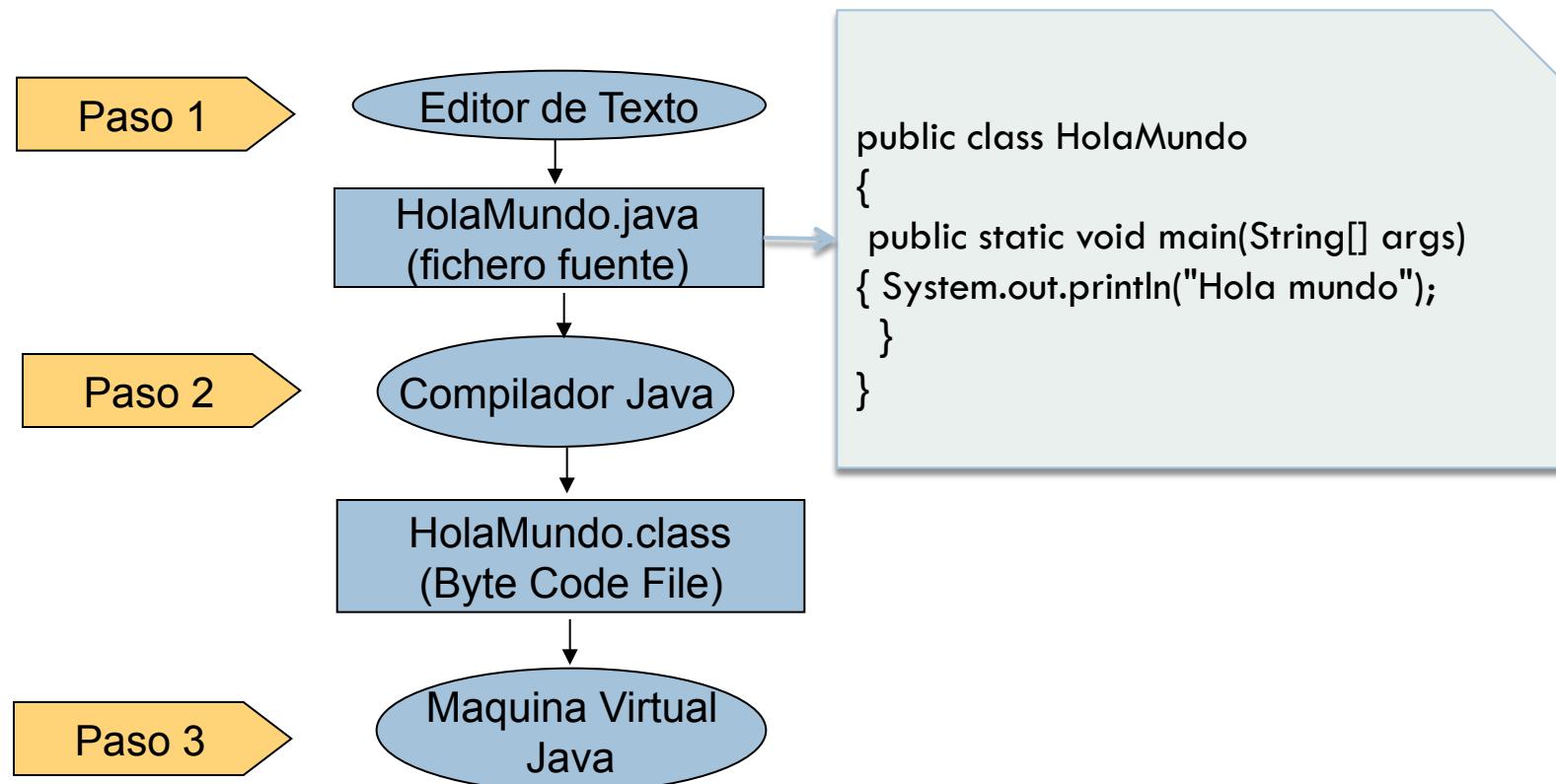
POLITÉCNICA DE MADRID

Java

INDICE

Introducción

## Compilación y ejecución del programa:



# JAVA

Java

Lenguaje de programación Java

Programas fuente  
de Java

Compilado y otras  
herramientas de Java

Bytecode compilado

Sistema de tiempo de  
ejecución Java

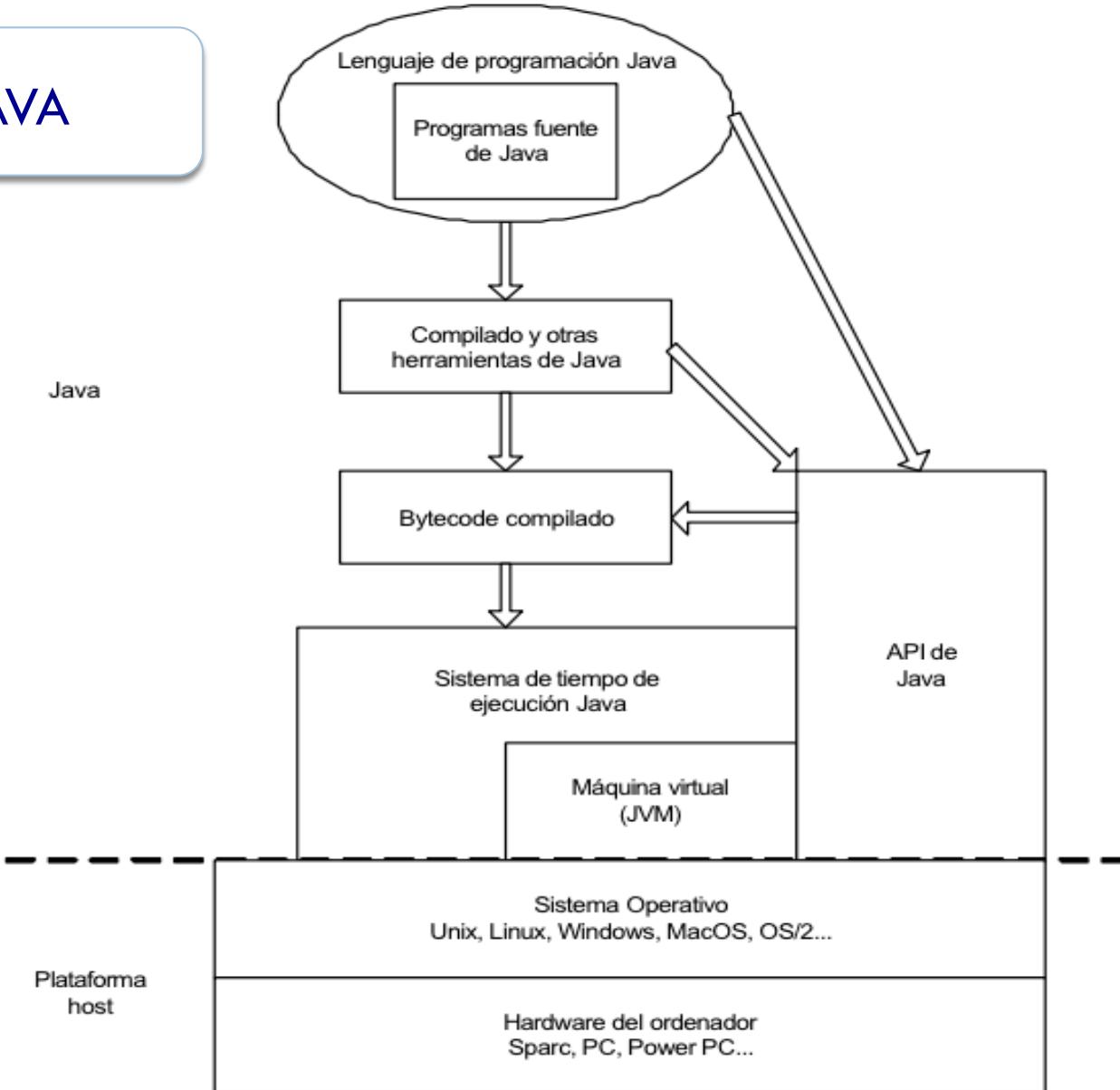
Máquina virtual  
(JVM)

API de  
Java

Sistema Operativo  
Unix, Linux, Windows, MacOS, OS/2...

Plataforma  
host

Hardware del ordenador  
Sparc, PC, Power PC...





E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

### Introducción

- JRE, *Java Runtime Environment*
- JDK, *Java Development Kit*
  - Java SE, Java EE y Java ME.
- El compilador de Java traduce las instrucciones a un código intermedio que se conoce como **byte code**.
- La extensión **.class** es usada para identificar el archivo que contiene la versión en *byte code* del archivo fuente
- *Java Virtual Machine (JVM)*.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

### Introducción

- Existen varias alternativas para compilar y ejecutar un programa en Java:
  - Puede hacerse mediante **comandos** desde el *command prompt* del sistema operativo.
  - Puede hacerse desde un **entorno integrado de desarrollo** (*IDE, Integrated Development Environment*) *Eclipse, IntelliJ...*
  - En un IDE se integran un editor de texto, compilador, depurador y otras herramientas a las cuales se accede por un conjunto de menús

## ÍNDICE

- La orientación a objetos es un método de construcción de sistemas software que incluye todas las fases de desarrollo, desde el análisis hasta la implementación pasando por el diseño.
- Las entidades del mundo real tienen dos características que las definen: un estado y un comportamiento.
- En orientación a objetos cada entidad del mundo real relevante para la aplicación tiene su objeto software equivalente. En los objetos software el estado del mismo se representa con uno o más atributos (piezas de información identificadas con un nombre). El comportamiento de los objetos software se implementa con métodos (funciones) que se aplican al objeto y que pueden manipular el estado de estas variables. Ambos, atributos y métodos, reciben también el nombre de miembros.

## INDICE

- En orientación a objetos un concepto del mundo real se traduce a nivel de código en una clase. Cada objeto software es una instancia de una clase. La diferencia es importante, la clase podría definirse como una plantilla de lo que será el objeto, es decir la clase define que una persona tiene nombre, apellido, fecha de nacimiento etc., y un objeto es una persona concreta, es decir un objeto podría ser Juan García que nació el 15 de agosto de 1985.
- Una clase es un modelo de lo que representa en el mundo real y, como todo modelo , no es una copia exacta del mismo, es una abstracción, es decir recoge solo aquellos detalles, aquella información, que es relevante para la aplicación que se está desarrollando.

# PROGRAMACIÓN ORIENTADA A OBJETOS

Java

## INDICE



Avión del mundo real



### Clase Avión

```
string matrícula  
string marca  
string modelo  
integer numAsientos  
integer litrosCombustible
```

```
llenarDeposito()  
vaciarDeposito()  
gastarCombustible(int numLitros)
```

“Plantilla” que representa a nuestro concepto de mundo real.



Objeto o instancia concreta.

```
matrícula: 343LKD3L  
marca: Boeing  
modelo: 747  
numAsientos:392  
litrosCombustible: 0
```

# Conceptos de la POO

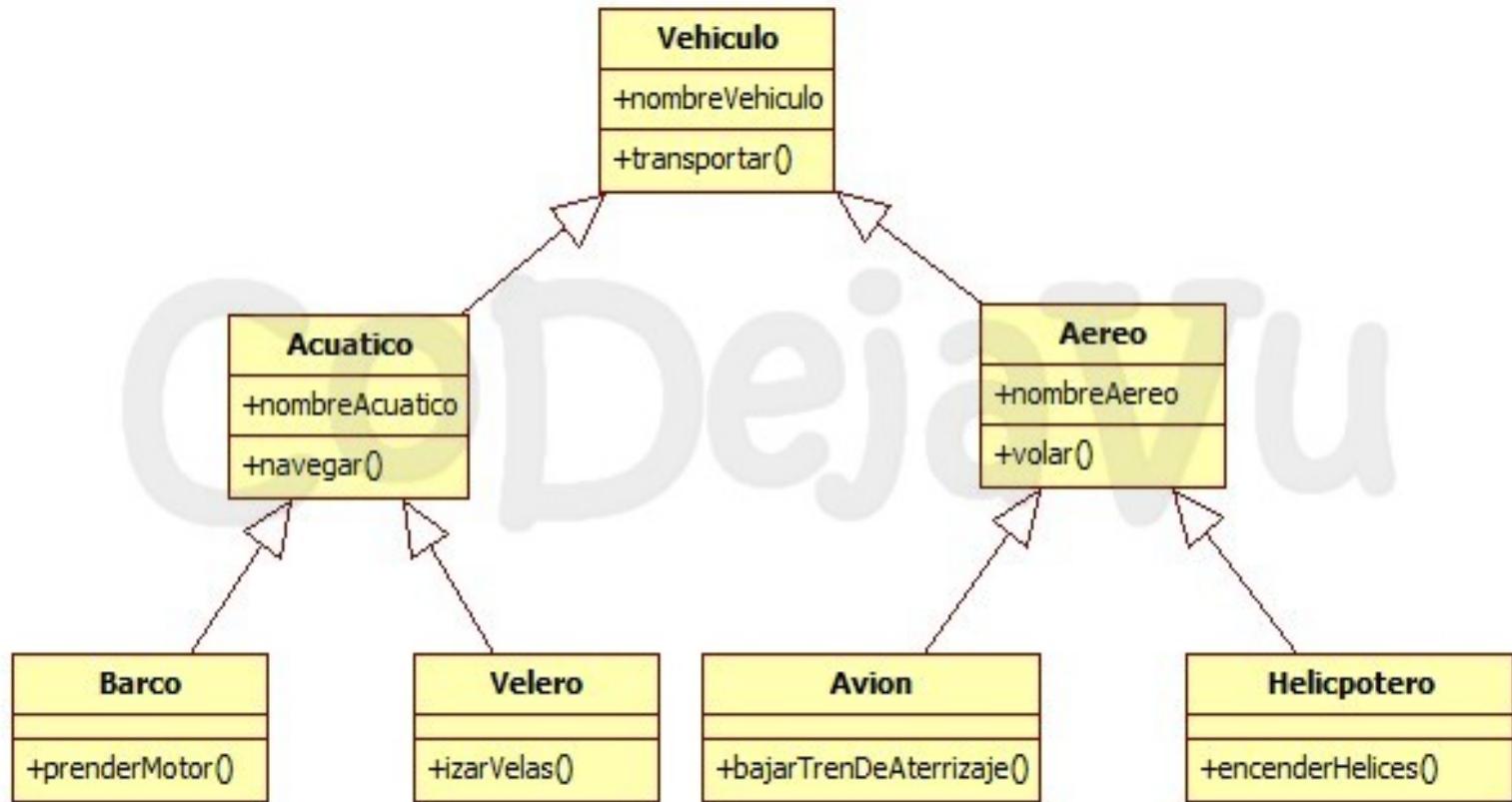
## ÍNDICE

- **Encapsulación.** Las clases pueden ser declaradas como públicas (public) y como package (accesibles sólo para otras clases del package). Las variables miembro y los métodos pueden ser public, private, protected y package. De esta forma se puede controlar el acceso y evitar un uso inadecuado.
- **Herencia.** Una clase puede derivar de otra (extends), y en ese caso hereda todas sus variables y métodos. Una clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados.
- **Polimorfismo.** Los objetos de distintas clases pertenecientes a una misma jerarquía o que implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo.

# Conceptos de la POO

## INDICE

- Propiedades de la POO





E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
class FiguraGeometrica{  
}  
class Triangulo extends FiguraGeometrica {  
}  
public class Principal{  
    public void metodo(){  
        /**Puedo crear objetos polimorficos*/  
        /**Objeto Triangulo de tipo FiguraGeometrica*/  
        FiguraGeometrica triangulo=new Triangulo();  
    }  
}
```

- Vemos que FiguraGeometrica es la superClase y Triangulo es la clase hija o subClase, y por medio del polimorfismo podemos crear una instancia de Triangulo de tipo FiguraGeometrica...



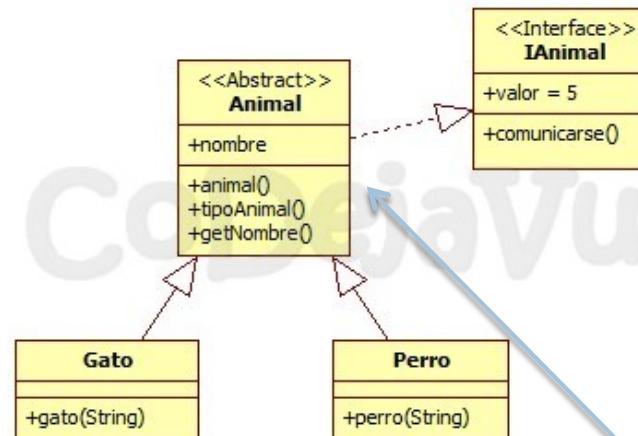
E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE



```
public interface IAnimal {
    int valor=5;
    /*Método Comunicarse, sera
    implementado por las clases concretas
     * que hereden de la clase Animal */
    public void comunicarse();
}
```

```
public abstract class Animal implements IAnimal {
    private String nombre;
    /**Constructor de la clase Animal*/
    public Animal (String nombre){
        this.nombre=nombre;
        System.out.println("Constructor Animal, " +
            "nombre del animal : "+this.nombre);
    }
    public String getNombre(){
        return nombre;
    }
    /*Metodo Abstracto tipoAnimal, la implementación depende
     * de las clases concretas que extiendan la clase Animal */
    public abstract void tipoAnimal();
}
```



E.T.S.I.S.I.

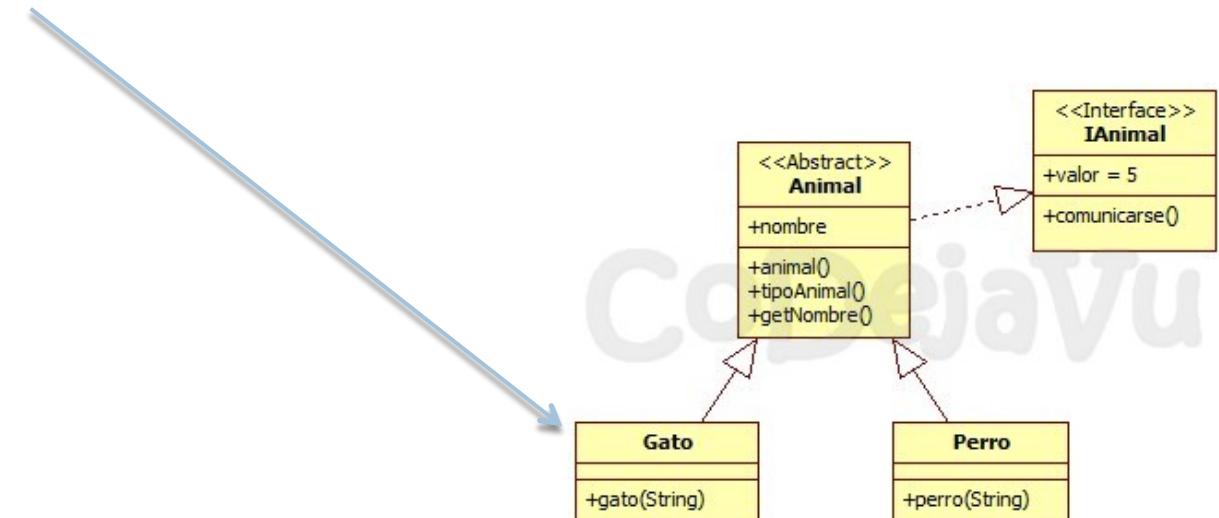
UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
public class Gato extends Animal{
    /**Constructor explicito clase Gato */
    public Gato(String nombre) {
        super(nombre); //envia el parametro a el constructor de la clase padre
        System.out.println("Constructor Gato, nombre : "+nombre);
    }
    public void tipoAnimal() {
        System.out.println("Tipo Animal : Es un Gato");
    }
    public void comunicarse(){
        System.out.println("Metodo comunicarse : El gato maulla... Miau Miau");
    }
}
```





E.T.S.I.S.I.

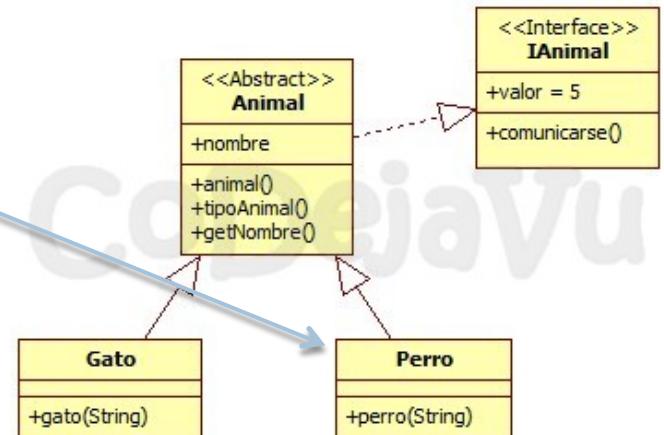
UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
public class Gato extends Animal{  
    public class Perro extends Animal{  
        public Perro(String nombre) {  
            super(nombre);  
            System.out.println("Constructor perro, nombre : "+nombre) }  
  
        public void tipoAnimal() {  
            System.out.println("Tipo Animal : Es un Perro");  
        }  
        public void comunicarse(){  
            System.out.println("Metodo comunicarse : El perro Ladra... Guau Guau");  
        }  
    }  
}
```





E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

```
public class Test {  
    public static void main (String[] arg){  
        /**Creamos anim, un objeto Perro de tipo Animal*/  
        Animal anim= new Perro("goliath") ;  
        anim.tipoAnimal();  
        anim.comunicarse();  
        System.out.println();  
        /**Creamos perro, un objeto Perro de tipo Perro*/  
        Perro perro=new Perro("hercules");  
        perro.tipoAnimal();  
        System.out.println();  
        /**Creamos animalPolimorfico, un objeto perro de tipo Animal  
         * asignamos una referencia ya existente*/  
        Animal animalPolimorfico=perro;  
        animalPolimorfico.tipoAnimal();  
        System.out.println();
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
/**reasignamos la referencia del objeto anim a el objeto perro*/
perro=(Perro) anim;
perro.tipoAnimal();
System.out.println();

/**Creamos gat, un objeto Gato de tipo Animal*/
Animal gat=new Gato("pichi");
gat.tipoAnimal();
gat.comunicarse();
System.out.println();

/**Creamos cat, un objeto Gato de tipo IAnimal
 * Para esto aplicamos polimorfismo usando la Interface*/
IAnimal cat = new Gato("pitufa");
cat.comunicarse();
System.out.println("\nConstante en la interfaz Animal : "+IAnimal.valor);
Animal animales[]={ new Perro("simon"),new Perro("paco"),new Gato("mimi")};
for(Animal a : animales){
    a.tipoAnimal();
}
System.out.println();

}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Constructor Animal, nombre del animal : goliath

Constructor perro, nombre : goliath

Tipo Animal : Es un Perro

Metodo comunicarse : El perro Ladra... Guau Guau

Constructor Animal, nombre del animal : hercules

Constructor perro, nombre : hercules

Tipo Animal : Es un Perro

Tipo Animal : Es un Perro

Tipo Animal : Es un Perro

Constructor Animal, nombre del animal : pichi

Constructor Gato, nombre : pichi

Tipo Animal : Es un Gato

Metodo comunicarse : El gato maulla... Miau Miau



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Constructor Animal, nombre del animal : pitufa

Constructor Gato, nombre : pitufa

Metodo comunicarse : El gato maulla... Miau Miau

Constante en la interfaz Animal : 5

Constructor Animal, nombre del animal : simon

Constructor perro, nombre : simon

Constructor Animal, nombre del animal : paco

Constructor perro, nombre : paco

Constructor Animal, nombre del animal : mimi

Constructor Gato, nombre : mimi

Tipo Animal : Es un Perro

Tipo Animal : Es un Perro

Tipo Animal : Es un Gato



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## ÍNDICE

### □ ESTRUCTURA GENERAL DE UN PROGRAMA JAVA

- Contiene una clase que contiene el programa principal (aquel con la función main()) y algunas clases de usuario que son utilizadas por el programa principal. Los ficheros fuente tienen la extensión \*.java, mientras que los ficheros compilados tienen la extensión \*.class.
- Un fichero fuente (\*.java) puede contener más de una clase, pero sólo una puede ser public. El nombre del fichero fuente debe coincidir con el de la clase public (con la extensión \*.java). Si la clase no es public, no es necesario que su nombre coincida con el del fichero.
- Una clase puede ser public, friendly, pero no private o protected
- De ordinario una aplicación está constituida por varios ficheros \*.class. La aplicación se ejecuta por medio del nombre de la clase que contiene la función main(). Las clases de Java se agrupan en Paquetes (packages), que son librerías de clases.

## ÍNDICE

- Clases
- En una clase se declaran tanto los datos que van a constituir la representación interna de los objetos de esa clase (atributos) como las operaciones que accederán a esos datos (métodos). Métodos y atributos constituyen los miembros de la clase. Una clase es una agrupación de datos (variables o campos) y de métodos que operan sobre esos datos. La definición de una clase en Java se realiza en la siguiente forma:
  - [public] class Classname {  
    // definición de variables y métodos ...
  - }
- donde la palabra public es opcional: si no se pone, la clase tiene la visibilidad por defecto, Todos los métodos y variables deben ser definidos dentro del bloque {...} de la clase.
- Un objeto es un ejemplar concreto de una clase.
  - Classname unObjeto; Classname otroObjeto;
- Todas las variables y funciones de Java deben pertenecer a una clase. No hay variables y funciones globales.
- Si una clase deriva de otra (extends), hereda todas sus variables y métodos.
- Una clase sólo puede heredar de una única clase (en Java no hay herencia múltiple). Si al definir una clase no se especifica de qué clase deriva, por defecto la clase deriva de Object.

## ÍNDICE

- En un fichero se pueden definir varias clases, pero en un fichero no puede haber más que una clase public. Este fichero se debe llamar como la clase public que contiene con extensión \*.java. Con algunas excepciones, lo habitual es escribir una sola clase por fichero.
- Los métodos de una clase pueden referirse de modo global al objeto de esa clase al que se aplican por medio de la referencia this.
- Las clases se pueden agrupar en packages, introduciendo una línea al comienzo del fichero (package packageName;). Esta agrupación en packages está relacionada con la jerarquía de directorios y ficheros en la que se guardan las clases.
- Cuerpo de la clase : Modelaremos como ejemplo el concepto de empleado. Para cada empleado se guarda un nombre, un sueldo y una fecha de contratación. Sobre un objeto de tipo empleado podemos invocar métodos para conocer su nombre, incrementar su sueldo en un porcentaje y obtener el año en que fue contratado:

## ÍNDICE

```
import java.util.Calendar;
import java.util.GregorianCalendar;
class Empleado {
    private String nombre; private double salario; private GregorianCalendar fechaContratacion;

    public Empleado(String n, double s, GregorianCalendar d)
    { nombre = n;
        salario = s;
        fechaContratacion = d; }

    public String getNombre()
    {return nombre; }

    public void setNombre(String nuevoNombre)
    {nombre = nuevoNombre; }

    public double sueldo()
    {return salario; }

    public void aumentarSalario(double)
    {salario *= 1 + porPorcentaje / 100; }

    public GregorianCalendar getFechaContratacion()
    {return fechaContratacion; }

    public int añoContratacion()
    {return fechaContratacion.get(Calendar.YEAR); }}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

# PROGRAMACIÓN ORIENTADA A OBJETOS

Java

## ÍNDICE

- Constructores : Toda clase tiene que tener al menos un constructor, así que si no declaramos ninguno dispondremos por lo menos del llamado constructor por defecto o de oficio. Dicho constructor no tiene parámetros y da valor 0 (numéricos), false (booleano), null a todos los atributos no inicializados explícitamente en su declaración.
- Acceso a los miembros del propio objeto
  - podemos acceder a los atributos del objeto con solo escribir el nombre del atributo. En el ejemplo anterior
    - `return nombre;`
  - No obstante hay una forma de explicitar que se está referenciando el objeto que recibe el mensaje con `this`. Se podría haber escrito
    - `return this.nombre;`
- Visibilidad public y private
- Paso de parámetros: Todos los parámetros a métodos se pasan por valor.
- Miembros de clase :A veces conviene declarar atributos que no pertenecen a ninguna instancia en particular de una clase. Son los miembros de clase
  - `public static int NombreDelMiembro;`

## ÍNDICE

- Herencia : la posibilidad de definir nuevos tipos de datos a partir de los ya existentes mediante el mecanismo de herencia. La herencia consiste en definir una nueva clase describiendo únicamente las características que la diferencian de alguna clase previamente definida.
- Supongamos por ejemplo que nuestra aplicación necesita manejar un nuevo concepto, el de directivo. Supongamos que en nuestra empresa los directivos tienen varias características que los diferencian de los empleados normales: tienen una secretaria y sus aumentos de sueldo se calculan de forma diferente al resto de empleados.
  - habrá que definir una nueva clase Directivo En este caso decimos que Empleado es la superclase de Directivo, y Directivo es una subclase de Empleado. La sintaxis general para indicar relaciones de herencia es:
    - class NombreSubclase extends NombreSuperclase...

## ÍNDICE

```
import java.util.Calendar;
import java.util.GregorianCalendar;

class Directivo extends Empleado
{private String nombreSecretaria;
public Directivo(String n, double s, GregorianCalendar d)
{ super(n, s, d);
nombreSecretaria = ""; }

public void aumentarSalario(double porPorcentaje)
{ // añadir 1/2% bonus de por cada año de servicio
GregorianCalendar today = new GregorianCalendar();
double bonus = 0.5 * (today.get(Calendar.YEAR) -
añoContratacion());
super.aumentarSalario(porPorcentaje + bonus); }

public void setNombreSecretaria(String n)
{nombreSecretaria = n;}
public String getNombreSecretaria()
{return nombreSecretaria; }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- Sobrescribir métodos En el ejemplo de Directivo el método aumentarSalario(...), que ya se heredaba de Empleado, se ha vuelto a definir con un comportamiento distinto. A esto se le llama sobrescribir un método (overriding) no confundir con sobrecarga de métodos
- Los constructores no se heredan.
- Polimorfismo Consiste en que una misma referencia a objeto puede referenciar en distintos momentos de una misma ejecución, a objetos con diferente tipo (diferente forma). Esto es debido a que a una variable de un tipo A podemos no sólo asignarle objetos del tipo A, sino también cualquier objeto de un tipo que descienda de A.

## ÍNDICE

- Empleado jefe = new Directivo(...); // es correcto
  - jefe.getNombreSecretaria();  
// Error de compilación, la clase Empleado no dispone de este método
  - Por otro lado podemos llamar a un método para el cual existan varias implementaciones. En la llamada
    - jefe.aumentarSalario(30);
      - La decisión de qué método utilizar se toma en tiempo de ejecución, y se ejecuta el de la clase a la cual pertenece el objeto referenciado, en este caso la clase Directivo. Este modo de proceder se conoce con el nombre de enlace dinámico (dynamic binding).
- ```
Empleado[] plantilla = new Empleado[3];
plantilla[0] = new Empleado(...);
plantilla[1] = new Empleado(...);
plantilla[2] = new Directivo(...);
for (int i = 0; i < 3; i++) plantilla[i].aumentarSalario(5);
```
- En este caso al tercero se le aplicara el de directivo al pertenecer a esa clase

## ÍNDICE

- Casting de objetos: Un objeto no puede cambiar de tipo a lo largo de su vida. No obstante, a veces es necesario usar el casting para indicar al compilador que el objeto indicado es en realidad de un tipo más especializado de lo que el puede deducir. Los castings están sólo permitidos dentro de una línea de herencia:  

```
Empleado jefe = new Directivo(...);  
Directivo m = (Directivo) jefe
```

  - Sin el casting la segunda asignación no sería correcta, porque la variable jefe es de tipo Empleado, y no todo Empleado es un Directivo
- El casting sólo debe hacerse si estamos seguros que el objeto convertido es del tipo indicado. El siguiente código producirá un error de ejecución:  

```
Empleado jefe = new Empleado(...);  
Directivo m = (Directivo) jefe;
```

## INDICE

- Clases abstractas :Puede tener sentido modelar conceptos abstractos de los cuales no queremos crear instancias directamente, pero que nos son útiles para definir propiedades compartidas por varias subclases.
- Interface
  - Una interface es un conjunto de declaraciones de funciones. Si una clase implementa (implements) una interface, debe definir todas las funciones especificadas por la interface.
  - Una interface puede derivar de otra o incluso de varias interfaces, en cuyo caso incorpora las declaraciones de todos los métodos de las interfaces de las que deriva (a diferencia de las clases, las interfaces de Java sí tienen herencia múltiple).
  -



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

# PROGRAMACIÓN ORIENTADA A OBJETOS

Java

## ÍNDICE

### □ VARIABLES MIEMBRO

- La programación orientada a objetos está centrada en los datos. Una clase está constituida por unos datos y unos métodos que operan sobre esos datos.
- Variables miembro de objeto
  - Cada objeto, es decir cada ejemplar concreto de la clase, tiene su propia copia de las variables miembro. Las variables miembro de una clase pueden ser de tipos primitivos (boolean, int, long, double, ...) o referencias a objetos de otra clase.
  - Los datos tienen que ser inicializados. Por eso las variables miembro de tipos primitivos se inicializan siempre de modo automático, De todas formas, lo más adecuado es inicializarlas también en el constructor.
  - Las variables miembro pueden también inicializarse explícitamente en la declaración, como las variables locales, por medio de constantes o llamadas a métodos
    - `long NTotal = 100;`
  - Las variables miembro se inicializan en el mismo orden en que aparecen en el código de la clase. Esto es importante porque unas variables pueden apoyarse en otras previamente definidas.

## INDICE

### □ VARIABLES MIEMBRO

- una clase tiene su propia copia de las variables miembro.
- Los métodos de objeto se aplican a un objeto concreto poniendo el nombre del objeto y luego el nombre del método, separados por un punto. A este objeto se le llama argumento implícito. Las variables miembro del argumento implícito se acceden directamente o precedidas por la palabra this y el operador punto.
- Las variables miembro pueden ir precedidas en su declaración por uno de los modificadores de acceso: public, private, protected y Friendly (que es el valor por defecto y se omite).

# Modificadores de Alcance

Java

ÍNDICE

| <b>zona</b>                             | <b>private<br/>(privado)</b> | <b>sin<br/>modificador<br/>(friendly)</b> | <b>protected<br/>(protégido)</b> | <b>public<br/>(público)</b> |
|-----------------------------------------|------------------------------|-------------------------------------------|----------------------------------|-----------------------------|
| Misma clase                             | X                            | X                                         | X                                | X                           |
| Subclase en el mismo paquete            |                              | X                                         | X                                | X                           |
| Clase (no subclase) en el mismo paquete |                              | X                                         |                                  | X                           |
| Subclase en otro paquete                |                              |                                           | X                                | X                           |
| No subclase en otro paquete             |                              |                                           |                                  | X                           |



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- Variables miembro de clase (static)
  - Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se les llama variables de clase o variables static. Las variables static se suelen utilizar para definir constantes comunes para todos los objetos de la clase o variables que sólo tienen sentido para toda la clase (por ejemplo, un contador de objetos creados).
  - Las variables de clase se crean anteponiendo la palabra static a su declaración. Para llamarlas se suele utilizar el nombre de la clase
  - Si no se les da valor en la declaración, las variables miembro static se inicializan con los valores por defecto para los tipos primitivos
  - Lo importante es que las variables miembro static se inicializan siempre antes que cualquier objeto de la clase



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- **Métodos de objeto**
  - Los métodos son funciones definidas dentro de una clase. Salvo los métodos static o de clase, se aplican siempre a un objeto de la clase por medio del operador punto (.). Los métodos pueden además tener otros argumentos explícitos que van entre paréntesis, a continuación del nombre del método y denominados parámetros.
  - El valor de retorno puede ser un valor de un tipo primitivo o una referencia.
  - Los métodos pueden definir variables locales. Su visibilidad llega desde la definición al final del bloque en el que han sido definidas.
  - **Sobrecarga de Métodos:** Java permite métodos sobrecargados (overloaded), es decir, métodos distintos que tienen el mismo nombre, pero que se diferencian por el numero y/o tipo de los argumentos.
  - En Java los argumentos de los tipos primitivos se pasan siempre por valor.
- **Métodos de clase (static)** puede haber métodos que no actúen sobre objetos concretos a través del operador punto. A estos métodos se les llama métodos de clase o static. Los métodos y variables de clase se crean anteponiendo la palabra **static**. Para llamarlos se suele utilizar el nombre de la clase, en vez del nombre de un objeto de la clase. Los métodos y las variables de clase son lo más parecido que **Java** tiene a las funciones y variables globales de C/C++



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- Un package es una agrupación de clases.
- El usuario puede crear sus propios packages. Para que una clase pase a formar parte de un package llamado `nombredelpaquete`, hay que introducir en ella la sentencia:
  - `package nombredelpaquete;`
  - que debe ser la primera sentencia del fichero sin contar comentarios y líneas en blanco.
- Los nombres de los packages se suelen escribir con minúsculas, Todas las clases que forman parte de un package deben estar en el mismo directorio.
- Usos:
  - Se usan para agrupar clases relacionadas.
  - Para ayudar en el control de la accesibilidad de clases y miembros.
  - Para evitar conflictos de nombres



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

### Paquetes

#### Interfaces

Métodos

#### Clases

Variables

Métodos

#### Excepciones



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

### □ CLASES INTERNAS

- Una clase interna es una clase definida dentro de otra clase, llamada clase contenedora, en alguna variante de la siguiente forma general:

```
class ClaseContenedora { ... }
```

```
class ClaseInterna { ... } ... }
```

- Las clases internas pueden también ser private y protected
- Los métodos de las clases internas acceden directamente a todos los miembros, incluso private, de la clase contenedora.

### □ cuatro tipos de clases internas:

- Clases internas static. Se conocen también con el nombre de clases anidadas
- Clases internas miembro. Son clases definidas al máximo nivel de la clase contenedora, sin la palabra static.
- Clases internas locales.
- Clases anónimas.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- Clases internas locales
  - Las clases internas locales o simplemente clases locales no se declaran dentro de otra clase al máximo nivel, sino dentro de un bloque de código, normalmente en un método, aunque también se pueden crear en un inicializador static o de objeto.
- Las principales características de las clases locales son:
  - Como las variables locales, las clases locales sólo son visibles y utilizables en el bloque de código en el que están definidas. Los objetos de la clase local deben ser creados en el mismo bloque en que dicha clase ha sido definida. De esta forma se puede acercar la definición al uso de la clase.
  - Las clases internas locales tienen acceso a todas las variables miembro y métodos de la clase contenedora. Pueden ver también los miembros heredados, tanto por la clase interna local como por la clase contenedora.
  - Un objeto de una clase interna local sólo puede existir en relación con un objeto de la clase contenedora, que debe existir previamente.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- Restricciones en el uso de las clases internas locales:
  - No pueden tener el mismo nombre que ninguna de sus clases contenedoras.
  - No pueden definir variables, métodos y clases static.
  - No pueden ser declaradas public, protected, private o package, pues su visibilidad es siempre la de las variables locales, es decir, la del bloque en que han sido definidas.
  - Las clases internas locales se utilizan para definir clases Adapter en el AWT.
- Clases anónimas
  - Las clases anónimas son muy similares a las clases internas locales, pero sin nombre. En las clases internas locales primero se define la clase y luego se crean uno o más objetos. En las clases anónimas se unen estos dos pasos: Como la clase no tiene nombre sólo se puede crear un único objeto, ya que las clases anónimas no pueden definir constructores. Las clases anónimas se utilizan en el AWT para definir clases y objetos que gestionen los eventos de los distintos componentes de la interface de usuario



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- VARIABLES FINALES
- Una variable de un tipo primitivo declarada como final no puede cambiar su valor a lo largo de la ejecución del programa. Puede ser considerada como una constante, y equivale a la palabra const de C/C++.
- Java permite separar la definición de la inicialización de una variable final. La inicialización puede hacerse más tarde, en tiempo de ejecución, llamando a métodos o en función de otros datos. La variable final así definida es constante (no puede cambiar), pero no tiene por qué tener el mismo valor en todas las ejecuciones del programa, pues depende de cómo haya sido inicializada.
- Además de las variables miembro, también las variables locales y los propios argumentos de un método pueden ser declarados final.

```
CaracterPublico/Privado static final TipoDeLaConstante = valorDeLaConstante;  
private static final float PI = 3.1416
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- Los nombres de Java son sensibles a las letras mayúsculas y minúsculas. Así, las variables Suma, suma y SUMA son consideradas variables completamente diferentes. Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad. Se recomienda seguir las siguientes instrucciones:
- En Java es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes:
  - Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue a otra
- Los nombres de clases e interfaces comienzan siempre por mayúscula (Ejemplos: Empleado, Directivo...)
- Los nombres de objetos, los nombres de métodos y variables miembro, y los nombres de las variables locales de los métodos, comienzan siempre por minúscula (Ejemplos: main(), nomina(), numEmpleados, x, y, z).
- Los nombres de las variables finales, es decir de las constantes, se definen siempre con mayúsculas



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Los lenguajes de programación, incluyendo Java, incluyen una serie de elementos comunes:
  - Palabras reservadas
  - Identificadores
  - Signos de puntuación
  - Operadores
  - Reglas de Sintaxis



E.T.S.I.S.I.

UNIVERSIDAD  
POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

```
public class Nomina {  
  
    public static void main(String[] args) {  
        int horas = 40;  
        double precioHora = 25.0, total;  
  
        total = horas * precioHora;  
        System.out.print("Nomina: € ");  
        System.out.println(total);  
    }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Las **palabras reservadas** (*keywords*) tienen un significado especial en el lenguaje.
- En el programa aparecen: **public**, **class**, **static**, **void**, **int** y **double**.
- No pueden ser utilizadas para ninguna otra cosa que no sea para lo que fueron definidas.
- En Java las palabras reservadas se escriben usando sólo letras minúsculas.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Los **identificadores** son palabras que el programador utiliza para nombrar cosas tales como programas, clases, variables y rutinas (métodos), entre otros.
- En el programa aparecen:
  - **Nomina, String y System** para el programa y otras clases
  - **args, horas, precioHora, total** y **out** para las variables
  - **main** y **println** para los métodos



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- En Java los identificadores que corresponden a las clases comienzan con letra mayúscula.
- Los identificadores que corresponden a las variables y a los métodos se escriben comenzando con letra minúscula y usando una letra mayúscula al comienzo de cada nueva “palabra”
- Java es un lenguaje **sensible a mayúsculas y minúsculas** (*case sensitive*).
- Esto quiere decir que todos los identificadores tienen que escribirse tal como fueron definidos.  
Por ejemplo, `precioHora` y `PrecioHora` serían dos identificadores distintos.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Los **signos de puntuación** sirven para propósitos específicos, tales como marcar el final de una instrucción o indicar el comienzo y el fin de un conjunto de instrucciones.
  - En Java las instrucciones terminan con punto y coma (';').
  - En Java las cadenas de caracteres (*Strings*) se colocan dentro de comillas dobles ('"').
  - En el programa aparecen otros signos de puntuación tales como las llaves ('{' y '}'), los paréntesis ('(' y ')'), los corchetes ('[' y ']') y el punto ('.').



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Los **operadores** son símbolos que llevan a alguna operación con datos, también conocidos como **operandos**.
- Algunos ejemplos de **operadores aritméticos** son:
  - + para sumar dos números
  - para restar dos números
  - \* para multiplicar dos números
  - / para dividir dos números
- Algunos operadores pueden utilizarse para varias cosas.
- Por ejemplo, el operador + puede ser utilizado para:
  - Sumar dos números
  - Indicar que un número es positivo
  - Concatenar (unir) dos cadenas de caracteres



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Un operador importante es el `=`, que requiere una variable a la izquierda y una expresión a la derecha:  
`total = horas * precioHora;`
- Este operador, llamado **operador de asignación**, evalúa la expresión de la derecha y asigna el resultado a la variable de la izquierda.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

```
public class Nomina {  
    public static void main(String[] args) {  
        int horas = 40;  
        double precioHora = 25.0, total;  
  
        total = horas * precioHora;  
        System.out.print("Nomina: € ");  
        System.out.println(total);  
    }  
}
```

- Todo programa en Java debe tener por lo menos una clase.
- Una **clase** (*class*) es un contenedor de subprogramas (llamados **métodos** en Java).
- En un archivo fuente se puede tener más de una clase pero sólo una puede tener el atributo **public**.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Cuando un archivo fuente en Java contiene una clase pública, el nombre de la clase pública tiene que ser el mismo del archivo.
- En el ejemplo, la clase pública que representa el programa se llama **Nomina**.
- Por lo tanto, el archivo fuente se **tiene** que llamar **Nomina.java**
- La definición de una clase comienza con un encabezado que contiene la palabra **class**.
- Todas las definiciones e instrucciones que pertenecen a una clase están encerrados entre llaves: { }
- Estas definiciones e instrucciones son conocidas como el **cuerpo de la clase**.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

### Introducción

### El lenguaje Java

### Sintaxis

- En el programa de ejemplo se definió esta clase:

```
public class Nomina {  
    cuerpo de la clase Nomina  
}
```

- Como se indicó anteriormente, una clase es un contenedor de métodos.
- Un **método** es un contenedor de instrucciones que llevan a cabo una tarea.
- Toda aplicación tiene que tener un método principal de nombre **main** que es el punto de comienzo de la aplicación.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

En el programa de ejemplo se definió el método **main**:

```
public class Nomina{  
    public static void main(String[] args) {
```

*cuerpo del método main*

```
    }  
}
```

- El método main siempre lleva el mismo encabezado.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sintaxis

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Los métodos también llevan un encabezado y el **cuerpo del método** se coloca dentro de llaves.
- Los métodos contienen **sentencias** que son las instrucciones que se le dan al ordenador.
- Existen dos tipos principales de sentencias:
  - Sentencias de declaración permiten indicar el tipo de datos, nombre y, posiblemente, un valor inicial para cada variable.
  - Sentencias ejecutables permiten que la computadora lleva a cabo instrucciones tales como:
    - Obtener un valor del usuario (input)
    - Asignarle a una variable el resultado de una expresión
    - Mostrar un valor en la pantalla (output)



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)

# Nuestro programa contiene las siguientes sentencias ejecutables (**en negrita**):

```
public class Nomina {  
    public static void main(String[] args) {  
        int horas = 40;  
        double precioHora = 25.0, total;  
        total = horas * precioHora;  
        System.out.print("Nomina: €");  
        System.out.println(total);  
    }  
}
```

- Java provee la clase **System** que contiene una referencia a la pantalla llamada **out**.
- La variable **out** reconoce los métodos **print** y **println** que se utilizan para mostrar valores en la pantalla (una operación de *output*).
- La clase **System** es un contenedor de definiciones y métodos pero no es una aplicación ya que no contiene el método **main**.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de  
Datos

- **Tipos de datos.**

El lenguaje Java reconoce de forma predefinida los siguientes tipos de datos: Numéricos:

- Enteros (**int**). En el rango es  $\pm 2.147.483.648$  (4 Byte) Además existen otros tipos de datos enteros:  
*byte*  $\pm 127$  (1 Byte)  
*short*  $\pm 32768$  (2 Byte)  
*long*  $\pm 9.223.372.036.854.775.808$ . (8 Byte)
- Racionales o fraccionarios (**float**). (4 Byte) También se pueden utilizar números reales con mayor precisión por medio del tipo **double** (8 Byte)
- Lógicos (**boolean**). Se refieren a los valores utilizados en el Álgebra de Boole (verdadero: **true** o falso: **false**).
- Carácter (**char**). Se utiliza para representar todos los caracteres del estándar Unicode (que incluye el ASCII).
- Cadena de caracteres (**String**). Este tipo de datos consiste en una secuencia de caracteres (por ejemplo “*El cielo está enladrillado*”).



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

Salida

- Los clase `System` y los métodos `print` y `println` pertenecen a lo que se denomina como la **Interfaz de Programación de Aplicaciones** (Java API, *Application Programming Interface*).
- Un API es una **biblioteca** (*library*) que contiene una serie de clases para llevar a cabo ciertas operaciones.
- Las clases y los métodos en el Java API están disponibles para todos los programas en Java.
- Para mostrar en la pantalla (`out`) el mensaje:  
`Programar es divertido!`
- se puede utilizar el método `println` de la siguiente manera:
  - `System.out.println("Programar es divertido!");`
- Como el mensaje es una cadena de caracteres, éste debe colocarse entre comillas dobles ('"').



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Salida

- El método **println** muestra un valor en la pantalla y mueve el cursor al inicio de la próxima línea.
- El método **print** muestra un valor en la pantalla pero no mueve el cursor.
- Por ejemplo, otra forma de mostrar el mensaje **Programar es Divertido!** es:

```
System.out.print("Programar");  
System.out.println(" es divertido! ");
```

- Otra forma de mostrar varios valores en una sola línea es usar el operador **+**, como muestra este ejemplo:  

```
System.out.println("Nomina:" + total + "€");
```
- Cuando uno de los valores es un carácter o una cadena de caracteres, al operador **+** se le llama **concatenación**.
- Este operador simplemente une los dos valores.
- El resultado saldrá así en la pantalla:  
**Nomina: 1000.0€**



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Entrada

La sentencia **readline**, cuya sintaxis básica es:

```
BufferedReader linea = new BufferedReader (new InputStreamReader (System.in));
String cadena;
cadena = linea.readLine();
```

En las dos primeras líneas, se declaran el objeto linea del tipo BufferedReader y la variable cadena de tipo String.  
El usuario escribe los caracteres correspondientes y finaliza pulsando la tecla “Intro”.

Se utiliza el método readLine sobre el objeto linea, y se almacena en cadena. Se continúa con la ejecución del programa.

Para leer otros datos que no sean de tipo String, habría que utilizar una sintaxis similar a la siguiente:

```
BufferedReader linea = new BufferedReader (new InputStreamReader (System.in));
int n;
n = Integer.parseInt (linea.readLine());
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## INDICE

## □ La API de Java

- Uno de los atractivos y la gran utilidad de Java radica en la interfaz para programación de aplicaciones (API) de Java. La API consiste en un conjunto de paquetes (packages) que se distribuyen con el JSRK como bibliotecas de clase. Estos paquetes proporcionan una interfaz común para desarrollar programas en todas las plataformas Java. La API ofrece todas las posibilidades para desarrollar desde programas de consola hasta aplicaciones con interfaz GUI pasando por aplicaciones cliente/servidor, videojuegos entre otros.
- Los paquetes son conjuntos de clases, interfaces y excepciones relacionados. Por ejemplo, vienen separados según se trate de programas de ventana, applets, software de conexión de red, etcétera.
- <http://docs.oracle.com/javase/8/docs/api/>



E.T.S.I.S.I.

UNIVERSIDAD  
POLITÉCNICA DE MADRID

Java

## INDICE

docs.oracle.com/javase/7/docs/api/

Java™ Platform Standard Ed. 7

All Classes

Packages

- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font

All Classes

- AbstractAction
- AbstractAnnotationValueVisitor6
- AbstractAnnotationValueVisitor7
- AbstractBorder
- AbstractButton
- AbstractCellEditor
- AbstractCollection
- AbstractColorChooserPanel
- AbstractDocument
- AbstractDocument.AttributeContext
- AbstractDocument.Content
- AbstractDocument.ElementEdit
- AbstractElementVisitor6
- AbstractElementVisitor7
- AbstractExecutorService
- AbstractInterruptibleChannel
- AbstractLayoutCache
- AbstractLayoutCache.NodeDimensions
- AbstractList
- AbstractListModel
- AbstractMap
- AbstractMap.SimpleEntry
- AbstractMap.SimpleImmutableEntry
- AbstractMarshallerImpl
- AbstractMethodError
- AbstractOwnableSynchronizer
- AbstractPreferences
- AbstractProcessor
- AbstractQueue
- AbstractQueuedLongSynchronizer
- AbstractQueuedSynchronizer
- AbstractRegionPainter

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

## Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

### Packages

| Package                   | Description                                                                                                                                                                                                                      |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| java.applet               | Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.                                                                                                        |
| java.awt                  | Contains all of the classes for creating user interfaces and for painting graphics and images.                                                                                                                                   |
| java.awt.color            | Provides classes for color spaces.                                                                                                                                                                                               |
| java.awt.datatransfer     | Provides interfaces and classes for transferring data between and within applications.                                                                                                                                           |
| java.awt.dnd              | Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI. |
| java.awt.event            | Provides interfaces and classes for dealing with different types of events fired by AWT components.                                                                                                                              |
| java.awt.font             | Provides classes and interface relating to fonts.                                                                                                                                                                                |
| java.awt.geom             | Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.                                                                                                              |
| java.awt.im               | Provides classes and interfaces for the input method framework.                                                                                                                                                                  |
| java.awt.im.spi           | Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.                                                                                                             |
| java.awt.image            | Provides classes for creating and modifying images.                                                                                                                                                                              |
| java.awt.image.renderable | Provides classes and interfaces for producing rendering-independent images.                                                                                                                                                      |
| java.awt.print            | Provides classes and interfaces for a general printing API.                                                                                                                                                                      |
| java.beans                | Contains classes related to developing beans – components based on the JavaBeans™ architecture.                                                                                                                                  |
| java.beans.beancontext    | Provides classes and interfaces relating to bean context.                                                                                                                                                                        |
| java.io                   | Provides for system input and output through data streams, serialization and the file system.                                                                                                                                    |
| java.lang                 | Provides classes that are fundamental to the design of the Java programming language.                                                                                                                                            |
| java.lang.annotation      | Provides library support for the Java programming language annotation facility.                                                                                                                                                  |
| java.lang.instrument      | Provides services that allow Java programming language agents to instrument programs running on the JVM.                                                                                                                         |



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Entrada

La clase Scanner se encuentra en el paquete java.util. (**import java.util.Scanner;**)  
Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada.  
Si el dispositivo de entrada es el teclado escribiremos:

**Scanner sc = new Scanner(System.in);**

Se ha creado el objeto sc asociado al teclado representado por *System.in*

**Ejemplos de lectura:**

Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, nextLine() para leer un string...

**Ejemplo** de lectura por teclado de un número entero:

```
int n;  
System.out.print("Introduzca un número entero: ");  
n = sc.nextInt();
```

**Ejemplo** de lectura de un número de tipo double:

```
double x;  
System.out.print("Introduzca número de tipo double: ");  
x = sc.nextDouble();
```

**Ejemplo** de lectura de una cadena de caracteres:

```
String s;  
System.out.print("Introduzca texto: ");  
s = sc.nextLine();
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

- Existen tres formas de hacer comentarios en Java:
  - Comentarios de una sola línea
  - Comentarios de más de una línea
  - Comentarios de documentación
- Para escribir un **comentario de una sola línea** se usan los símbolos //:

```
// Esto es un comentario.
```
- Para escribir un **comentario de varias líneas** se comienzan con /\* y se finaliza con \*/:

```
/*
 * Esto es un comentario
 * de dos líneas.
 */
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

+

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

## Operadores Aritméticos: Los habituales

Suma + .

Resta - .

Multiplicación \* .

División / .

Resto de la División % .

## Operadores de Asignación: El principal es '=' pero hay más operadores de asignación con distintas funciones

'+=' : op1 += op2 -> op1 = op1 + op2

'-=' : op1 -= op2 -> op1 = op1 - op2

'\*=' : op1 \*= op2 -> op1 = op1 \* op2

'/=' : op1 /= op2 -> op1 = op1 / op2

'%=' : op1 %= op2 -> op1 = op1 % op2



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

+

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)

**Operadores Incrementales:** Son los operadores que nos permiten incrementar las variables en una unidad.

- '++'
  - $i = i + 1; \Rightarrow i++;$
- '--'
  - $i = i - 1; \Rightarrow i--;$

El operador unario ++ está a la derecha del operando.

La sentencia  $j = i++;$  asigna a  $j$ , el valor que tenía  $i$ .

- $j = i++; \Rightarrow j = i;$   
 $i = i + 1;$

El operador unario ++ está a la izquierda del operando

- $j = ++i; \Rightarrow j = i + 1;$   
 $j = i;$



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

+

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)

**Operadores Relacionales:** Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor.

Devuelven siempre un valor boolean.

'>': Mayor que

'<': Menor que

'==': Igualas

'!=': Distintos

'>=': Mayor o igual que

'<=': Menor o igual que

**Operadores Unarios:** El mas (+) y el menos (-). Para cambiar el signo del operando.

**Operadores Lógicos:** Nos permiten construir expresiones lógicas.

'&&' : devuelve true si ambos operandos son true.

'||' : devuelve true si alguno de los operandos son true.

'!' : Niega el operando que se le pasa.

**Operador de concatenación con cadena de caracteres '+':**

Por Ejemplo:

```
System.out.println("El total es"+ result +"unidades");
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Sentencia if

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

**Bifurcaciones:** Permiten ejecutar código en función de una expresión evaluada **Bifurcaciones if:**

Su sintaxis es:

```
if (ExpresionBooleana) {conjuntoDeSentencias}
else {conjuntoAlternativo}
```

```
if (ExpresionBooleana) {conjuntoDeSentencias}
else if {conjuntoAlternativo}
else if {conjuntoAlternativo2}
```

**Ejemplos:**

```
if (i == 5){ System.out.println(" i vale 5 ");}
else {System.out.println("i no vale 5");}
```

```
if (i == 5){ System.out.println(" i vale 5 ");}
else if (i < 5){System.out.println("i es menor que 5");}
else if (i > 5){System.out.println("i es mayor que 5");}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)

Clase que comprueba si un numero es par o impar utilizando un if:

```
public class Prueba_if
{
    public static void main(String [] args) throws NumberFormatException, IOException
    {
        int dato;
        BufferedReader linea=new BufferedReader (new InputStreamReader (System.in));
        System.out.print("Número: ");
        dato = Integer.parseInt(linea.readLine());
        if ((dato % 2) == 0)
            System.out.println ("es par");
        else System.out.println ("es impar");
    }
}
```

Ejercicio: Hacer una clase que determine cuantas cifras tiene un número introducido por el usuario comprendido entre 0 y 99999.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

```
public class ejer2 {  
    public static void main(String[] args)  
    { int num;  
        BufferedReader linea=new BufferedReader (new InputStreamReader (System.in));  
        System.out.print("Introduzca un número entre 0 y 99.999: ");  
        num = Integer.parseInt (linea.readLine());  
        if(num<10)  
            System.out.println("tiene 1 cifra");  
        else{ if(num<100)  
            System.out.println("tiene 2 cifras");  
        else{ if(num<1000)  
            System.out.println("tiene 3 cifras");  
        else{ if(num<10000)  
            System.out.println("tiene 4 cifras");  
        else{ if(num<100000)  
            System.out.println("tiene 5 cifras");  
        }  
    }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)

**Bifurcaciones switch** Son las que permiten realizar varias acciones distintas dependiendo del estado de una variable.

```
switch (Expresion){  
    case valor1: conjuntoDeSentencias;  
        break;  
    case valor2: SentenciasAlternativas;  
        break;  
    case valor3: SentenciasAlternativas2;  
        break;  
    case valor4: SentenciasAlternativas3;  
        break;  
    default:SentenciasAlternativas4;  
        break;  
}
```

La sentencia 'break' detrás de cada opción de case sirve para que no evalue el resto de opciones del 'switch' , Ejemplo:

```
switch (i) {  
    case 1 : System.out.println( "i contiene un 1") break;  
    case 2: System.out.println( "i contiene un 2") break;  
    case 3 : System.out.println( "i contiene un 3") break;  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)

```
public class Prueba_swicth
{
    public static void main(String[] args) throws NumberFormatException, IOException
    { int opc;
        BufferedReader linea=new BufferedReader (new InputStreamReader (System.in));
        System.out.print("Opcion: ");
        opc = Integer.parseInt (linea.readLine());
        switch (opc)
        { case 1: System.out.println("lunes"); break;
            case 2: System.out.println("martes"); break;
            case 3: System.out.println("miercoles"); break;
            case 4: System.out.println("jueves"); break;
            case 5: System.out.println("viernes"); break;
            case 6: System.out.println("sabado"); break;
            case 7: System.out.println("domingo"); break;
            default: System.out.println("opcion no valida");
                      break;
        } }
    }
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

### **Bucle while**

El bucle *while* es el bucle básico de iteración. Sirve para realizar una acción sucesivamente mientras se cumpla una determinada condición.

La forma general del bucle *while* es la siguiente:

**while ( expresiónBooleana )**

```
{  
    sentencias;  
}
```

Las *sentencias* se ejecutan mientras la *expresiónBooleana* tenga un valor de *verdadero*.

Por ejemplo, multiplicar un número por 2 hasta que sea mayor que 100:

```
int i = 1;  
while ( i <= 100 ) {  
    i = i * 2;  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)[Bucles](#)

```
public class Prueba_while
{
    public static void main (String[] args) throws NumberFormatException, IOException
    {
        int opc;
        BufferedReader linea=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Opción: ");
        opc = Integer.parseInt(linea.readLine());
        while (opc != 0)
        {
            switch (opc)
            {
                case 1: System.out.println("lunes"); break;
                case 2: System.out.println("martes"); break;
                case 3: System.out.println("miercoles"); break;
                case 4: System.out.println("jueves"); break;
                case 5: System.out.println("viernes"); break;
                case 6: System.out.println("sabado"); break;
                case 7: System.out.println("domingo"); break;
                default: System.out.println("opción no valida"); break;
            }
            System.out.print("Opción: ");
            opc = Integer.parseInt(linea.readLine());
        }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

## Bucles

### **Bucle do-while**

El bucle *do-while* es similar al bucle *while*, pero en el bucle *while* la expresión se evalúa al principio del bucle y en el bucle *do-while* la evaluación se realiza al final.

La forma general del bucle *do-while* es la siguiente:

```
do {  
    sentencias;  
} while ( expresiónBooleana );
```

Ejemplo:

```
Int i=1;  
do {  
    System.out.println( "el valor de i es: " + i);  
    i++;  
} while ( i <11 );
```

**Ejercicio:** Codificar una clase que pida una operación de Suma, resta, división o multiplicación y dos números y realice dicha operación.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

```
import java.io.*;
public class calculadora {
    public static void main(String[] args) throws IOException {
        BufferedReader en = new BufferedReader (new InputStreamReader(System.in));
        int opcion, resultado,numero1,numero2;
        opcion = 0;
        do { System.out.print("¿ Que desea hacer?\n1)Suma\n2)Resta\n3)Multiplicación\n4)División\n\n> ");
            opcion = Integer.parseInt(en.readLine());
            System.out.print("Ingrese el primer número: >> ");
            numero1 = Integer.parseInt(en.readLine());
            System.out.print("Ingrese el segundo número: >> ");
            numero2 = Integer.parseInt(en.readLine());
            switch(opcion){
                case 1:
                    resultado = numero1 + numero2;
                    System.out.println("La suma es "+resultado+"\n");
                    break;
                case 2:
                    resultado = numero1 - numero2;
                    System.out.println("La resta es "+resultado+"\n");
                    break;
                case 3:
                    resultado = numero1 * numero2;
                    System.out.println("La multiplicación es "+resultado+"\n");
                    break;
                case 4:
                    resultado = numero1 / numero2;
                    System.out.println("La división es "+resultado+"\n");
                    break;
            }
        }while(opcion != 0);
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

### **Bucle for**

La forma general de la sentencia *for* es la siguiente:

```
for ( iniciación ; terminación ; incremento )
  {sentencias;
  }
```

```
public class Numeros7en7 {
  public static void main(String[] args)
  {
    for (int i=100;i>=0;i-=7)
      System.out.println(i);
  }
}
```

Ejercicio hacer una clase que pida un numero entre 1 y 9 e imprima su tabla de multiplicar



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

```
import java.io.*;
public class tabla {
    public static void main(String[] args) throws IOException {
        BufferedReader en = new BufferedReader (new InputStreamReader(System.in));
        int numero ;
        opcion = 0;
        System.out.print("Indique el numero :");
        numero = Integer.parseInt(en.readLine());
        for ( int factor = 1; factor <= 9; factor ++ )
            { System.out.println( numero + " x " + factor + " = " + numero*factor );
            }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

### Ejercicios:

- 1.- Codificar una clase que pida dos números e indique cual es el mayor y el menor o si son iguales.
- 2.- Pedir una nota de 0 a 10 y mostrarla de la forma: Insuficiente, Suficiente, Bien.
- 3.- Realizar un juego para adivinar un número. Para ello pedir un número N, y luego ir pidiendo números indicando “mayor” o “menor” según sea mayor o menor con respecto a N. El proceso termina cuando el usuario acierta.
- 4.- Pedir 15 números y escribir la suma total.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## Ejercicio 1

### INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

```
public class MayorMenor {  
    public static void main(String[] args) {  
        int num1,num2;  
        BufferedReader linea=new BufferedReader(new InputStreamReader(System.in));  
        System.out.print("Introduce 1er numero: ");  
        num1= Integer.parseInt(linea.readLine());  
        System.out.print("Introduce segundo numero: ");  
        num2= Integer.parseInt(linea.readLine());  
  
        if(num1==num2)  
            System.out.println("son iguales");  
        else if(num1<num2)  
            System.out.println("el numero 1 es mayor que el numero 2");  
        else System.out.println("el numero 2 es mayor que el numero 1");  
    }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

String

Los String en Java son objetos que contienen cadenas de caracteres.

Para crear un string explícitamente escribimos

```
String cadena=new String("El primer string");
```

También se puede escribir, alternativamente

```
String cadena="El primer string";
```

Para crear un string nulo se puede hacer de estas dos formas

```
String cadena="";  
String cadena=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase String. Sin embargo,

```
String cadena;
```

Se ha declarando un objeto cadena de la clase String, pero aún no se ha creado ningún objeto de esta clase.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

## String

Para obtener la longitud, número de caracteres que tiene un string se utiliza el método **length**.

```
String cadena="El primer string";
int longitud=cadena.length();
```

Si se quiere obtener la posición de la primera ocurrencia de la letra p, se usa método **indexOf**.

```
String cadena="El primer string";
int pos=cadena.indexOf('p');
```

Comparar Strings : método **equals**

```
String cadena1="El lenguaje Java";
String cadena2=new String("El lenguaje Java");
if(cadena1==cadena2){
    System.out.println("Los mismos objetos");
}else{
    System.out.println("Distintos objetos");
}
if(cadena1.equals(cadena2)){
    System.out.println("El mismo contenido");
}else{
    System.out.println("Distinto contenido");
}
```

```
String cadena1="El lenguaje Java";
String cadena2=cadena1;
System.out.println("Son el mismo objeto "+(cadena1==cadena2));
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## ÍNDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

## String

```
String cadena="El lenguaje Java";  
boolean resultado=cadena.equals("El lenguaje Java");
```

El método **compareTo** devuelve un entero menor que cero si el objeto string es menor (en orden alfabético) que el string dado, cero si son iguales, y mayor que cero si el objeto string es mayor que el string dado.

```
String cadena="Tomás";  
int resultado=cadena.compareTo("Alberto");
```

La variable entera resultado tomará un valor mayor que cero, ya que Tomás está después de Alberto en orden alfabético.

Para convertir un número en string se emplea el método estático **valueOf**

```
int valor=10;  
String str=String.valueOf(valor);
```

La clase String proporciona versiones de **valueOf** para convertir los datos primitivos: int, long, float, double.

Para convertir un string a número entero

```
String cadena="12";  
int numero=Integer.valueOf(cadena).intValue();
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Arrays

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Arrays

Matrices, arrays o vectores en java.

```
import java.util.*
```

Java posee la capacidad de definir un conjunto de variables del mismo tipo agrupadas todas ellas bajo un mismo nombre, y distinguiéndolas mediante un índice numérico. En Java, los Array pueden ser de cualquier tipo de dato, incluidos objetos.

Para definir un array en java es como definir una variable o atributo, pero al especificar el tipo lo que hacemos es colocar un par de corchetes [] para indicar que lo que estamos definiendo es un array.

Para declarar un array de enteros escribimos  
int[] numeros;

Para crear un array de 4 número enteros escribimos  
numeros=new int[4];

La declaración y la creación del array se puede hacer en una misma línea.  
int[] numeros =new int[4];



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Arrays

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Arrays

Para inicializar el array de 4 enteros escribimos

```
numeros[0]=2; numeros[1]=-4; numeros[2]=15; numeros[3]=-25;
```

Se pueden inicializar en un bucle **for** como resultado de alguna operación

```
for(int i=0; i<4; i++){ numeros[i]=i*i+4; }
```

No necesitamos recordar el número de elementos del array, el método *length* nos proporciona la dimensión del array. Escribimos de forma equivalente

```
for(int i=0; i<numeros.length; i++){ numeros[i]=i*i+4; }
```

Una matriz bidimensional puede tener varias filas, y en cada fila no tiene por qué haber el mismo número de elementos. Por ejemplo

```
double[][] matriz={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};
```

La primer fila tiene cuatro elementos {1,2,3,4}

La segunda fila tiene dos elementos {5,6}

La tercera fila tiene seis elementos {7,8,9,10,11,12}

La cuarta fila tiene un elemento {13}

Para mostrar los elementos de este array bidimensional escribimos el siguiente código

```
for (int i=0; i < matriz.length; i++)
{ for (int j=0; j < matriz[i].length; j++)
{ System.out.print(matriz[i][j]+", ");
}
System.out.println(""); }
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Métodos

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

- La cabecera de un método consiste en un nombre, una lista de parámetros (que puede ser vacía), y el tipo de resultado (si no devuelve resultado, el método será de tipo void).
- La sintaxis (cabecera o prototipo) en Java es:  
**<tipo devuelto> <nombre del método> (<lista de argumentos>)**
- Siendo la sintaxis de  
**<lista de argumentos> la siguiente: <tipo> <arg1>, <tipo><arg2>, ...**
- **float media(int[] dato1,int nE)**
- Las sintaxis de las llamadas es similar a la de las cabeceras, si bien en este caso se utiliza una versión “reducida” de la **<lista de argumentos>** en la que solo aparecen sus nombres (reales), separados por comas.
- Los métodos que no sean de tipo void deben acabar con la sentencia: **return <variableDeTrabajo>;**



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)[Bucles](#)[Métodos](#)

- En java los parámetros se pasan solo por valor, es decir una vez terminada la ejecución del método las variables pasadas tienen el mismo valor que antes de invocar dicho método.

```
static float media(int[] dato1,int nE) {  
    float resul=0;  
    for( int i=0; i<nE; i++)  
        resul+=dato1[i];  
    return resul/nE; }
```

```
static int min(int[]dato1, int nE)  
{  
    int resul;  
    resul = dato1[0];  
    for(int i=1; i<nE;i++)  
        if(dato1[i] <resul)  
            resul = dato1[i];  
    return resul;  
}
```

```
static int max(int[]dato1, int nE)  
{  
    int resul;  
    resul = dato1[0];  
    for(int i=1; i<nE;i++)  
        if(dato1[i] > resul)  
            resul = dato1[i];  
    return resul;
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

**INDICE****Introducción****El lenguaje Java****Sintaxis****Tipos de Datos****Operadores****Bifurcaciones****Bucles****Métodos**

```
public static void main(String[] args) throws NumberFormatException, IOException
{
    int rango;
    int numero;
    BufferedReader linea = new BufferedReader(new InputStreamReader(System.in));
    System.out.println ("Introduce el numero de elementos: ");
    numero= Integer.parseInt (linea.readLine());
    int[] d = new int[numero];
    float media;
    System.out.println ("Introduce " + numero + " numeros enteros: ");
    for(int i=0;i<numero;i++)
        d[i] = Integer.parseInt (linea.readLine());
    media = media (d,numero);
    System.out.println("El valor medio es: " + media);
    rango = max(d,numero) - min (d,numero);
    System.out.println("y el rango: " + rango);
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## WRAPPERS

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

- WRAPPERS (envoltorios) son clases diseñadas para ser un complemento de los tipos primitivos. En efecto, los tipos primitivos son los únicos elementos de Java que no son objetos.. Los tipos primitivos siempre se pasan como argumento a los métodos por valor, mientras que los objetos se pasan por referencia. No hay forma de modificar en un método un argumento de tipo primitivo y que la modificación persista. Una forma de conseguir esto es utilizar un Wrapper, esto es un objeto cuya variable miembro es el tipo primitivo que se quiere modificar. Las clases Wrapper también proporcionan métodos para realizar otras tareas con los tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.
- Existe una clase Wrapper para cada uno de los tipos primitivos numéricos, esto es, existen las clases Byte, Short, Integer, Long, Float y Double (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de Java)



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

WRAPPERS

## ÍNDICE

[Introducción](#)[El lenguaje Java](#)[Sintaxis](#)[Tipos de Datos](#)[Operadores](#)[Bifurcaciones](#)[Bucles](#)[Métodos](#)

- Clase Integer La clase `java.lang.Integer` tiene como variable miembro un valor de tipo int.

| Métodos                                                                                                                                                               | Función que desempeñan                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <code>Integer(int)</code> y <code>Integer(String)</code>                                                                                                              | Constructores de la clase                                                              |
| <code>doubleValue()</code> , <code>floatValue()</code> , <code>longValue()</code> , <code>intValue()</code> ,<br><code>shortValue()</code> , <code>byteValue()</code> | Conversores con otros tipos primitivos                                                 |
| <code>Integer decode(String)</code> , <code>Integer parseInt(String)</code> ,<br><code>String toString()</code> , <code>Integer valueOf(String)</code>                | Conversores con String                                                                 |
| <code>String toBinaryString(int)</code> , <code>String toHexString(int)</code> ,<br><code>String toOctalString(int)</code>                                            | Conversores a cadenas representando enteros en otros sistemas de numeración            |
| <code>Integer getInteger(String)</code>                                                                                                                               | Determina el valor de una propiedad del sistema a partir del nombre de dicha propiedad |
| <code>MAX_VALUE</code> , <code>MIN_VALUE</code> , <code>TYPE</code>                                                                                                   | Constantes predefinidas                                                                |



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

WRAPPERS

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Los Wrappers son utilizados para convertir cadenas de caracteres (texto) en números. Esto es útil cuando se leen valores desde el teclado, desde un fichero de texto.

Ejemplo:

```
String numEntero= "8978";
Int numEntero=Integer.valueOf(numEntero).intValue(); // numEntero = 8979
double numDouble=Double.valueOf(numEntero).doubleValue();//
numDouble = 8979
String numEntString = "1111";
numEntero=Integer.valueOf(numEntString).intValue(); // numEntero= 1111
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## Recursividad

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

- Se define objeto recursivo como: “aquel que forma parte de sí mismo o que forma parte de su propia definición”.
- Ejemplos:
  - número natural:
    - El 1 es un número natural.
    - El siguiente de un número natural es un número natural.
  - la definición de factorial de número natural:
    - $0!=1;$
    - $N > 0, N! = N * (N-1)!$
- La recursividad en programación se obtiene mediante subprogramas que se llaman a sí mismos
- creando otra copia o instancia de ellos al hacerlo, así sucesivamente hasta que se alcanza una condición denominada de terminación o de parada.
- Una vez que se alcanza la condición de terminación, en la instancia n-ésima se retorna al punto del modulo de llamada de la instancia previa desde donde esta se realizó, y así sucesivamente.

i



E.T.S.I.S.I.

UNIVERSIDAD  
POLITÉCNICA DE MADRID

Java

Recursividad

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

Para todo **entero positivo**  $n$ , el **factorial de  $n$**  o  **$n$  factorial** o **factorial de  $n$**  se define como el **producto** de todos los números enteros positivos desde 1 hasta  $n$

- $n! = n \times (n-1)!$

**int factorial (int dato)**

```
{ int resul = 1;  
if (dato > 0) resul = dato * factorial (dato - 1);  
return resul;  
}
```



E.T.S.I.S.I.

UNIVERSIDAD  
POLITÉCNICA DE MADRID

Java

Recursividad

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

**Sucesión de Fibonacci:** La sucesión inicia con 0 y 1, y a partir de ahí cada elemento es la suma de los dos anteriores.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584

```
public class SerieFibonacci {  
    static void serieFibonacci(int numEle, int penultimo, int ultimo)  
    { int aux;  
        if (numEle > 2)  
        { aux = ultimo+penultimo;  
            System.out.print(aux+", ");  
            serieFibonacci(numEle-1,ultimo,aux);  
        }  
        else System.out.println("Fin del listado"); }  
    public static void main (String [ ] args) throws IOException  
    {int dato;  
        BufferedReader linea = new BufferedReader (new InputStreamReader(System.in));  
        System.out.print("Introduzca el dato: ");  
        dato = Integer.parseInt (linea.readLine ());  
        System.out.print(0+", "+1+", ");  
        serieFibonacci(dato,0,1); }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD  
POLITÉCNICA DE MADRID

Java

Recursividad

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

**Sucesión de Fibonacci:** La sucesión inicia con 0 y 1, y a partir de ahí cada elemento es la suma de los dos anteriores.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584

```
public class SerieFibonacci {  
    static int fibonacci(int n){  
        if(n==1) return 0;  
        else if (n==2) return 1;  
        else return fibonacci(n-1)+fibonacci(n-2);  
    }  
}
```

```
public static void main (String [ ] args) throws IOException  
{int dato;  
    BufferedReader linea = new BufferedReader (new InputStreamReader(System.in));  
    System.out.print("Introduzca el dato: ");  
    dato = Integer.parseInt (linea.readLine ());  
    for(int i=1; i<=dato;i++)  
        if (i!=dato) System.out.print(fibonacci(i)+ " , ");  
        else System.out.println(fibonacci(i));}  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

Recursividad

## INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

- 1.- Método recursivo que permita hacer la división por restas sucesivas.
- 2.- Método recursivo que permita sumar los elementos de un vector.
- 3.- Método recursivo que permita sumar los dígitos de un número.
- 4.- Método recursivo que determine si una palabra es Palíndroma.
- 5.- Método recursivo que permita invertir un número.
- 6.- ~~Método recursivo que calcule el Máximo común divisor de dos números.~~
- 7.- Método recursivo que muestre el numero menor de un vector.
- 8.- Método que calcule si un numero es primo
- 9.- Crear una clase para cada método de los anteriores y una Clase con un objeto de cada una de ellas, esta clase deberá mostrar un menú pidiendo la operación que se quiere realizar pedir los datos y mostrar por pantalla el resultado de la misma.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## Recursividad Indirecta

### INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

```
import java.io.*;
public class RecursividadIndirecta {
    static boolean esPar (int n) {
        boolean resul = true;
        if (n != 0) resul = esImpar (n - 1);
        return resul; }
    static boolean esImpar (int n) {
        boolean resul = false;
        if (n != 0) resul = esPar (n - 1);
        return resul; }
    public static void main (String [ ] args) throws IOException {
        int n;
        boolean esPar;
        BufferedReader linea = new BufferedReader (new InputStreamReader (System.in));
        System.out.print ("Escriba un número para saber si es par o impar: ");
        n = Integer.parseInt (linea.readLine ());
        esPar = esPar (n);
        if (esPar) System.out.println ("El número " + n + " es par");
        else System.out.println ("El número " + n + " es impar");
    }
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## Recursividad Indirecta

### INDICE

Introducción

El lenguaje Java

Sintaxis

Tipos de Datos

Operadores

Bifurcaciones

Bucles

Métodos

Recursividad

```
public boolean par(int n)
```

```
{
```

```
    if(n==0) return true;  
    else return impar(n-1);
```

```
}
```

```
public boolean impar(int n)
```

```
if(n==0) return false;  
else return par(n-1);
```

```
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

### Paquetes

#### Interfaces

Métodos

#### Clases

Variables

Métodos

#### Excepciones



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- Javadoc es una herramienta muy interesante del kit de desarrollo de Java para generar automáticamente documentación Java. genera documentación para paquetes completos o para archivos java.
- **javadoc archivo.java o paquete**
- En el código javadoc se pueden usar **etiquetas especiales**, las cuales comienzan con el símbolo @ :
  - **@author**. el autor del documento.
  - **@version**. el número de versión de la aplicación
  - **@see**. indica una referencia a otro código Java relacionado con éste.
  - **@since**. Indica desde cuándo esta disponible este código
  - **@deprecated**. Palabra a la que no sigue ningún otro texto en la línea y que indica que esta clase o método está obsoleta u obsoleto.
  - **@throws**. Indica las excepciones que pueden lanzarse en ese código.
  - **@param**. Palabra a la que le sigue texto qué describe a los parámetros que requiere el código para su utilización (el código en este caso es un método de clase). Cada parámetro se coloca en una etiqueta **@param** distinta, por lo que puede haber varios **@param** para el mismo método.
  - **@return**. Tras esta palabra se describe los valores que devuelve el código (el código en este caso es un método de clase)

## INDICE

- En el lenguaje Java, la clase `Exception` define condiciones de error que los programas pueden encontrar. En vez de dejar que el programa termine, se puede escribir código para gestionar estas condiciones de error y continuar con la ejecución del programa, se producen por ejemplo cuando:
  - El fichero que se quiere abrir no existe.
  - La conexión de red se ha perdido.
  - Los operando que se manejan se salen de rango.
  - El fichero de clase que se quiere cargar no existe.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

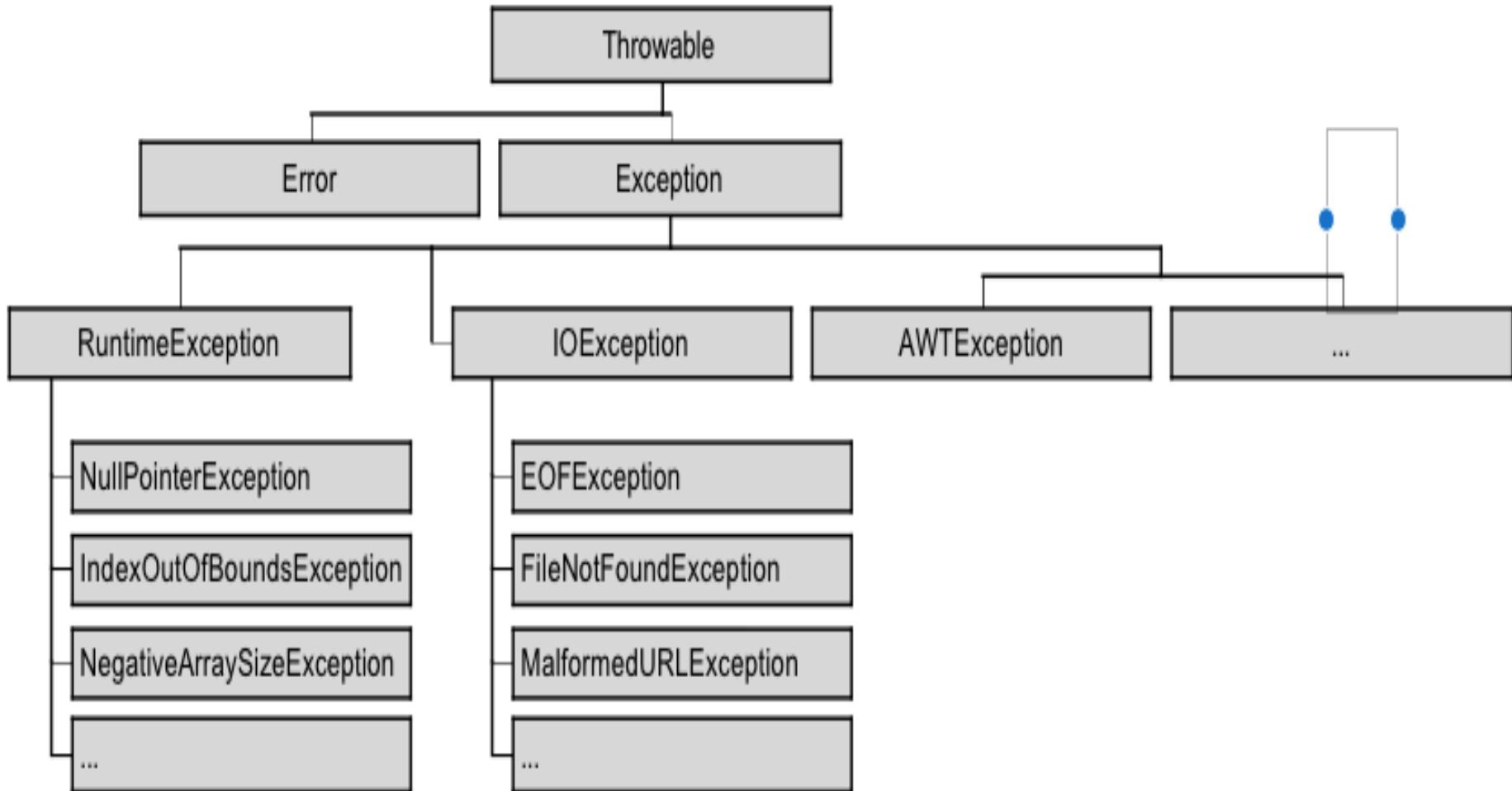
Java

## INDICE

- Mediante el uso de excepciones para controlar errores, los programas Java tienen las siguientes ventajas frente a las técnicas de manejo de errores tradicionales.
- **Separar el Manejo de Errores del Código "Normal". Estará en una zona separada donde podremos tratar las excepciones como un código 'especial'.**
- **Propagar los Errores sobre la Pila de Llamadas podemos propagar el error a la primera función que llamó a las diversas funciones hasta que llegamos al error.**
- **Agrupar Errores y Diferenciación. Gracias a esto tenemos todos los posibles errores juntos y podemos pensar una manera de tratarlos que sea adecuado.**

# Jerarquía de excepciones

INDICE



## ÍNDICE

- Categorías de excepciones: tres categorías genéricas de excepciones
  - Las clases que descienden de Error indican un problema bastante serio del cual la recuperación es prácticamente imposible Un ejemplo es la ejecución sin memoria.
  - RuntimeException y descendientes sirven para indicar un error de diseño o implementación del programa. Es decir, indica condiciones que no deberían pasar nunca si se programa cuidadosamente. La excepción `ArrayIndexOutOfBoundsException`
  - Otras excepciones (las que heredan de Exception pero no de RuntimeException) indican una dificultad en tiempo de ejecución debido a errores en el entorno de ejecución o de usuario. Un ejemplo claro de esto puede ser un fichero no encontrado..



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- La regla de "declarar o capturar", A continuación de la palabra reservada throws aparece una lista de todas las excepciones que se pueden dar dentro del método y no serán gestionadas por él, Es obligatorio declarar las excepciones no capturadas excepto para aquellas que descienden de Error o de RuntimeException.
- Si un método llama a otros métodos que pueden lanzar excepciones tiene 2 posibilidades:
  - Capturar las posibles excepciones y gestionarlas.
  - Desentenderse de las excepciones y remitirlas hacia otro método anterior en el stack para éste se encargue de gestionarlas.
- todos los tipos de excepciones pueden usar los métodos siguientes:
  - String getMessage(): Extrae el mensaje asociado con la excepción.
  - String toString(): Devuelve un String que describe la excepción.
  - void printStackTrace() Indica el método donde se lanzó la excepción.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- LANZAR UNA EXCEPTION
- Cuando en un método se produce una situación anómala es necesario lanzar una excepción. El proceso de lanzamiento de una excepción es el siguiente:
  - Se crea un objeto Exception de la clase adecuada.
  - Se lanza la excepción con la sentencia throw seguida del objeto Exception creado.
- // Código que lanza la excepción MiExcepcion una vez detectado el error  

```
MiExcepcion me = new MiExcepcion("Mensaje de MiExcepcion ");
throw me;
```
- Esta excepción deberá ser capturada (catch) y gestionada en el propio método o en algún otro lugar del programa (en otro método anterior en la pila o stack de llamadas)
- Al lanzar una excepción el método termina de inmediato, sin devolver ningún valor. Solamente en el caso de que el método incluya los bloques try/catch/finally se ejecutará el bloque catch que la captura o el bloque finally (si existe).
- Todo método en el que se puede producir uno o más tipos de excepciones (y que no utiliza directamente los bloques try/catch/finally para tratarlos) debe declararlas en el encabezamiento de la función por medio de la palabra throws. Si un método puede lanzar varias excepciones, se ponen detrás de throws separadas por comas .



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- CAPTURAR UNA EXCEPTION
  - Gestionar la excepción con una construcción del tipo try {...} catch {...}.
  - Re-lanzar la excepción hacia un método anterior en el stack, declarando que su método también lanza dicha excepción, utilizando para ello la construcción throws en la cabecera del método.
- Sentencias try y catch
  - En el caso de las excepciones que no pertenecen a las RuntimeException y que por lo tanto Java obliga a tenerlas en cuenta habrá que utilizar los bloques try, catch y finally.
  - Para capturar una excepción en particular hay que colocar el código que puede lanzar dicha excepción dentro de un bloque try, entonces se crea una lista de bloques catch adyacentes, uno para cada posible excepción que se quiera capturar. El código de un bloque catch se ejecuta cuando la excepción generada coincide en tipo con la del bloque catch
  - El bloque finally es opcional. Si se incluye sus sentencias se ejecutan siempre, sea cual sea la excepción que se produzca o si no se produce ninguna. El bloque finally se ejecuta aunque en el bloque try haya un return.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

### □ Relanzar una Excepción

- Existen algunos casos en los cuales el código de un método puede generar una Excepción y no se desea incluir en dicho método la gestión del error. Java permite que este método pase o relance (throws) la Excepción al método desde el que ha sido llamado, sin incluir en el método las instrucciones try/catch correspondientes. Esto se consigue mediante la adición de throws más el nombre de la Excepción concreta después de la lista de argumentos del método.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

En el siguiente ejemplo se presenta un método que debe "controlar" una IOException relacionada con la lectura ficheros y una MiExcepcion propia:

```
void metodo()
{
    ...
    try {// Código que puede lanzar las excepciones IOException y MiExcepcion
        }
    catch (IOException excep1)
    { // Se ocupa de IOException simplemente dando aviso
        System.out.println(excep1.getMessage());
    }
    catch (MiExcepcion excep2)
    { // Se ocupa de MiExcepcion dando un aviso y finalizando la función
        System.out.println(excep2.getMessage());
        return;
    }
    finally { // Sentencias que se ejecutarán en cualquier caso ... }

}
// Fin del método
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- CREAR NUEVAS EXCEPCIONES
- El programador puede crear sus propias excepciones sólo con heredar de la clase Exception o de una de sus clases derivadas. Lo lógico es heredar de la clase de la jerarquía de Java que mejor se adapte al tipo de excepción. Las clases Exception suelen tener dos constructores:
  - 1. Un constructor sin argumentos.
  - 2. Un constructor que recibe un String como argumento. En este String se suele definir un mensaje que explica el tipo de excepción generada. Conviene que este constructor llame al constructor de la clase de la que deriva super(String).
- Por ejemplo:

```
class MiExcepcion extends Exception {  
    public MiExcepcion() { super(); }  
    public MiExcepción(String s) { super(s); } } // Constructor por defecto //
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
public DividePorCeroException extends ArithmeticException
{
    public DividePorCeroException(String Mensaje)
    {
        super(mensaje);
    }

    public double dividir(int num, int den) throws DividePorCeroException
    {
        if (den==0) throw new DividePorCeroException("Error ");
        return ((double) num/ (double)den);
    }
}
```



## ÍNDICE

- 1.- Desarrolle un método al que le pasamos el nombre y apellidos de una persona como un sólo String con el típico formato "apellidos, nombre". El método nos devolverá un nuevo formato: "nombre apellidos". System.out.println (miMetodo ("García, Juan.")); imprime Juan García:
- 2.- Dada una variable entera t que almacena el tiempo transcurrido del día en segundos, escriba las expresiones que permitan extraer las horas, minutos y segundos transcurridos del día (h, m, s). Ejemplo:
- 3.- Una fuente de datos registra varias edades, la edad 0 indica el final de los datos, realice un programa para determinar el promedio de las edades ingresadas y además el porcentaje de personas mayores a los 50 años.
- 4.-Desarrollar un programa que, utilizando un metodo muestre en pantalla N filas de números naturales impares, de los siguientes números y en la forma siguiente:

1  
1 3  
1 3 5  
1 3 5 7  
1 3 5 7 9 |  
1 3 5 7 9 11  
1 3 5 7 9 11 13

- 5.- Crea un array con un tamaño pedido por teclado y con valores introducidos por el usuario. Muestra por consola el indice y el valor al que corresponde. Haz dos métodos, uno para llenar valores y otro para mostrar.



## ÍNDICE

- 6.- Escribir un programa que llene un vector con una lista de números del 1 al 20, luego despliegue este vector indicando a la derecha de cada uno si es divisible por 3 o no.
- 7.- Hacer un clase que lea 10 valores enteros en un array desde el teclado y calcule y muestre: la suma, el valor medio, el mayor y el menor
- 8.- Se tienen N temperaturas almacenados en un vector, se desea calcular su media y cuáles de las temperaturas son inferiores a la media.
- 9.- Programa para ingresar n valores reales en un vector y luego invierta el vector.
- 10.- Calcula la letra de un DNI, pediremos el DNI por teclado y nos devolverá el DNI completo.
- 11.- Crea un array de números y otro de String de 10 posiciones donde insertaremos notas entre 0 y 10 (debemos controlar que inserte una nota valida), pudiendo ser decimal la nota con la coma como indicador de carácter decimal en el array de números, Si la nota no es una nota valida deberá lanzar una excepción para informa al usuario, Para ello habrá que crear primero la clase de esa excepción, en el de Strings se insertaran los nombres de los alumnos. Después, crearemos un array de String donde insertaremos el resultado de la nota con palabras.
  - Si la nota esta entre 0 y 4,99 , será un suspenso
  - Si esta entre 5 y 6,99 , será un bien.
  - Si esta entre 7 y 8,99 será un notable.
  - Si esta entre 9 y 10 será un sobresaliente.
  - Muestra por pantalla, el alumno su nota y su resultado en palabras.



## ÍNDICE

- 12.- crea una aplicación que pida un numero por teclado y después comprobaremos si el numero introducido es capicúa, es decir, que se lee igual sin importar la dirección. Por ejemplo, si introducimos 30303 es capicúa, si introducimos 30430 no es capicúa.
- 13.- Hacer un programa que ingrese una cadena de caracteres y determine el número de mayúsculas y el número de minúsculas.
- 14.- Escribir un programa que reciba como datos una cadena de caracteres y un carácter e indique el número de veces que se encuentra el carácter en la cadena y devuelva una cadena donde se han eliminado los espacios en blanco.



# Tabla cálculo DNI

## INDICE

| Posicion | Letra |
|----------|-------|
| 0        | T     |
| 1        | R     |
| 2        | W     |
| 3        | A     |
| 4        | G     |
| 5        | M     |
| 6        | Y     |
| 7        | F     |
| 8        | P     |
| 9        | D     |
| 10       | X     |
| 11       | B     |
| 12       | N     |
| 13       | J     |
| 14       | Z     |
| 15       | S     |
| 16       | Q     |
| 17       | V     |
| 18       | H     |
| 19       | L     |
| 20       | C     |
| 21       | K     |
| 22       | E     |



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- las clases **FileReader** y **FileWriter** para ficheros de texto en Java.
- Las clases **FileReader** y **FileWriter** permiten leer y escribir, respectivamente, en un fichero.
- Lo primero que debemos hacer es importar estas clases y las que controlan las excepciones.
- Después debemos crear un objeto de alguna de estas clases. Se pueden construir con un objeto **File**, **FileDescriptor** o **un String**.
- Al crear un objeto de estas clases, deben estar dentro de un **try-catch**. Debemos controlar las excepciones.
- Cuando creamos un objeto, abrimos un **stream** entre nuestro programa y el exterior, cuando debemos de usarlo debemos cerrar el **stream** con el método **close()**.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- `FileWriter fw=new FileWriter("D:\\fichero1.txt");`
- `FileReader fr=new FileReader("D:\\fichero1.txt");`
- Como vemos, escribimos la ruta del fichero. Si usas **FileWriter** y escribes una ruta de fichero que no existe lo crea, para **FileReader** si que debe existir el fichero, sino lanzara una excepción. Usamos doble barra (\\\\") por que es un carácter de escape, para poner \.
- Ahora vamos a ver como escribir y leer texto en el fichero.
- Para escribir, usaremos el método **write** de **FileWriter**, este método puede usar como parámetro un String con lo que queremos escribir o un número que se corresponderá un carácter de la tabla ASCII.
- Para leer, usaremos el método **read** de **FileReader**, este método no tiene parámetros pero devuelve un número que si le hacemos un casting a **char** este sera legible por nosotros. Cuando se termina el fichero, el método **read** devuelve -1.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

```
import java.io.*;
public class FicheroTextoApp {
    public static void main(String[] args) {
        try{
            //Abro stream, crea el fichero si no existe
            FileWriter fw=new FileWriter("D:\\fichero1.txt");
            //Escribimos en el fichero un String y un caracter 97 (a)
            fw.write("Esto es una prueba");
            fw.write(97);
            //Cierro el stream
            fw.close();
            //Abro el stream, el fichero debe existir
            FileReader fr=new FileReader("D:\\fichero1.txt");
            //Leemos el fichero y lo mostramos por pantalla
            int valor=fr.read();
            while(valor!=-1){
                System.out.print((char)valor);
                valor=fr.read();
            }
            //Cerramos el stream
            fr.close();
        }catch(IOException e){
            System.out.println("Error E/S: "+e);
        }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## INDICE

Desde java 7 tenemos una mejor forma de abrir y cerrar **stream**, ahorrándonos trabajo.

```
import java.io.*;
//Importamos todas las clases de java.io.
public class FicheroTextoApp {
    public static void main(String[] args) {
        try(FileWriter fw=new FileWriter("D:\\fichero1.txt");
            FileReader fr=new FileReader("D:\\fichero1.txt")){
            //Escribimos en el fichero un String y un caracter 97
            fw.write("Esto es una prueba");
            fw.write(97);
            //Guardamos los cambios del fichero
            fw.flush();
            //Leemos el fichero y lo mostramos por pantalla
            int valor=fr.read();
            while(valor!=-1){
                System.out.print((char)valor);
                valor=fr.read();
            }
        }catch(IOException e){
            System.out.println("Error E/S: "+e);
        }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

Si queremos añadir mas texto a un fichero de texto que ya tenga contenido, al construir el objeto de la clase `FileWriter` añadimos como segundo parámetro un `true`. Si no lo hacemos, sobrescribiremos el fichero entero al escribir algo nuevo.

```
import java.io.*;
//Importamos todas las clases de java.io.
public class FicheroTextoApp {
    public static void main(String[] args) {
        try(FileReader fr=new FileReader("D:\\fichero1.txt");
            FileWriter fw=new FileWriter("D:\\fichero1.txt")){
            escribeFichero(fw);
            //Guardamos los cambios del fichero
            fw.flush();
            leeFichero(fr);
        }catch(IOException e){
            System.out.println("Error E/S: "+e); }
        public static void escribeFichero(FileWriter fw) throws IOException{
            //Escribimos en el fichero
            fw.write("Esto es una prueba"); }
        public static void leeFichero(FileReader fr) throws IOException{
            //Leemos el fichero y lo mostramos por pantalla
            int valor=fr.read();
            while(valor!=-1){
                System.out.print((char)valor);
                valor=fr.read(); } }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

## ÍNDICE

- Las clases **BufferedReader** y **BufferedWriter** las podemos encontrar en **java.io**.
- Estas clases tienen la misma función que **FileReader** y **FileWriter**, leer y escribir en ficheros, pero **BufferedReader** y **BufferedWriter** optimizan estas funciones.
- Se crean igual que **FileReader** y **FileWriter**, pero como parámetro insertaremos un objeto **FileReader** para **BufferedReader** y un objeto **FileWriter** para **BufferedWriter**.
- `FileWriter fw=new FileWriter("D:\\fichero1.txt");`
- `FileReader fr=new FileReader("D:\\fichero1.txt");`
- `BufferedReader br=new BufferedReader(fr);`
- `BufferedWriter bw=new BufferedWriter(fw);`
- Otra forma de crear los buffered es asi:
- `BufferedReader br=new BufferedReader(new FileReader("D:\\fichero1.txt"));`
- `BufferedWriter bw=new BufferedWriter(new FileWriter("D:\\fichero1.txt"));`
- a mayor ventaja de los buffered es con el **BufferedReader** que nos permite leer una linea completa, en lugar de carácter a carácter como hacia **FileReader**, cuando el fichero termina, devuelve **null**, no un -1 como en **FileReader**.
- Con **BufferedWriter** también podemos añadir una linea, como si pulsáramos un Enter.



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

```
import java.io.*; //Importamos todas las clases de java.io.  
public class FicheroTextoBufferedApp {  
    public static void main(String[] args) {  
        try(BufferedReader br=new BufferedReader(new FileReader("D:\\fichero1.txt"));  
            BufferedWriter bw=new BufferedWriter(new FileWriter("D:\\fichero1.txt"))){  
            //Escribimos en el fichero  
            bw.write("Esto es una prueba usando Buffered");  
            bw.newLine();  
            bw.write("Seguimos usando Buffered");  
            //Guardamos los cambios del fichero  
            bw.flush();  
            //Leemos el fichero y lo mostramos por pantalla  
            String linea=br.readLine();  
            while(linea!=null){  
                System.out.println(linea);  
                linea=br.readLine(); }  
        }catch(IOException e){  
            System.out.println("Error E/S: "+e); }  
    }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

```
import java.io.*; //Importamos todas las clases de java.io.  
public class FicheroTextoBufferedApp {  
    public static void main(String[] args) {  
        try(BufferedReader br=new BufferedReader(new FileReader("D:\\fichero1.txt"));  
            BufferedWriter bw=new BufferedWriter(new FileWriter("D:\\fichero1.txt"))){  
            escribeFichero(bw);  
            bw.flush();  
            leeFichero(br);  
        }catch(IOException e){  
            System.out.println("Error E/S: "+e);}  
    }  
    public static void escribeFichero(BufferedWriter bw) throws IOException{  
        bw.write("Esto es una prueba usando Buffered");  
        bw.newLine();  
        bw.write("Seguimos usando Buffered"); }  
    public static void leeFichero(BufferedReader br) throws IOException{  
        //Leemos el fichero y lo mostramos por pantalla  
        String linea=br.readLine();  
        while(linea!=null){  
            System.out.println(linea);  
            linea=br.readLine(); }  
    }  
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

Crea un fichero de texto con el nombre y contenido que tu quieras. Ahora crea una aplicación que lea este fichero de texto carácter a carácter y muestre su contenido por pantalla sin espacios. Por ejemplo, si un fichero tiene el siguiente texto “Esto es una prueba”, deberá mostrar “Estoesunaprueba”.

```
import java.io.FileReader;
import java.io.IOException;
public class Ejercicio1App {
    public static void main(String[] args) {
        final String nomFichero = "\\pruebas.txt";
        try(FileReader fr = new FileReader(nomFichero)){
            int valor = fr.read();
            while(valor != -1)
                {//Si el caracter es un espacio no lo escribe
                    if(valor != 32){
                        System.out.print((char)valor);
                    }
                    valor = fr.read();
                }
        }catch(IOException e){
            System.out.println("Problemas con el E/S "+e);
        }
    }
}
```



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## ÍNDICE

- Crea una aplicación que pida la ruta de dos ficheros de texto y de una ruta de destino. Debes copiar el contenido de los dos ficheros en uno, este se guardara en la ruta donde le hayamos indicado por teclado. Para unir los ficheros en uno, crea un método donde le pases como parámetro todas las rutas. En este método, aparte de copiar debe comprobar que si existe el fichero de destino, nos muestre un mensaje informándonos de si queremos sobrescribir el fichero.
- Haz un programa que lea un fichero de texto y determine el número de caracteres y de entre ellos los que están en mayúsculas y los que están en minúsculas, la cantidad de dígitos (la de números distintos), y el numero de signos de puntuación y el número de palabras del mismo. También hay que crear una clase excepción que se lance cuando se encuentra un carácter erróneo en el texto que informe al usuario y a continuación siga procesando el fichero



E.T.S.I.S.I.

UNIVERSIDAD

POLITÉCNICA DE MADRID

Java

## INDICE

- Haz un programa que dados dos ficheros de texto los cuales contienen una sucesión de números ordenados separados por comas devuelva otro fichero mezcla de estos dos últimos que contendrá los números ordenados de los dos anteriores.
- Hacer un programa que ordene un fichero que contiene números separados por comas mediante el método de mezcla natural.