

# Introducción a Java

Jose Maria Torresano  
Noviembre 2008  
[entornos.jmt@gmail.com](mailto:entornos.jmt@gmail.com)

Empecemos

# 1. INTRODUCCIÓN

*Lo que **Java** intenta hacer y lo hace con bastante éxito, es abarcar dominios diferentes. De esa forma le permite efectuar trabajos para de aplicaciones del lado del servidor, del lado del cliente, para teléfonos móviles, programación científica, software de escritorio, navegación interplanetaria, cualquier cosa...*

**James Gosling**, creador de **Java**

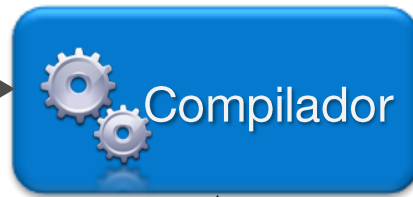
# Por qué Java

- Es un lenguaje moderno y cotizado.
- Se utiliza en la web, negocios, aplicaciones de telecomunicaciones, ...
- Tiene la filosofía de *codificar una vez/ejecutar en cualquier sitio*
- Desarrollado en los 90, incorpora muchas de las características de lenguajes anteriores:
  - Orientación a objetos.
  - Recogida de basura.
  - Portabilidad del código objeto.
  - ...

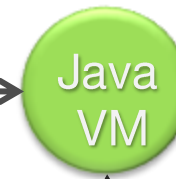
Por medio de la Máquina Virtual de Java (JVM)

# El modo de funcionar de Java

En Java el código fuente se guarda en un archivo de texto terminado en .java



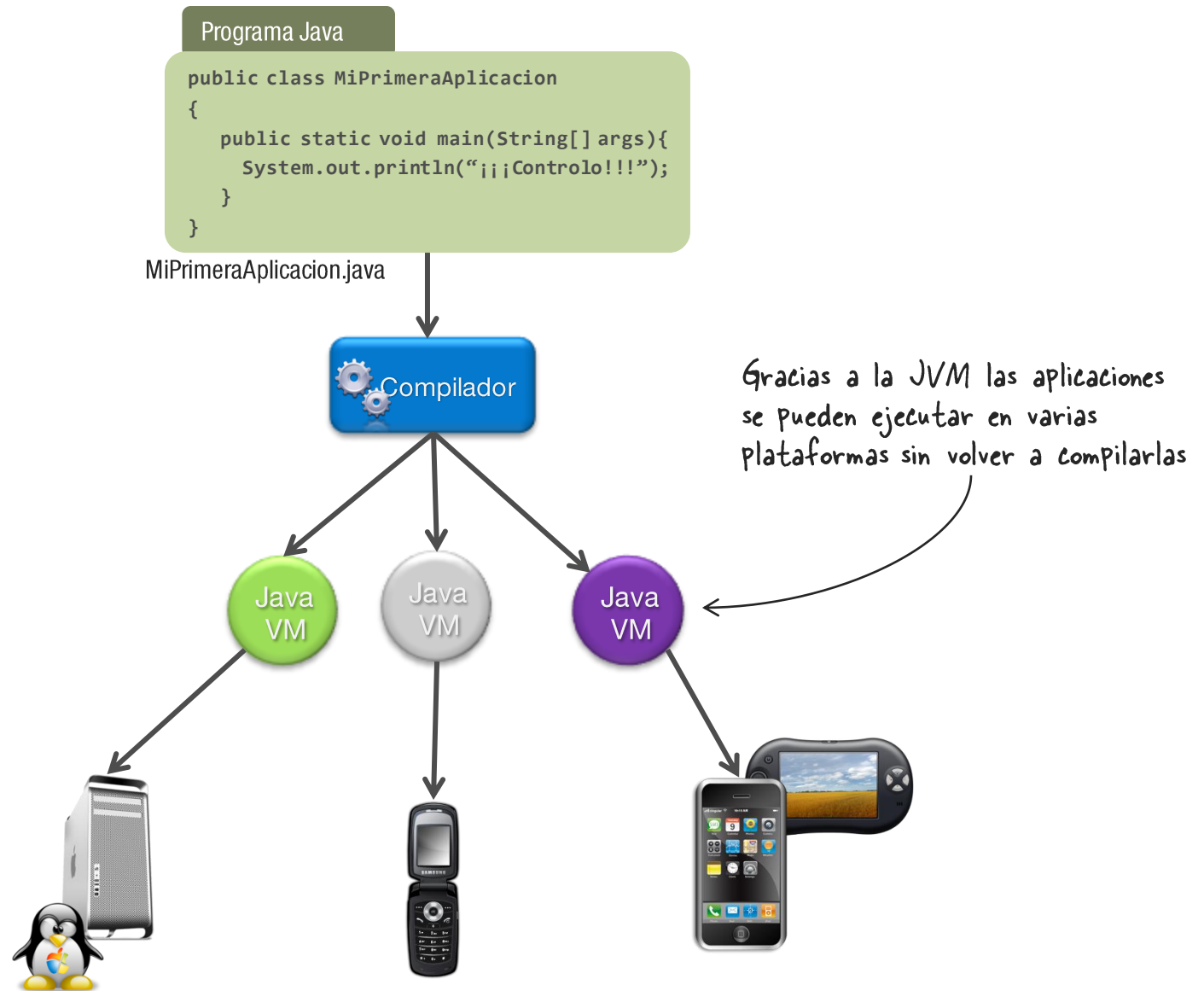
Un archivo .class no contiene código nativo al procesador sino bytecodes



El compilador de Java, javac, convierte el código fuente en un archivo .class

La JVM ejecuta los bytecodes en cada dispositivo

# El modo de funcionar de Java



# La plataforma Java

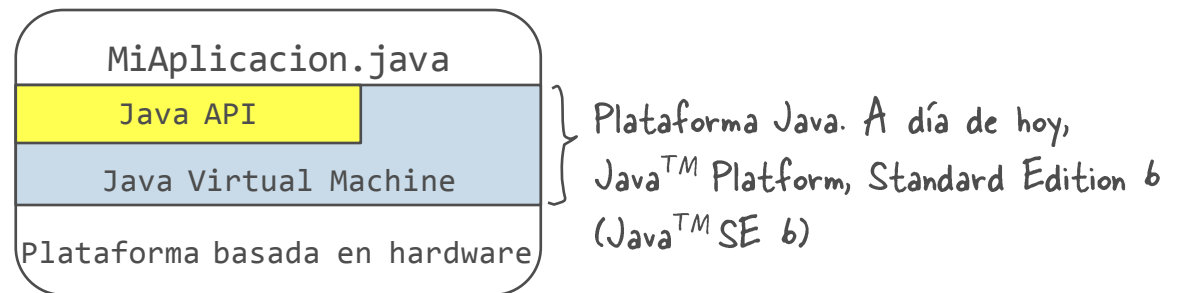
## Plataforma

Las plataformas más populares son Linux, Windows, MacOSX,...

Es el entorno hardware o software donde se ejecuta un programa.

La **plataforma Java** difiere de la mayoría de las demás plataformas en que es una plataforma solo de software que se ejecuta en otras plataformas basada en hardware. Consta de 2 componentes:

1. La Máquina Virtual Java (*Java Virtual Machine*)
2. El Interfaz de Programación de Aplicaciones Java (*Java API*)



# La plataforma Java

El nombre oficial de la plataforma Java es [Java™ Platform, Standard Edition 6](#). Sun utiliza dos formas de números de versión: 1.6.0 para referirse a la versión de la plataforma para desarrollo y 6 para la versión de producto.

En la versión 5 el nombre de la plataforma era J2SE™

	<b>Nombre</b>	<b>Abreviatura</b>
<b>Nombre plataforma</b>	Java™ Platform, Standard Edition 6	Java™ SE6
<b>Productos lanzados bajo la plataforma</b>	Java™ SE Development Kit 6	JDK™ 6
	Java™ SE Runtime Environment 6	JRE™ 6

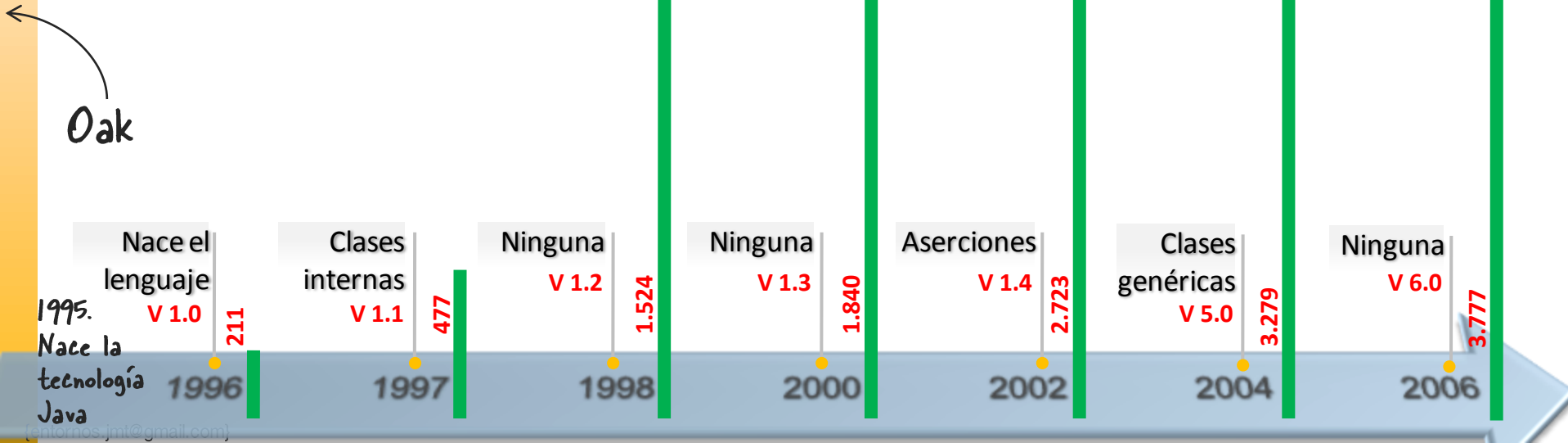
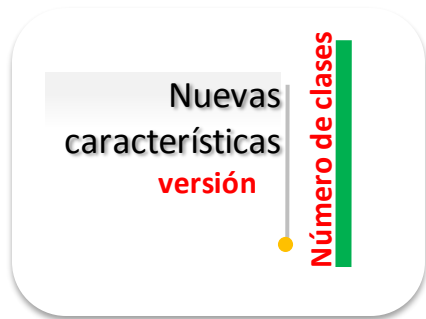


# La plataforma Java

	Java Language								
	java	javac	javadoc	apt	jar	javap	JPDA	JConsole	Java VisualVM
	Security	Int'l	RMI	IDL	Deploy	Monitoring	Troubleshoot	Scripting	JVM TI
	Deployment			Java Web Start			Java Plug-in		
	AWT			Swing			Java 2D		
	Accessibility		Drag n Drop		Input Methods		Image I/O	Print Service	Sound
JDK	IDL	JDBC™		JNDI™		RMI	RMI-IIOP		Scripting
JRE	Beans	Intl Support		I/O	JMX	JNI		Math	
	Networking	Override Mechanism		Security	Serialization	Extension Mechanism		XML JAXP	
	lang and util	Collections	Concurrency Utilities		JAR		Logging	Management	
	Preferences API	Ref Objects	Reflection		Regular Expressions		Versioning	Zip	Instrument
	Java Hotspot™ Client VM					Java Hotspot™ Server VM			
	Solaris™			Linux		Windows		Other	

Java SE API

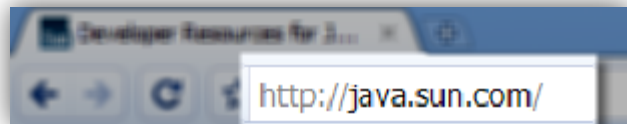
# La plataforma Java



Preparándonos

# 1. INSTALACIÓN JDK

# Java SE instalación



Sun Developer Network (SDN) » search tips Search >

APIs Downloads Products Support Training Participate

- Early Access
- Java SE**
- Java EE
- Java ME
- JavaFX
- Solaris
- NetBeans
- Sun Studio Compilers & Tools
- MySQL
- VirtualBox
- See All »

## Java SE Development Kit (JDK) 6 Update 10

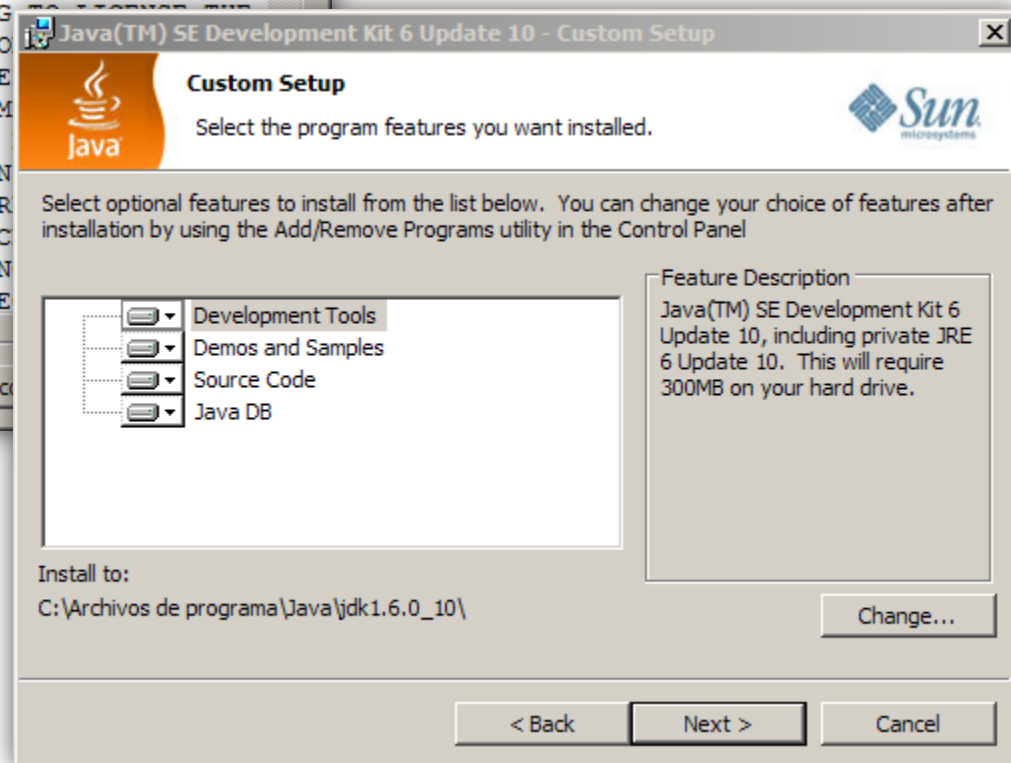
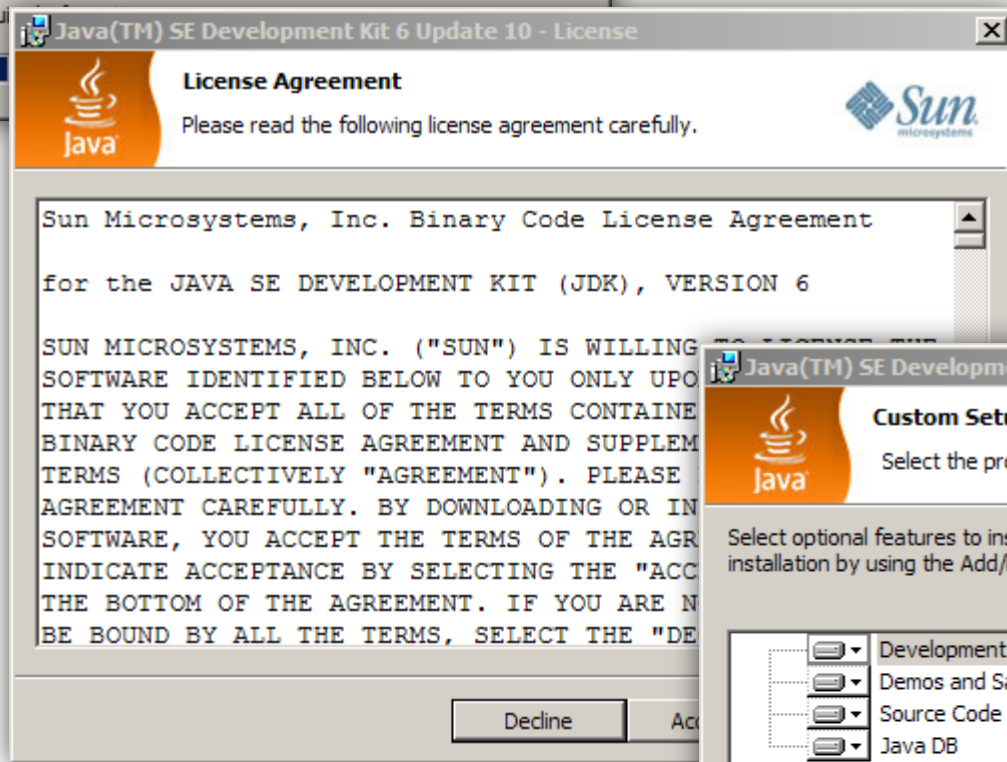
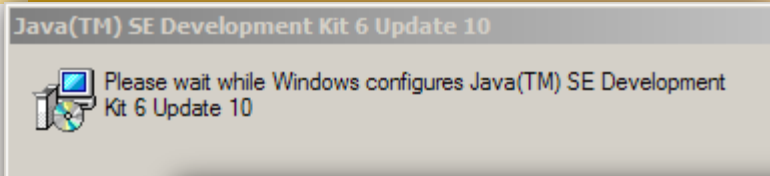
The Java SE Development Kit (JDK) includes the Java SE Runtime Environment (JRE) and command-line development tools that are useful for developing applets and applications.

» Download

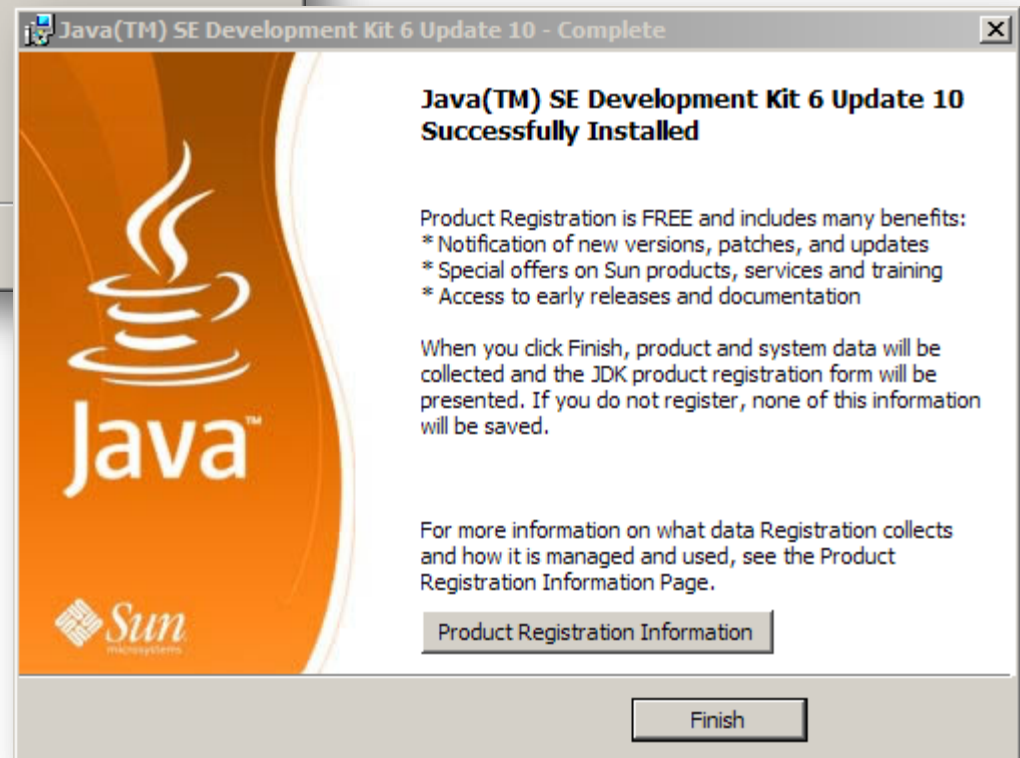
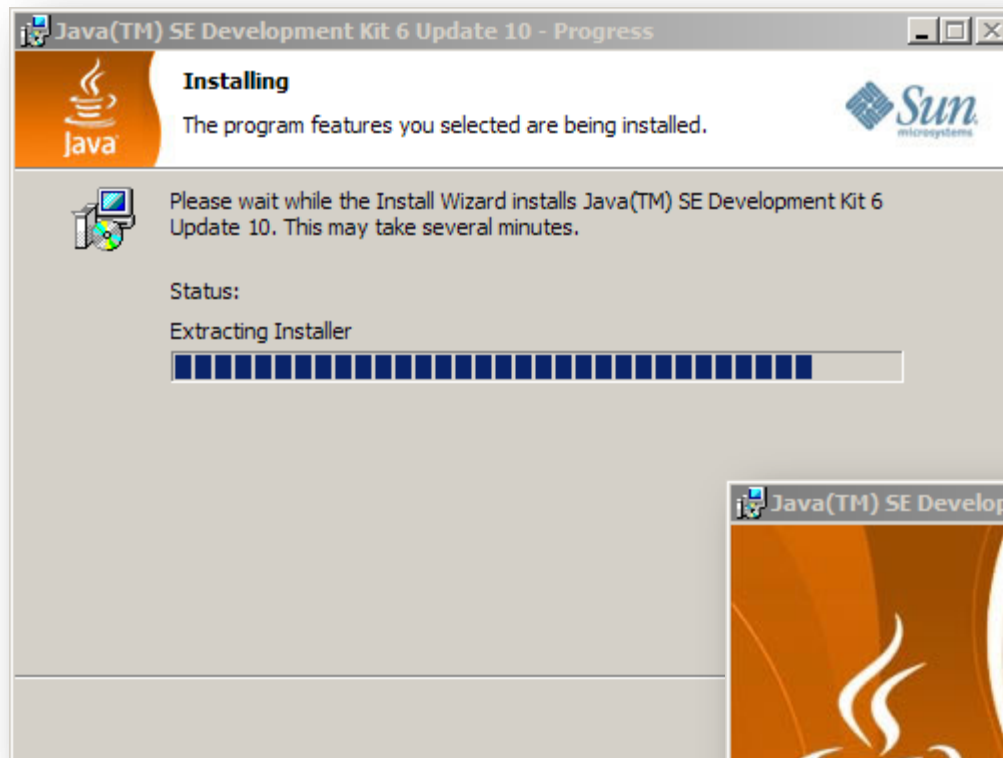
» More info about Java SE 6 Update 10 ...

[FAQ](#) | [Installation Instructions](#) | [ReadMe](#) | [ReleaseNotes](#) | [Sun License](#) | [Third Party Licenses](#)

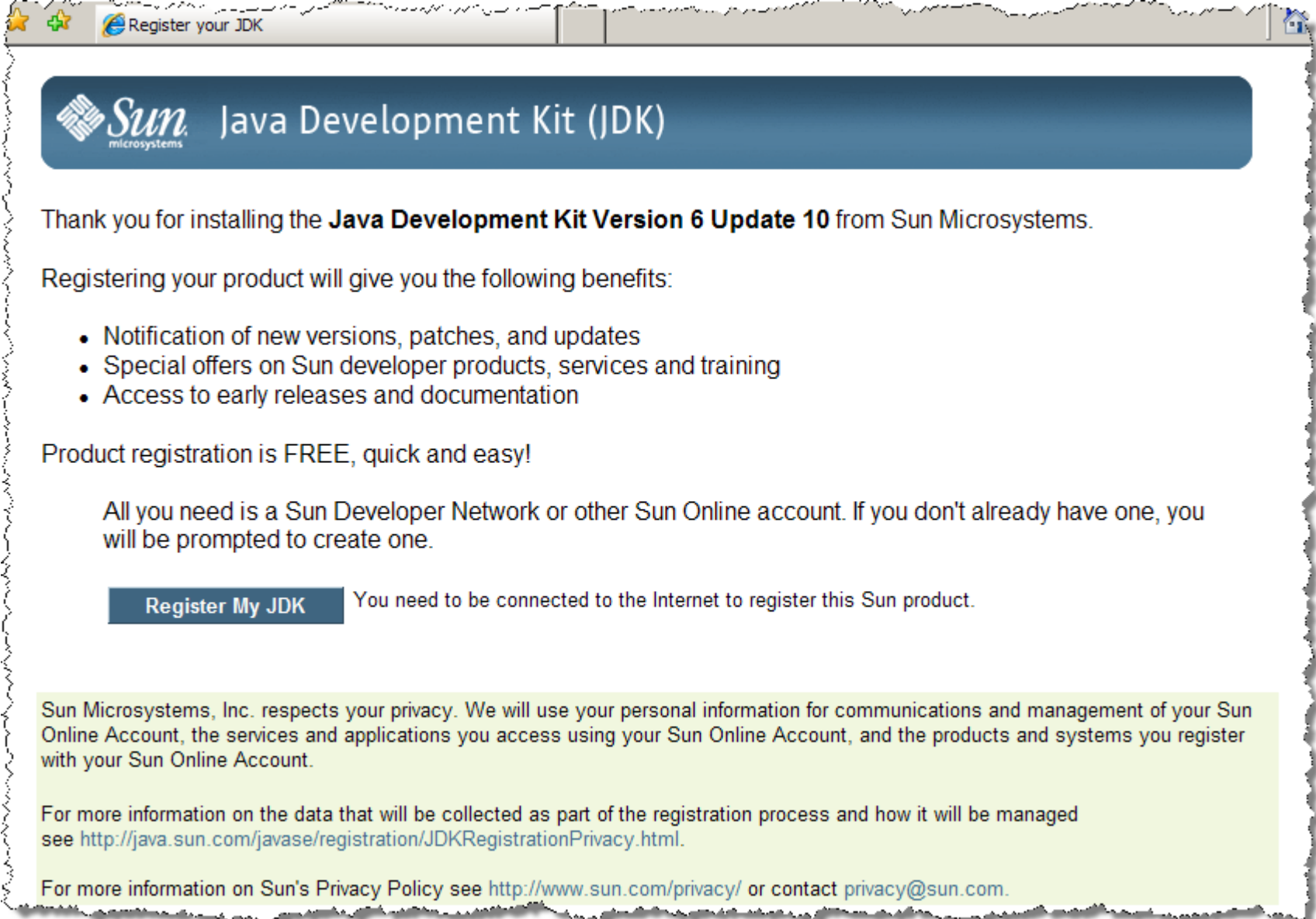
# Java SE instalación




# Java SE instalación



# Java SE instalación



Register your JDK

 **Java Development Kit (JDK)**

Thank you for installing the **Java Development Kit Version 6 Update 10** from Sun Microsystems.

Registering your product will give you the following benefits:

- Notification of new versions, patches, and updates
- Special offers on Sun developer products, services and training
- Access to early releases and documentation

Product registration is FREE, quick and easy!

All you need is a Sun Developer Network or other Sun Online account. If you don't already have one, you will be prompted to create one.

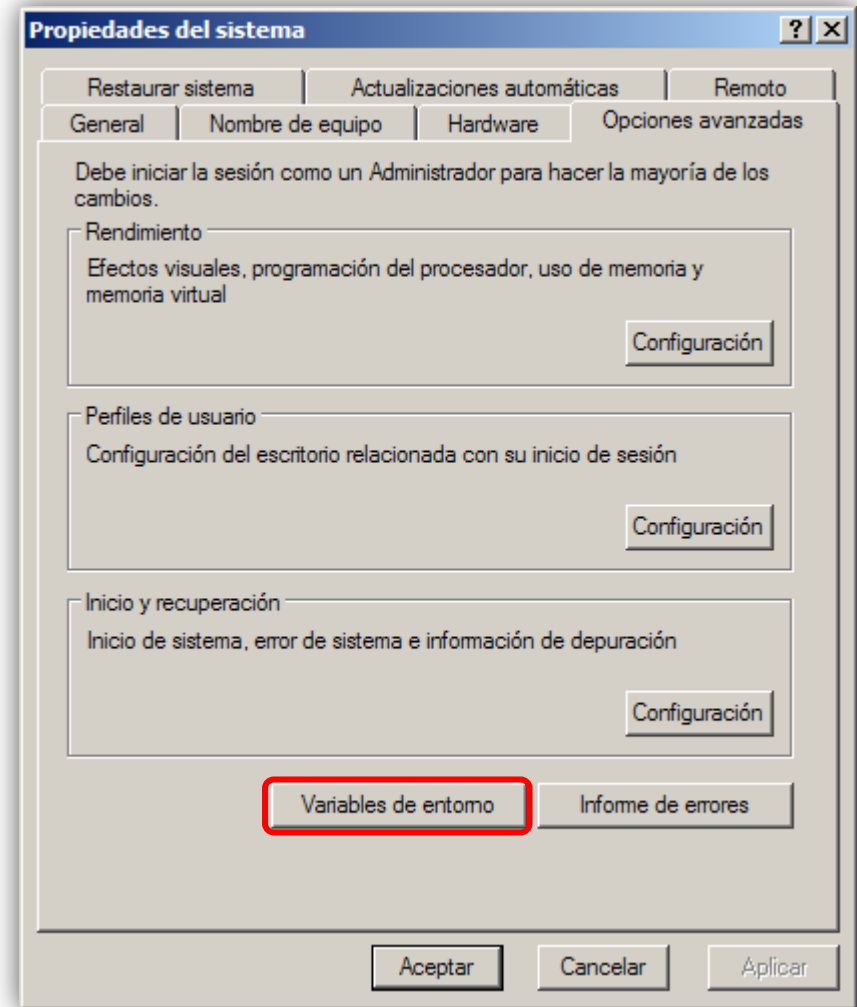
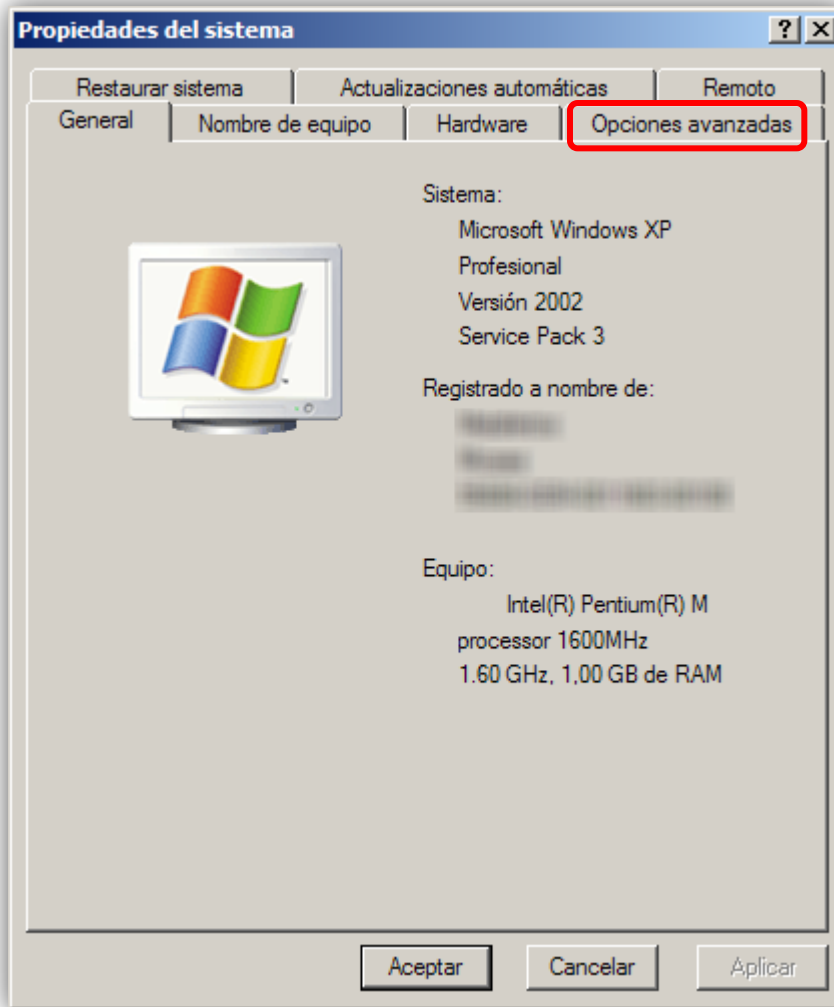
[Register My JDK](#) You need to be connected to the Internet to register this Sun product.

Sun Microsystems, Inc. respects your privacy. We will use your personal information for communications and management of your Sun Online Account, the services and applications you access using your Sun Online Account, and the products and systems you register with your Sun Online Account.

For more information on the data that will be collected as part of the registration process and how it will be managed see <http://java.sun.com/javase/registration/JDKRegistrationPrivacy.html>.

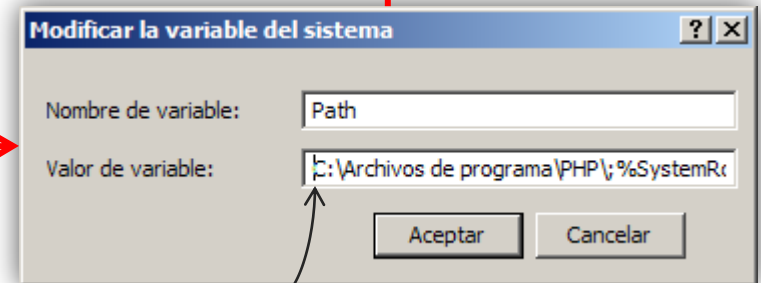
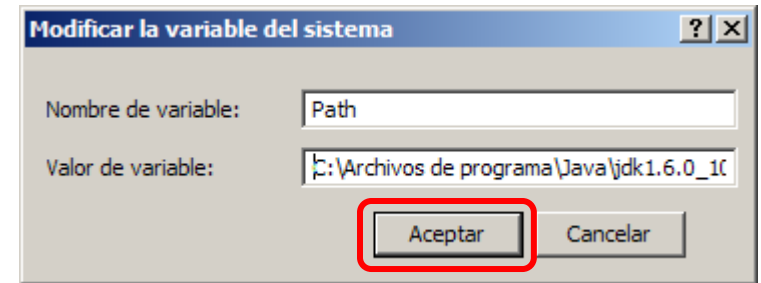
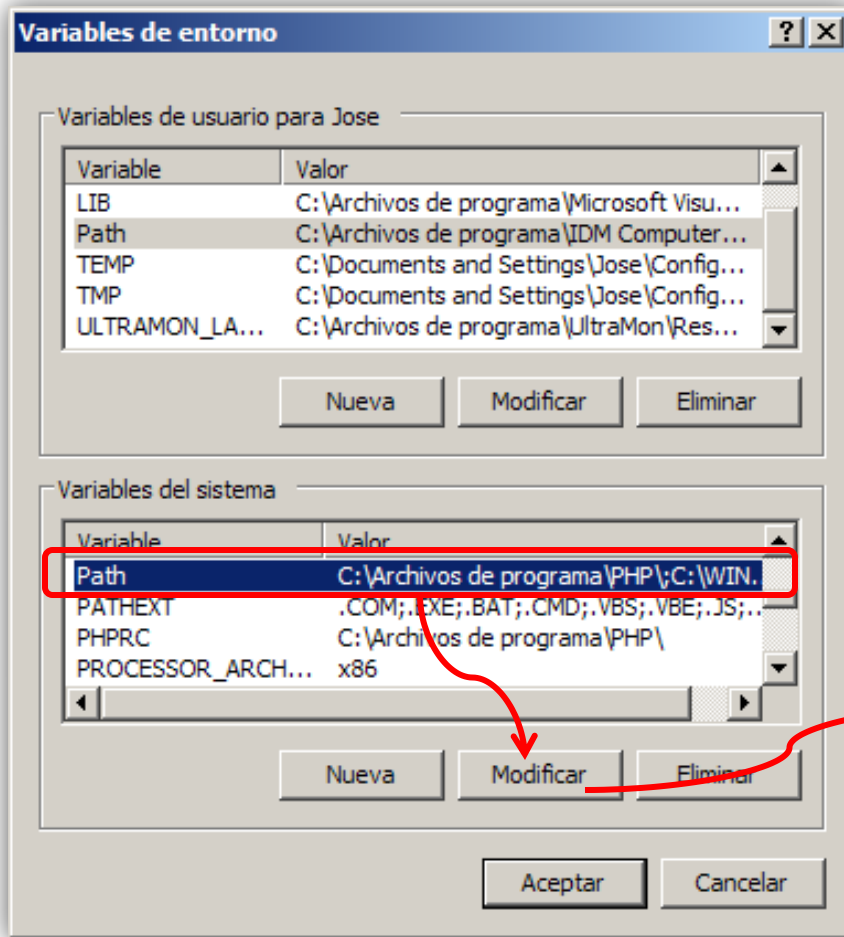
For more information on Sun's Privacy Policy see <http://www.sun.com/privacy/> or contact [privacy@sun.com](mailto:privacy@sun.com).

# Java SE instalación





# Java SE instalación



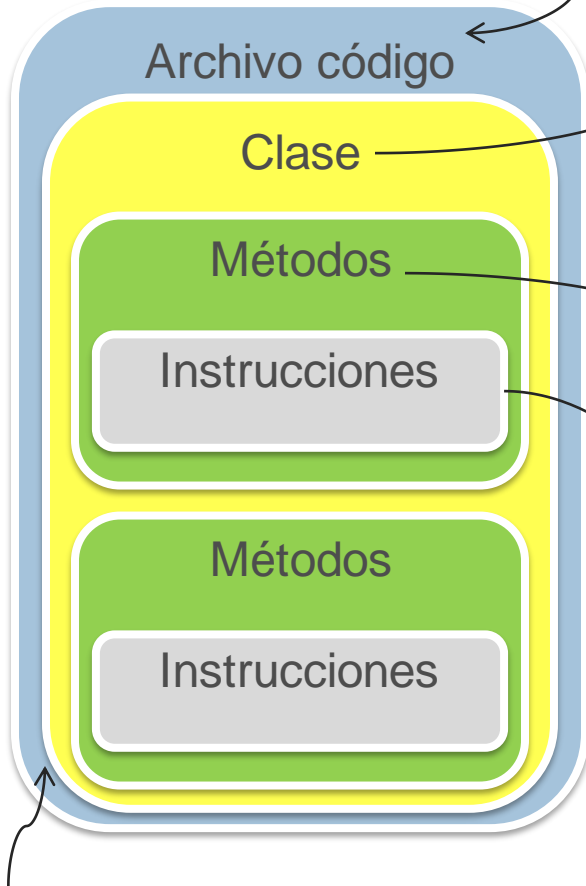
Agregar la ruta del JDK. En este caso  
C:\Archivos de programa\Java\jdk1.6.0\_10\bin;

Codifiquemos

## **2. APLICACIONES JAVA**

# Estructura del código en Java

El mismo nombre de la clase y terminado en .java



```
public class Perro  
{  
  
}
```

Sólo se pueden escribir dentro de una clase

```
public class Perro  
{  
    void ladra(){  
    }  
}
```

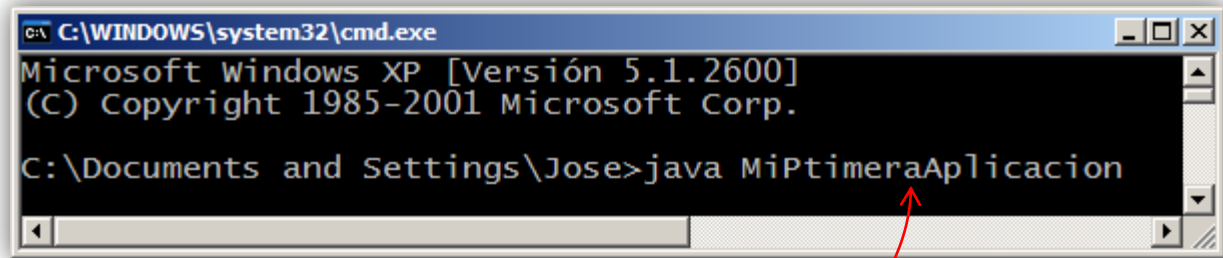
Sólo se puede escribir dentro de un método.

```
public class Perro  
{  
    void ladra(){  
        instruccion1;  
        instruccion2;  
    }  
}
```

El mismo archivo puede haber más de una clase pero solo una public

# Anatomía de una clase

Cuando se lanza una aplicación Java, la JVM busca la clase que se le pasa en la línea de comandos y, dentro de ella, el método **main** que tiene que tener los argumentos que se muestran en el ejemplo. A continuación, la JVM ejecuta el código que se encuentra dentro de **main**. Debe existir un **main** por aplicación, no por clase.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\Jose>java MiPrimeraAplicacion
```

Sólo puede haber una clase **public** por archivo

En Java las clases tienen ámbito. Ésta es **public** para que se pueda acceder y ejecutar **main**

Mismo nombre.  $A \neq a$

Esta es la forma de pasar argumentos a **main**

```
public class MiPrimeraAplicacion
{
    public static void main(String[] args){
        System.out.println("¡¡¡Controlo!!!");
    }
}
```

main debe tener esta forma

Imprime una línea en la consola

No es necesario el ;

# Escribiendo una clase con main

En Java todo va en una **class**. Escribimos nuestra aplicación en un archivo terminado en **.java**. Se compila a un archivo **.class** que es el que ejecuta la JVM.

```
public class MiPrimeraAplicacion
{
    public static void main(String[] args){
        System.out.println("¡¡¡Controlo!!!");
        System.out.println ("¡¡¡Java!!!");
    }
}
```

1

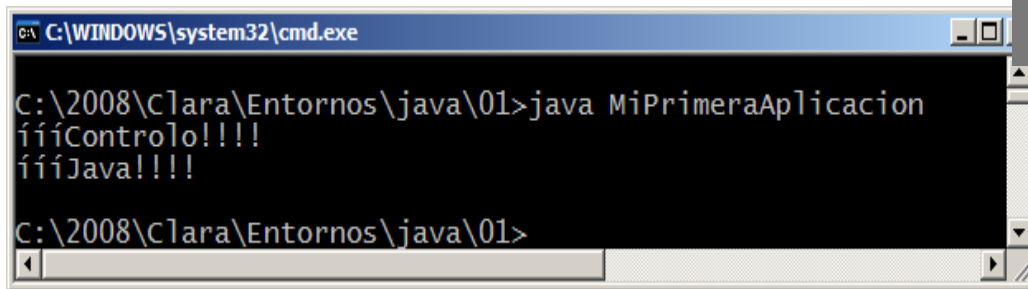
Guardar como MiPrimeraAplicacion.java

2



Compilar con  
`javac MiPrimeraAplicacion.java`

3



Ejecutar con  
`java MiPrimeraAplicacion`

# Estructura típica de un programa Java

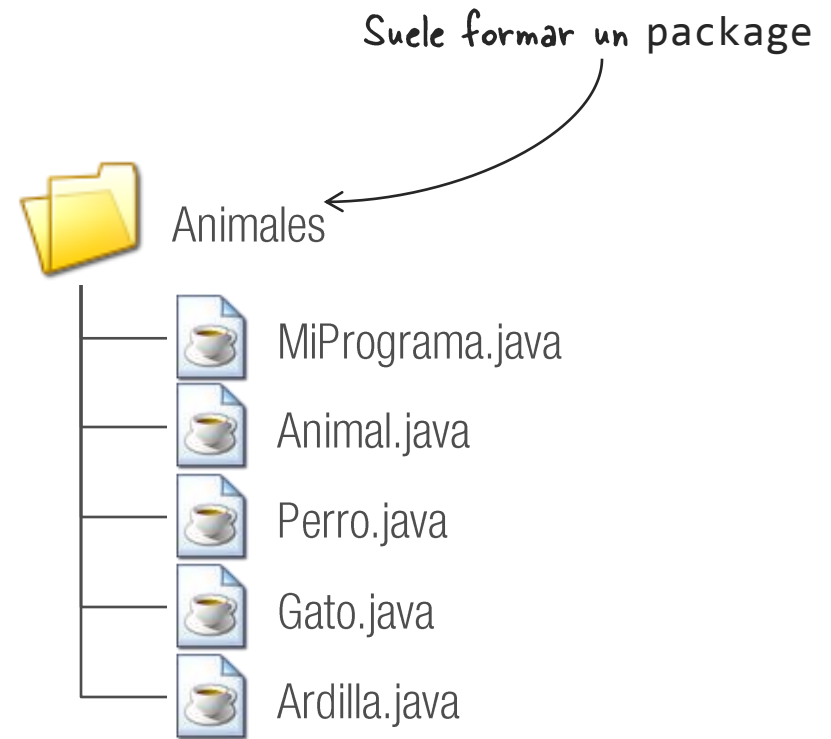
```
public class MiPrograma
{
    // definición de la clase
    private static void main...
}
```

```
class Animal
{
    // Definición de la clase
}
```

```
class Perro
{
    // Definición de la clase
}
```

```
class Gato
{
    // Definición de la clase
}
```

```
class Ardilla
{
    // Definición de la clase
}
```



# ¿A que es fácil?

Intenta adivinar que es lo que hace cada línea de código:

<pre>int tamaño = 22;</pre>	<i>Declarar un entero de nombre 'tamaño' y darle un valor de 22</i>
<pre>String nombre = "Ana";</pre>	
<pre>Perro miPerro = new Perro(nombre,tamaño);</pre>	
<pre>x = tamaño - 5;</pre>	
<pre>if (x &lt; 20) miPerro.ladra(8);</pre>	
<pre>while (x &gt; 3) {     miPerro.juega(); }</pre>	
<pre>int[] listaNumeros = {2,4,6,8};</pre>	
<pre>System.out.print("Hoola");</pre>	
<pre>System.out.print("Perro: " + nombre);</pre>	
<pre>String numero = "8";</pre>	
<pre>int z = Integer.parseInt(numero);</pre>	
<pre>try {     leeElArchivo("miArch.txt"); } catch (FileNotFoundException ex) {     System.out.print("No hay archivo"); }</pre>	

Comparemos

## **3. JAVA VS C++**



# Java vs C++

- **Java** utiliza Unicode como conjunto de caracteres para programar.
- En **Java** cada variable, constante y función (incluyendo main) debe estar dentro de una clase y cada clase parte de un package
- En **Java** sólo existen archivos con *fuentes* y archivos *bytecode*
- En **Java** hay 2 categorías de tipos: *tipos primitivos* y *tipos referencia*
- En **Java** no hay *struct*, *union* y *unsigned*.
- En **Java** las clases, cadenas y matrices (arrays) son punteros, tratados como referencias.
- **Java** cuenta con recolector de basura (liberación automática de memoria)
- En **Java** no se pueden sobrecargar operadores.
- En **Java** se pueden copiar variables de los tipos base usando la asignación (=) pero no variables de tipo objeto.

# Java vs C++

Tipos Primitivos	Nombre del tipo	Clase envolvente	Tamaño(bytes)	Valores por defecto
Enteros	byte	Byte	1	0
	short	Short	2	0
	int	Integer	4	0
	long	Long	8	0L ←
Reales	float	Float	4	0.0f ←
	double	Double	8	0.0d
Otros	char	Character	2	null
	boolean	Boolean	1	false

No existe la versión 'sin signo'

Se tiene que añadir una L o l al final del número porque sino es un int

Se tiene que añadir una F o f al final del número porque sino es un double

Los tipos por referencias son las String, los arrays y todos los objetos del sistema o creados por el programador. Se crean con la palabra new aunque pueden crearse implícitamente en el caso de String y arrays.

# Java vs C++

Java pasa los tipos siempre por valor, lo que significa que los pasa por copia



```
byte x = 7;
```



1

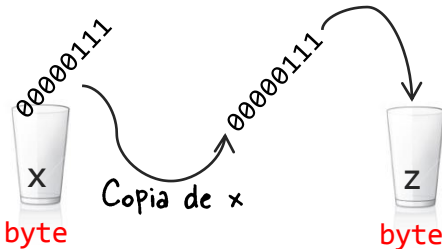
Declarar una variable byte y asignarle el valor de 7. El valor en binario se almacena en x.

```
void haz(byte z)  
{}
```



2

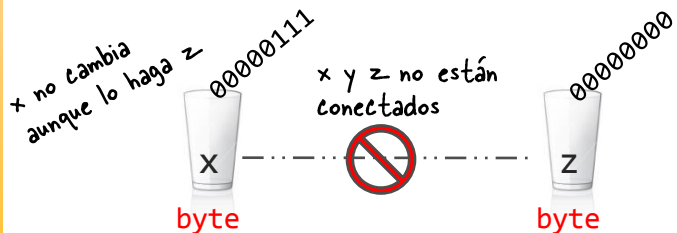
Declarar un método con un byte, z, como parámetro.



3

Llamada al método haz(), pasando x como argumento. Los bits de x se copian y la copia aparece en z.

```
obj.haz(x); void haz(byte z) {}
```



4

Cambia el valor de z dentro del método. El valor x no cambia. El argumento pasado a z era sólo una copia de x. El método no puede cambiar la variable x

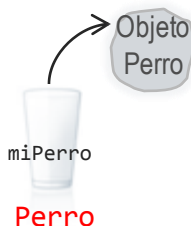
```
void haz(byte z) { z = 0;}
```

# Java vs C++

En el caso de tipos distintos a los primitivos, la copia es en realidad la dirección (referencia) al objeto.



```
Perro miPerro = new Perro();
```



1

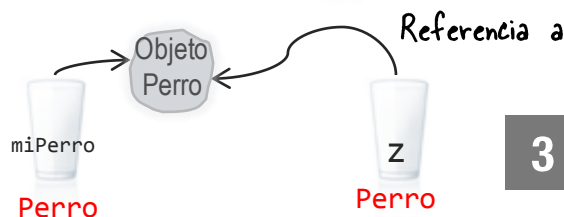
Declarar una variable `miPerro` y asignarle un objeto `Perro`. El valor que guarda `miPerro` es una referencia (puntero) a un objeto `Perro`.

```
void haz(Perro z) {}
```



2

Declarar un método con un `Perro`, `z`, como parámetro.

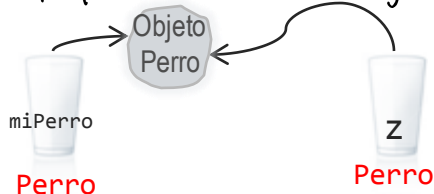


3

Llamada al método `haz()`, pasando `miPerro` como argumento. `z` recibe una copia de la referencia al objeto, esto es, una copia de `miPerro`.

```
obj.haz(miPerro); void haz(Perro z) {}
```

*miPerro y z están conectados porque referencian al mismo objeto*



4

Cambia el estado de `z` dentro del método lo que hace que cambie el objeto referenciado por `miPerro`

```
void haz(Perro z) { z.edad = 10;}
```

# Java vs C++

Java cuenta con 3 tipos de comentarios.

```
/*  
  Comentario 'bloque' (líneas múltiples) 1  
*/  
  
/**  
  Comentario de documentación utilizado para generar  
  documentación automática del programa (pe, con javadoc) 2  
  Empieza con /** y nos encontraremos variables del tipo  
  @version 1.0  
*/  
  
public class MiPrimeraAplicacion  
{  
    public static void main(String[] args){  
        System.out.println("¡¡¡Controlo!!!"); // Comentarios de 'línea' 3  
        System.out.println ("¡¡¡Java!!!");  
    }  
}
```

# Java vs C++

Etiquetas más comunes para la generación de documentos.

Etiquetas para referencias:

`@see <otra clase>`

Etiquetas de documentación de clases:

`@version <información sobre la versión>`

`@author <nombre del autor>`

Etiquetas de documentación de métodos:

`@param <nombre del argumento> <descripción>`

`@return <descripción>`

`@exception <excepción>`

# Java vs C++

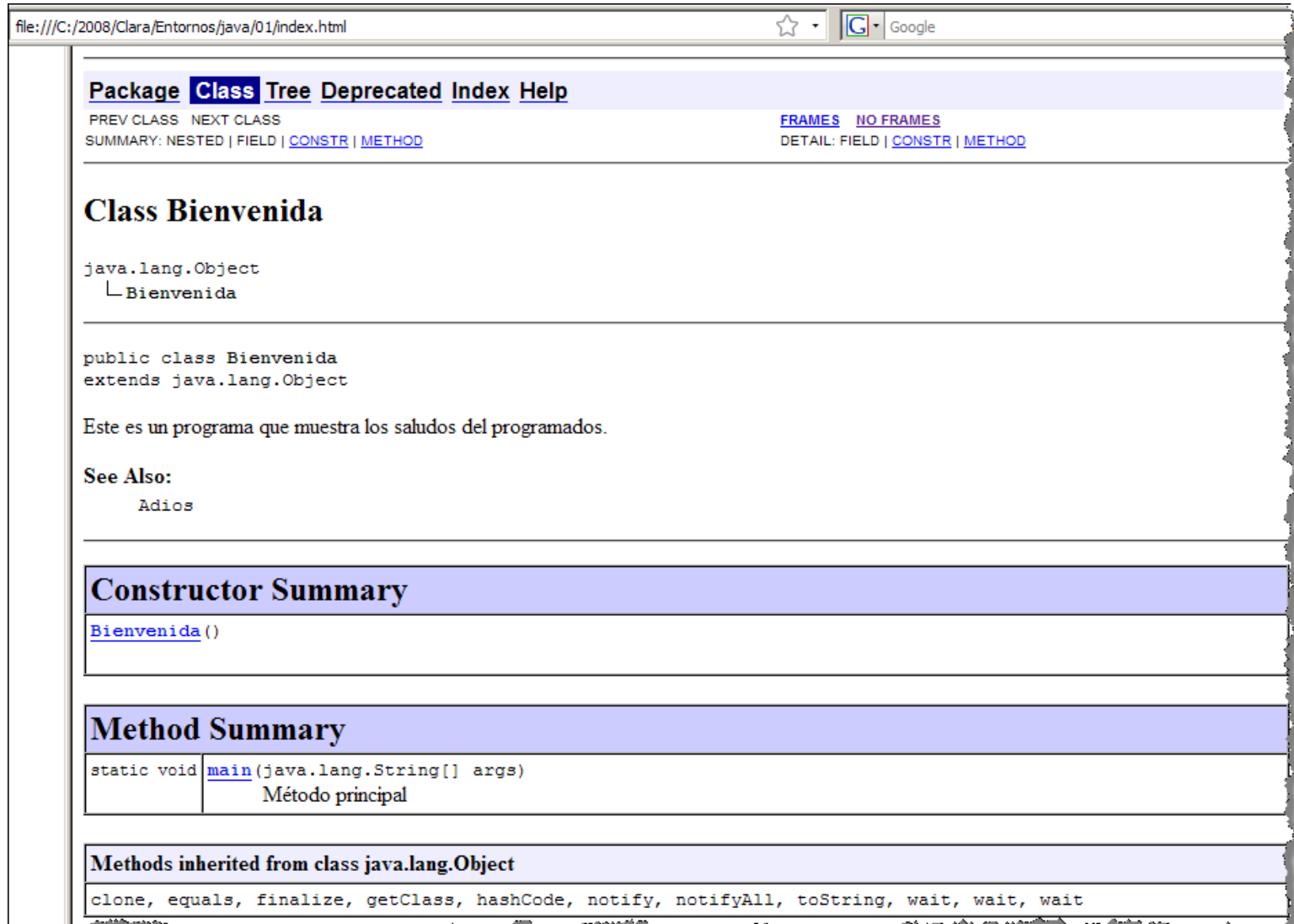
Uso típico para generación de documentos.

```
/**
 * Este es un programa que muestra los saludos del programados.
 * @version 1.00 2008-11-27
 * @author Yo Mismo
 * @see Adios
 */
public class Bienvenida
{
    /** Método principal
     * @param args argumentos en línea de comandos
     * @return Nada
     */
    public static void main(String[] args)
    {
        String[] saludos = new String[3];
        saludos[0] = "Bienvenidos al mundo Java";
        saludos[1] = "por Yo Mismo"; // Puedes cambiar a tu nombre
        saludos[2] = "y Sun Inc.";

        for (String s : saludos)
            System.out.println(s);
    }
}
```

# Java vs C++

javadoc Bienvenida.java genera una serie de archivos .html. Index.html es el principal:



The screenshot shows a web browser window with the address bar containing the file path: file:///C:/2008/Clara/Entornos/java/01/index.html. The browser's search bar contains the text "Google". The page content is as follows:

**Package** **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS [FRAMES](#) [NO FRAMES](#)  
SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)      [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

---

## Class Bienvenida

java.lang.Object  
└ Bienvenida

---

```
public class Bienvenida
extends java.lang.Object
```

Este es un programa que muestra los saludos del programados.

**See Also:**  
Adios

---

### Constructor Summary

[Bienvenida\(\)](#)

---

### Method Summary

static void	<a href="#">main</a> (java.lang.String[] args) Método principal
-------------	--

---

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



# Java vs C++

Con estos operadores no cuenta Java.

->  
.\*  
->\*  
\* unario  
& unario  
sizeof  
, Coma (como operador). Si soportada en los for  
delete

Con estos operadores no cuenta C/C++.

>>> Desplazamiento a la derecha con rellenos de 0  
>>>= Indica si un objeto es una instancia de una clase o de una de sus subclases  
instanceof

# Java vs C++

## Jerarquía de operadores en Java

TIPO	OPERADOR
Operadores sufijos	[] . (argumentos) expr++ expr--
Operadores unarios	++expr --expr +expr -expr ~ !
Creación y <i>casting</i>	new (tipo) expr
Multiplicativos	* / %
Aditivos	+ -
Desplazamiento	<< >> >>>
Relacional	< > <= >= instanceof
Igualdad	== !=
AND	(bits) &
OR exclusivo	(bits) ^
OR inclusivo	(bits)
AND lógico	&&
OR lógico	
Condicional	? :
Asignación	= += -= *= /= %= &= ^=  = <<= >>= >>>=

Todos los operadores binarios se evalúan de izquierda a derecha, excepto los operadores de asignación.

# Java vs C++

Salvo pequeñas diferencias de matiz, las siguientes instrucciones funcionan igual que en C/C++:

```
for() {}  
while () {}  
do {} while();  
if(){}else{}  
:?  
if() {} else if {} ... else{}  
swicth() {case: break; ... default: break;  
continue  
break  
return
```

## *for each*

Permite movernos por los elementos de una array y colecciones de elementos sin tener que usar un índice.

```
for( variable : colección) {}
```

Variable debe ser del mismo tipo de los elementos de la colección

```
int[] a = {1, 2, 3,};  
for (int e : a)  
    System.out.print(e);
```

Es único

# 4. JAVA

# Variables

## Variables de instancia

Variables declaradas sin el modificador `static`. Tienen un valor para cada objeto instanciado.

## Variables estáticas

Variables declaradas con el modificador `static`. Sólo existe una copia con independencia del número de objetos que existan de la clase. Las comparten todos los objetos y no hace falta instanciar un objeto para acceder a su valor.

## Variables locales

Variables declaradas dentro de los métodos. Sólo son accesibles dentro de los métodos donde se declaran y esconden, para ese métodos, cualquier variable externa declarada con el mismo nombre.

## Parámetros

Variables utilizadas para pasar datos a los métodos. Los parámetros con tipos básicos se pasan por copia; el resto, por referencia incluyendo los arrays.

# Palabras

Palabras claves reservadas en Java.

abstract	default	goto <sup>1</sup>	null <sup>2</sup>	synchronized
boolean	do	if	package	this
break	double	implements	private	throw
byte	else	import	protected	throws
case	extends	instanceof	public	transient
catch	false <sup>2</sup>	int	return	true <sup>2</sup>
char	final <sup>3</sup>	interface	short	try
class	finally	long	static	void
const	float	native	super	volatile
continue	for	new	switch	while

<sup>1</sup> Palabras no usadas por el lenguajes pero reservadas

<sup>2</sup> En realidad son constantes del lenguaje

<sup>3</sup> Java utiliza el modificador final para las constantes.

# E/S consola

En **Java** la entrada desde teclado y la salida por pantalla se realizan a través de la clase **System**.

**System** contiene tres atributos estáticos:

**System.in**: Objeto de la clase **InputStream** que lee datos de la entrada estándar (teclado).

**System.out**: Objeto de la clase **PrintStream** que escribe datos en la salida estándar (pantalla).

**System.err**: Objeto de la clase **PrintStream** que escribe mensajes de error en la salida estándar (pantalla).

# E/S consola

En **Java** la salida por pantalla se realiza a través de la clase de los métodos:

`System.out.print(<argumento>)`: Imprime por pantalla el argumento dado, independientemente del tipo que sea.

`System.out.println(<argumento>)`: Igual que el anterior, pero añadiendo un salto de línea.

Ambos métodos pueden imprimir:

Valores directamente

```
System.out.println("Hola Mundo");  
double numeroPI = 3.141592654;  
System.out.println(numeroPI);
```

Valores concatenados con el operador +

```
System.out.println("Hola Mundo" + numeroPI);
```

**Java** también cuenta con `System.out.printf(<cadena formato><especificadores>)` que funciona prácticamente como en C.



# E/S consola

En **Java** la entrada por teclado se realiza a través de la clase del método `readLine()` que permite leer una **String**.

Para usar este método, se debe crear un buffer para el flujo (stream) de caracteres de entrada. Para ello realizamos las siguientes declaraciones de variables:

```
InputStreamReader lector = new InputStreamReader(System.in);  
BufferedReader bufer = new BufferedReader(lector);
```

Una vez realizadas estas declaraciones, la entrada de datos la podemos realizar de la siguiente manera:

```
String entrada = bufer.readLine();
```

# E/S consola

Un programa **Java** que use la entrada por teclado, debe incluir la sentencia:

```
import java.io.*;
```

También el método en donde se incluya la sentencia `readLine()`, deberá incluir a continuación de su cabecera la expresión:

```
throws java.io.IOException
```

Por ejemplo:

```
public static void main(String[] args)
throws java.io.IOException    {
    ...
}
```

Si la entrada es un número, deberemos convertirla (normalmente con una clase *wrapper*) :

```
int x = Integer.parseInt(entrada);
```

# E/S consola

En **Java** la entrada por teclado también se puede realizar mediante **Scanners** que está en `import java.util.*;`

Para usar este método, se debe crear un escáner y leer tokens con él.

```
Scanner lector = Scanner.create(System.in);
```

Para leer los tokens, utilizamos un método `nextTipo`, con Tipo cualquiera de los tipos primitivos mas `String`, `BigInteger` y `BigDecimal`, `nextLine` y `next`:

```
String nombre = lector.nextString();
```

```
int edad = lector.nextInt();
```

Lee hasta un `rt/lf`      Hasta un espacio

También contamos con los métodos de la forma `hasNextTipo`, que devuelve un valor `bool` si el siguiente token es del tipo testado.

# Arrays

Se crean con new

En **Java** los arrays (matrices) son **objetos**. Cada elemento de un array es una variable. Esto es, uno de los 8 tipos primitivos o una referencia. Cualquier cosa que se pueda poner en una variable, se puede asignar a un elemento del array. Todos los elementos del array deben ser del mismo tipo.

Declaración de arrays

```
<tipo> variable[];  
<tipo>[] variable;
```

Los [] se usan para declarar el array y para acceder a los elementos. En la declaración pueden ir detrás del tipo o de la variable

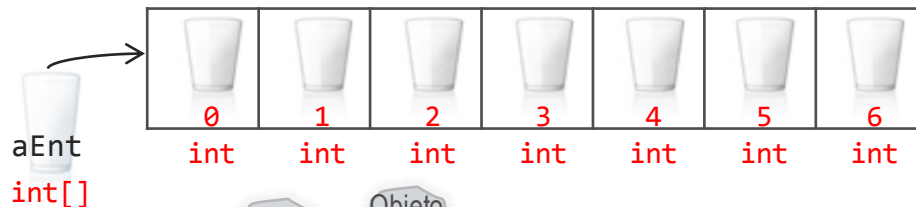
```
int[] aEnt; aEnt  
int[]
```

Creación del objeto array

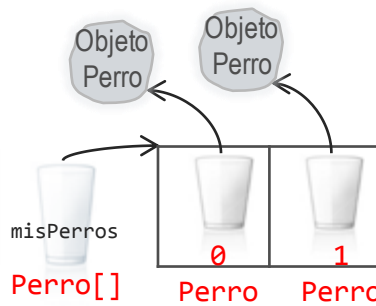
```
variable = new <tipo>[tamaño]
```

Se debe especificar el tamaño (hay excepciones)

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



```
Perro[] misPerros;  
misPerros = new Perro[2];
```



```
misPerros[0] = new Perro;  
misPerros[1] = new Perro;
```

# Arrays

```
int[] array = {1, 2, 3, 4, 5};
```

Otra forma de inicializar un array

## length

Atributo que devuelve el tamaño del array.

```
int[] aEnt = new int[8];  
System.out.print(aEnt.length);
```

Imprime 8

## System.arraycopy(org, orgInd, des, desInd, num)

Copia una array a otro. El array de destino debe tener suficiente espacio para contener el de origen

```
int[] org = {1,2,3,4,5};  
int[] des = new int[4];  
System.arraycopy(org,0,des,0,5);
```

## Arrays.toString(array)

Método que devuelve una cadena con los elementos de un array.

```
int[] aEnt = {1,2,3,4,5};  
System.out.print(Arrays.toString(aEnt));
```

Imprime [1 2 3 4 5]

## Arrays.sort(array)

Ordena un array numérico.

```
int[] aEnt = {2,5,3,1,4};  
Arrays.sort(aEnt);
```

## Arrays.equals(a, b)

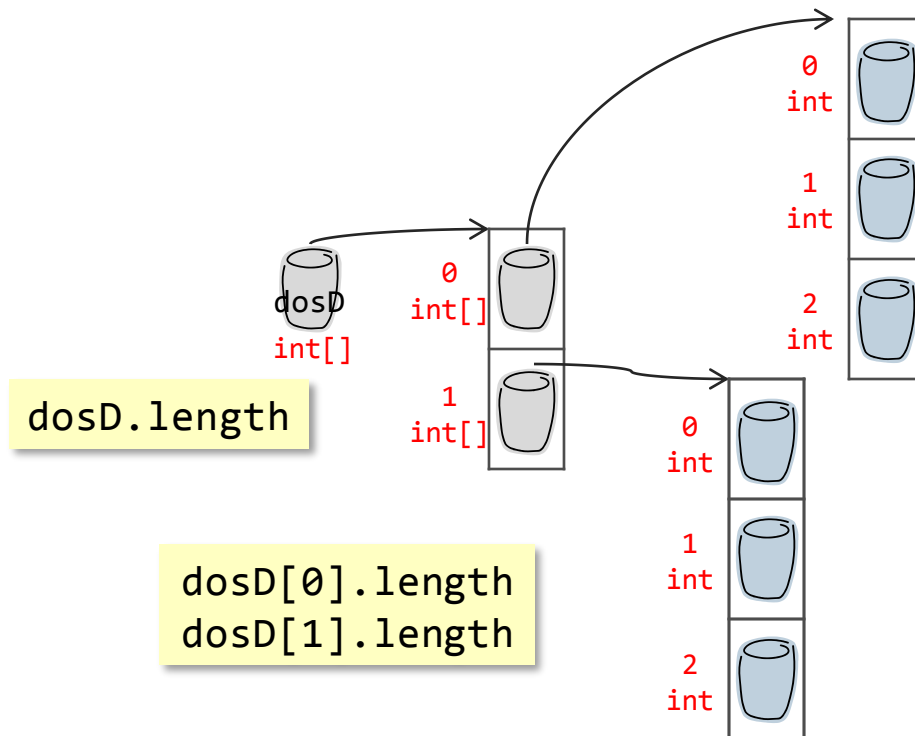
Compara dos arrays de tipos básicos y devuelve un boolean. Compara el tamaño y el contenido de los arrays.

```
int[] a = {2,5,3,1,4};  
int[] b = {2,5,3,1,4};  
Arrays.equals(a, b);
```

# Arrays

En **Java** sólo existen arrays de una dimensión. Para crear arrays multidimensionales hay que crear arrays de arrays. Se necesita un grupo de corchetes por cada dimensión.

```
int[] unaD = new int[2];  
int[][] dosD = new int[2][3];  
int[][][] tresD = new int[2][3][4];  
...
```



# Java vs C++

Si la salida de java DooBee es DooBeeDooBeeDo  
Rellena el código que falta:

```
public class DooBee {
    public static void main(String[] args)
    {
        int x = 1;
        while ( x < _____ ) {
            System.out._____( "Doo" );
            System.out._____( "Bee" );
            x++;
        }
        if ( x == _____ ) {
            System.out._____( "Do" );
        }
    }
}
```

# Java vs C++

Si la salida de java Mezcla es a-b c-d

Ordena los trozos (los trozos más pequeños los puedes añadir tu):

```
if ( x == 1 ) {  
    System.out.print("d");  
    x--;  
}
```

```
while ( x > 0 ) {
```

```
x = 3;
```

```
if ( x == 2 ) {  
    System.out.print("b c");  
}
```

```
x--;  
System.out.print("-");
```

```
if ( x > 2 ) {  
    System.out.print("a");  
}
```

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



# Java vs C++

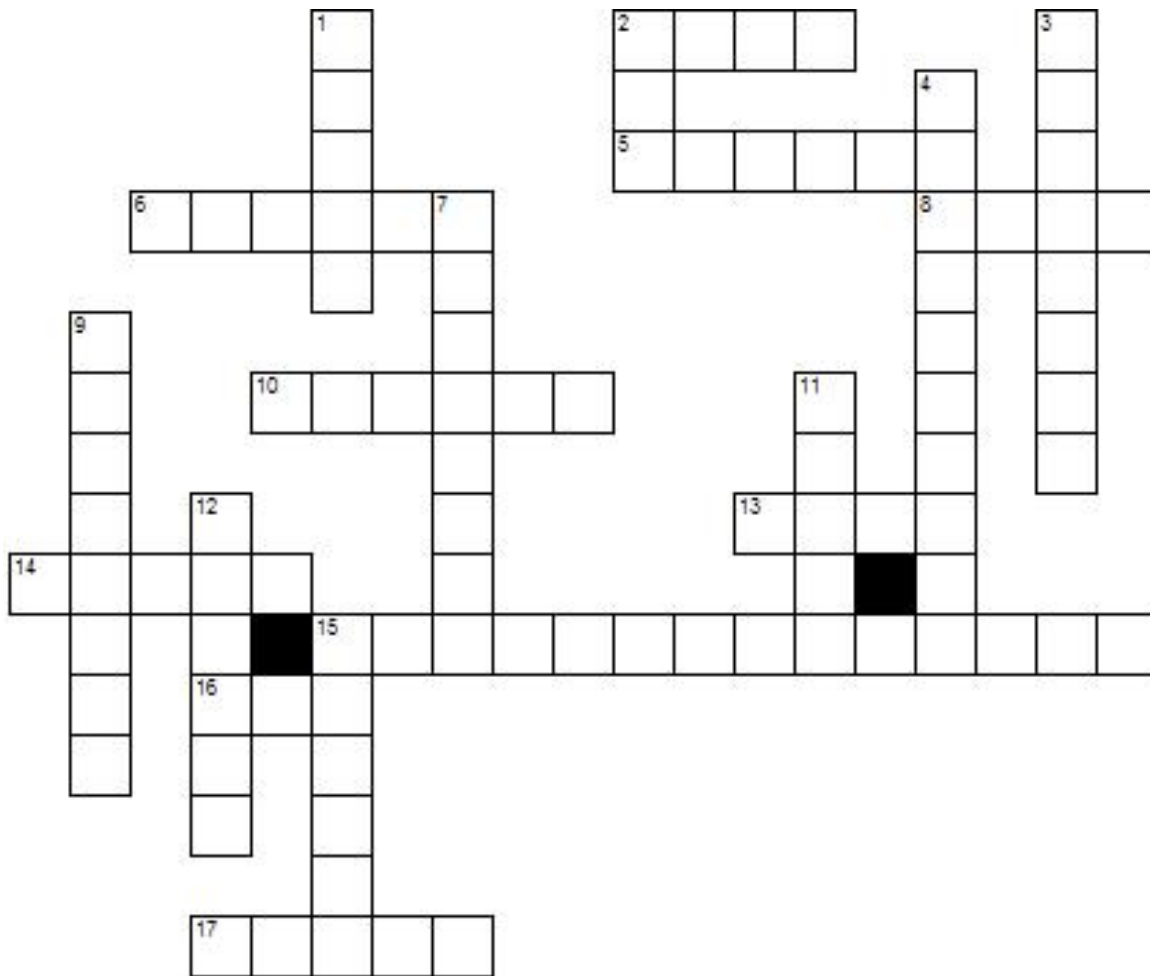
Sé un compilador, my friend

```
class EjercicioA {  
    public static void main(String[] args) {  
        int x = 1;  
        while ( x < 10 ) {  
            if ( x > 3 ) {  
                System.out.print("Una X");  
            }  
        }  
    }  
}
```

```
public static void main(String[] args) {  
    int x = 5;  
    while ( x > 1 ) {  
        x--;  
        if ( x < 3 ) {  
            System.out.print("Otra X");  
        }  
    }  
}
```

```
class EjercicioC {  
    int x = 5;  
    while ( x > 1 ) {  
        x = x - 1;  
        if ( x < 3 ) {  
            System.out.print("Pues X");  
        }  
    }  
}
```

# Java vs C++



## Horizontales

2. Invocador en línea de comando.
5. La forma de conseguir cosas
6. Casa de todos.
8. Sólo tengo uno.
10. Contenedores de cosas
13. Venir con las manos vacías.
14. No puedo hacer las dos cosas a la vez.
15. Decir algo.
16. Tipo de variable numérica.
17. ¿Atrás de nuevo?.

## Verticales

1. Hasta que mejore la aptitud
2. Consumidor de bytecode
3. No puedo dejar de nombrarla
4. Consumidor de código
7. ¿Para qué son buenos los prompt?.
9. Anunciar un nuevo método o clase.
11. No es un entero (pero \_\_\_\_a).
12. Un montón de caracteres.
15. ¡Quédate quieto!

# Java vs C++

Asociar el código hay que poner en el bloque con su salida

```
public class Prueba {  
    public static void main(String[] args) {  
        int x = 0;  
        int y = 0;  
        while ( x < 5) {  
              
            System.out.print(x + " " + y + " ");  
            x--;  
        }  
    }  
}
```

```
y = x - y;
```

```
y = y + x;
```

```
y = y + 1;  
if ( y > 4 ) {  
    y = y - 1;  
}
```

```
x = x + 1;  
y = y + x;
```

```
if ( y > 5 ) {  
    x = x + 1;  
    if ( y < 3 ) {  
        x = x - 1;  
    }  
}  
y = y + 2;
```

```
22 46
```

```
11 34 59
```

```
02 14 26 38
```

```
02 14 36 48
```

```
00 11 21 32 42
```

```
11 21 32 42 53
```

```
00 11 23 36 410
```

```
02 14 25 36 47
```