JAVA

Curso Express

¿Qué es JAVA?

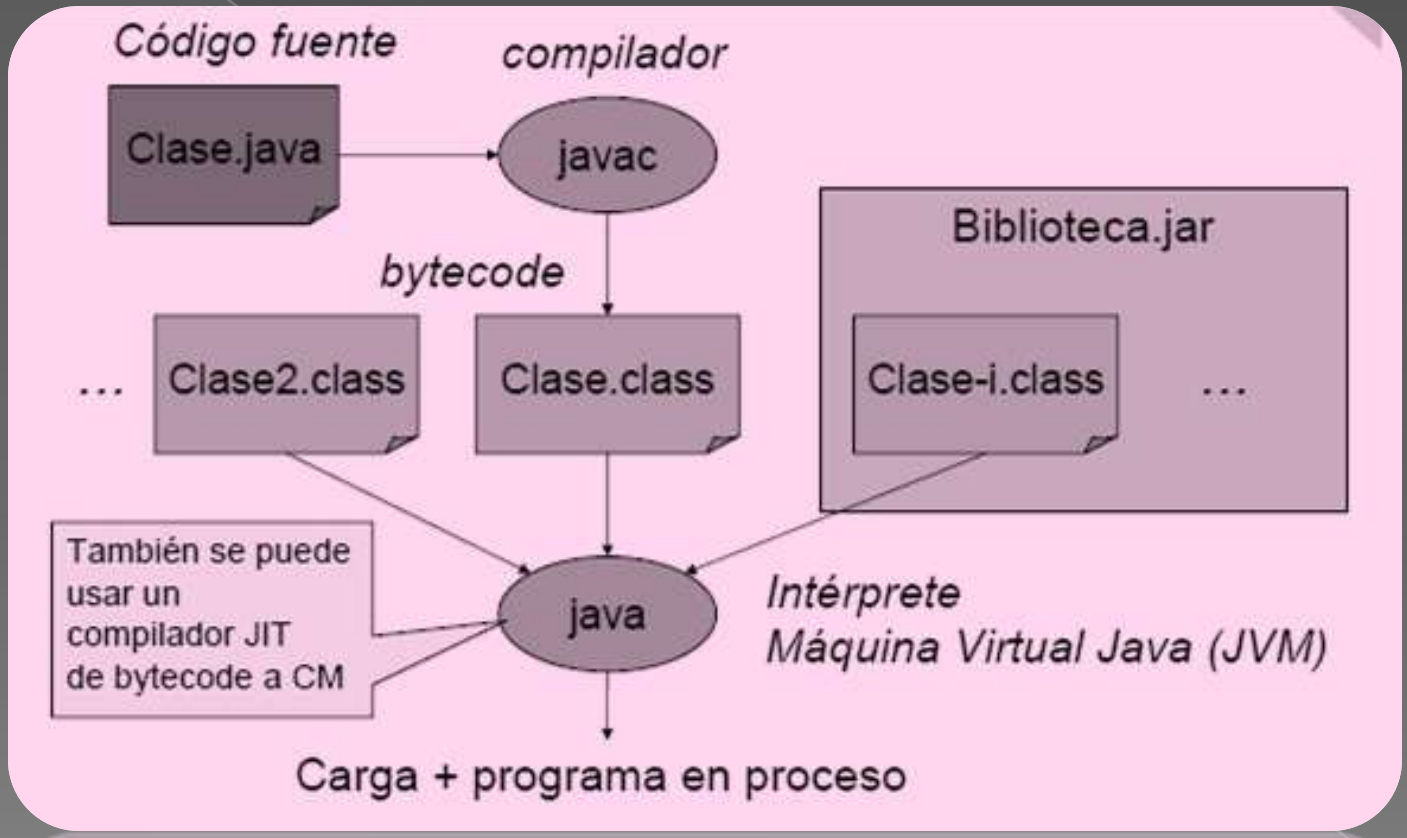
- Java es un lenguaje de programación con el que podemos realizar cualquier tipo de programa. Fue diseñado para ser multiplataforma y poder ser empleado el mismo programa en diversos sistemas operativos.

- Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

¿Cómo funciona JAVA?

Las aplicaciones Java están típicamente compiladas en un ***bytecode***, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el ***bytecode*** es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del ***bytecode*** por un procesador Java también es posible.

¿Cómo funciona JAVA?



JRE y JDK

- Java se distribuye en 3 formatos:
  - JRE (Java Runtime Environment) solo para ejecutar programas Java (tiene las clases Core).
  - JDK (Java Development Kit) contiene **además** compiladores necesarios.
  - Plug-in para instalar en los clientes web.
- Además es interesantes bajarse la documentación en formato JavaDoc.
- La dirección es http://java.sun.com
  - También IBM tiene su sistema http://oss.software.ibm.com/developerworks/opensource/jikes/

Programación en JAVA

Cuando se programa en Java, se coloca todo el código en **métodos**, de la misma forma que se escriben funciones en lenguajes como C.

```
//clase Circulo
public class Circulo
 {
  //metodos
  public double perimetro();
  public double area();
  }
```

Comentarios

En Java hay tres tipos de comentarios:

```
// comentarios para una sola línea

/* comentarios de una o
   más líneas
*/

/** comentario de documentación, de una o más líneas
*/
```

Identificadores

Los identificadores nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar.

En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.

Ejemplos de identificadores válidos:

perimetro, area, nom_circulo, $volumen, etc…

Ejemplo de identificadores inválidos:

5area, #volumen, +variable, además de las palabras clave.

Palabras Clave

Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como indentificadores:

class	finally	long	const	implements	static
abstract	default	else	final	instanceof	package
continue	break	case	float	interface	return
for	do	extends	native	private	short
new	byte	catch	goto	import	throw
switch	double	false	null	public	true
char	byvalue	int	this	void	while
transient	synchronized	protected	super	threadsafe	If

Palabras Reservadas

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico. Son:

cast	future	generic	inner
operator	outer	rest	var

Literales

Un valor constante en Java se crea utilizando una representación literal de él. Java utiliza cinco tipos de elementos: enteros, reales en coma flotante, booleanos, caracteres y cadenas, que se pueden poner en cualquier lugar del código fuente de Java. Cada uno de estos literales tiene un tipo correspondiente asociado con él.

- **Enteros:** byte, short, int y long (de 8,16,32 y 64 bytes respectivamente)
- **Reales en coma flotante**: float y double (de 32 y 64 bytes respetivamente)
- **Booleanos:** true y false.
- **Caracteres:** a , \t, \u???? [????] es un número unicode.
- **Cadenas:** "Esta es una cadena literal".

Arrays

Se pueden declarar en Java arrays de cualquier tipo:

```
char s[];
int iArray[];
```

Incluso se pueden construir arrays de arrays:

```
int tabla[][] = new int[4][5];
```

Los límites de los arrays se comprueban en tiempo de ejecución para evitar desbordamientos y la corrupción de memoria.

En Java un array es realmente un objeto, porque tiene redefinido el operador []. Tiene una función miembro: length. Se puede utilizar este método para conocer la longitud de cualquier array.

```
int a[][] = new int[10][3];
a.length;        /* 10 */
a[0].length;     /*  3 */
```

Operadores

Operadores Aritméticos	
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto

Operadores

Operadores Relacionales	
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
!=	Distinto
==	Igual

Operadores

Operadores Relacionales	
&&	And
||	Or
!	Not

Operadores

Operadores Bit a Bit	
&	And
|	Or
^	Xor
<<	*Propagación a la izquierda:* desplaza el valor hacia la izquierda introduciendo ceros, si se sale de rango se pierden valores.
>>	*Propagación hacia la derecha:* desplaza el valor en binario hacia la derecha introduciendo por la izquierda el bit de signo y eliminado los valores que se salgan por la derecha.
>>>	*Zero fill, propagación a la derecha:* Similar a << pero hacia la derecha. No introduce el bit de signo.

Operadores

Operadores de asignación		
Operador	**Expresión**	**Equivalencia**
=	A=B=C	A=C; B=C;
+=	A+=4	A=A+4
-=	A-=3*B	A=A-(3*B)
*=	A*=2	A=A*2
/=	A/=35+B	A=A/(35+B)
%=	A%=8	A=A%8
>>=	A>>=1	A=A>>1
<<=	A<<=B	A=A<<B
>>>=	A>>>=B+1	A=A>>>(B+1)
&=	A&=(C+=3)	C=C+3;A=A&C;
^=	A^=2	A=A^2
|=	A|=C	A=A|C

Operadores

Operador de selección.

Este operador se utiliza para ejecutar una operación u otra dependiendo de la condición. El formato es el siguiente:

Condición ? Exp1 : Exp2

Si se cumple la condición se evalúa y se devuelve la expresión Exp1 si no la Exp2. *Podemos poner un sólo valor.* Ejemplo:

i = (x!=y)?6:(k+1)

Operador *new*.

Este operador se va a utilizar para crear una instacia de un tipo de objetos previamente definido. La sintaxis a seguir es la siguiente:

variableObjeto = *new* tipoDeObjeto(parámetro 1, parámetro 2, ...)

Estos parámetros son los que se le pasan al constructor de dicho objeto en cuestión.

Operador *typeof*.

Este operador aplicado a una variable devuelve el tipo de objeto al que pertenece el dato contenido por dicha variable. Su sintaxis es:

***typeof*(variable)**

Separadores

() - paréntesis. Para contener listas de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.

{} - llaves. Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.

[ ] - corchetes. Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.

; - punto y coma. Separa sentencias.

, - coma. Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.

. - punto. Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

Excepciones

```
try {
        sentencias;
} catch( Exception ) {
        sentencias;
}
```

Java implementa excepciones para facilitar la construcción de código robusto. Cuando ocurre un error en un programa, el código que encuentra el error lanza una excepción, que se puede capturar y recuperarse de ella. Java proporciona muchas excepciones predefinidas.

Excepciones

- Las excepciones hay que atraparlas para que sirva de algo.
  - Si no se atrapan se elevan (raise) al nivel superior.
- Tras atraparlas, podemos recuperar el error, o lanzarlas hacia arriba bajo otra forma más abstracta

```
try {
    FileInputStream in = new File .... ("archivo") ;
    in.read(buf, offset, size);
   // código a tutiplen
} catch (final FileNotFoundException ex) {
    ex.printStackTrace(); // y más cosas
} catch (final IOException ex) { ...
} finally {
    if (in!= null) in.close()
}
```

Bloque a comprobar

Bloque a ejecutar

Siempre se ejecutará

Clases

Las clases son lo más simple de Java. Todo en Java forma parte de una clase, es una clase o describe como funciona una clase.

Todos los datos básicos, como los enteros, se deben declarar en las clases antes de hacer uso de ellos.

La palabra clave import puede colocarse al principio de un fichero, fuera del bloque de la clase.

Tipos de clases

abstract

Una clase abstract tiene al menos un método abstracto. Una clase abstracta no se instancia, sino que se utiliza como clase base para la herencia.

final

Una clase final se declara como la clase que termina una cadena de herencia. No se puede heredar de una clase final. Por ejemplo, la clase Math es una clase final.

public

Las clases public son accesibles desde otras clases, bien sea directamente o por herencia. Son accesibles dentro del mismo paquete en el que se han declarado. Para acceder desde otros paquetes, primero tienen que ser importadas.

synchronizable

Este modificador especifica que todos los métodos definidos en la clase son sincronizados, es decir, que no se puede acceder al mismo tiempo a ellos desde distintos threads; el sistema se encarga de colocar los flags necesarios para evitarlo. Este mecanismo hace que desde threads diferentes se puedan modificar las mismas variables sin que haya problemas de que se sobreescriban.

Control de Acceso

Cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos definidos en la clase:

public

```
public void CualquieraPuedeAcceder(){}
```

protected

```
protected void SoloSubClases(){}
```

private

```
private String NumeroDelCarnetDeIdentidad;
```

friendly (sin declaración específica)

```
void MetodoDeMiPaquete(){}
```

Por defecto, si no se especifica el control de acceso, las variables y métodos de instancia se declaran friendly (amigas), lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete. Es lo mismo que protected.

Variables y Métodos Estáticos

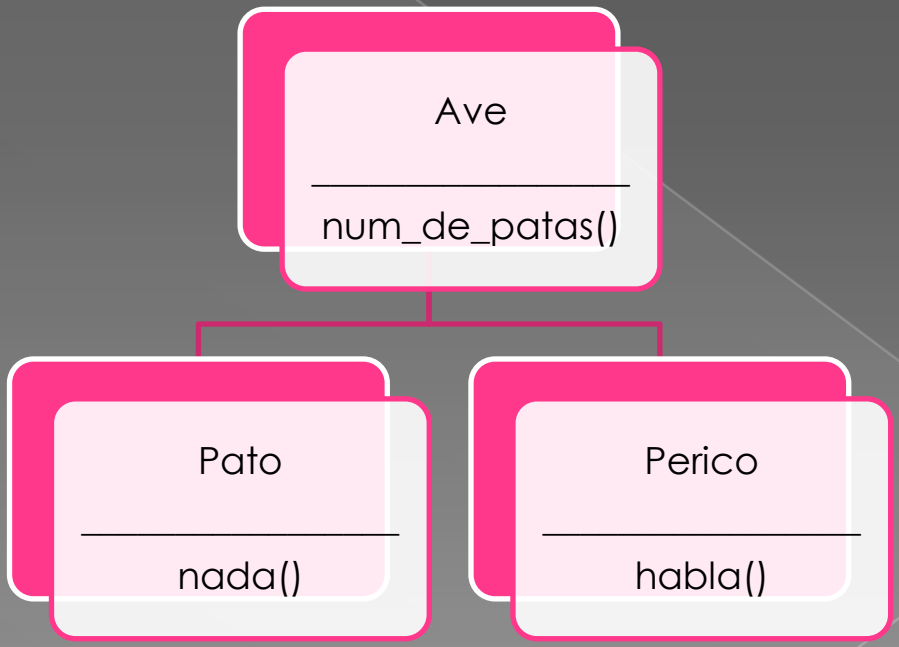
Todas las clases que se derivan, cuando se declaran estáticas, comparten la misma página de variables; es decir, todos los objetos que se generen comparten la misma zona de memoria. Las funciones estáticas se usan para acceder solamente a variables estáticas.

```
class Documento extends Pagina {
        static int version = 10;
         int numero_de_capitulos;
        static void annade_un_capitulo() {
                        numero_de_capitulos++;        // esto no funciona
         }
        static void modifica_version( int i ) {
                        version++;                    // esto si funciona
        }
 }
```

Herencia

La Herencia es el mecanismo por el que se crean nuevos objetos definidos en términos de objetos ya existentes.

La palabra clave **extends** se usa para generar una subclase (especialización) de un objeto.



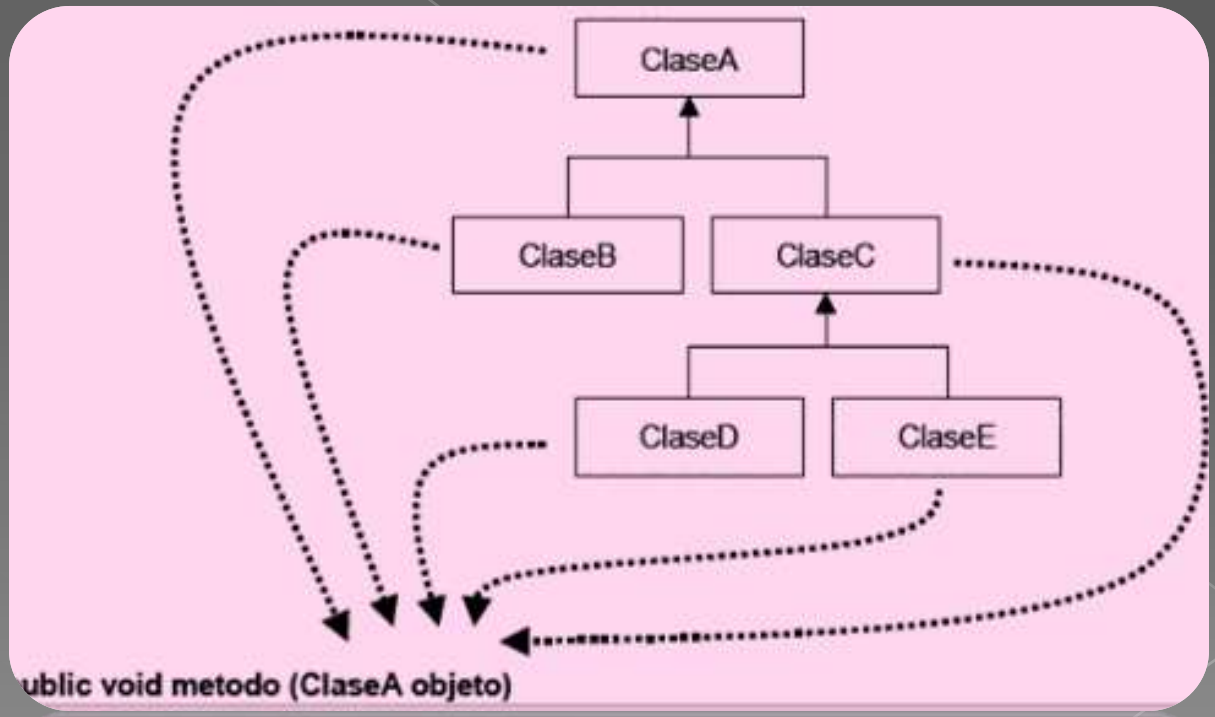
This

Al acceder a variables de instancia de una clase, la palabra clave this hace referencia a los miembros de la propia clase.

```
public class MiClase {
        int i;
        public MiClase() {
                i = 10;
        }
        // Este constructor establece el valor de i
        public MiClase( int valor ) {
                this.i = valor;  // i = valor
        }
        public void Suma_a_i( int j ) {
                i = i + j;
        }
 }
```

Polimorfismo

Es la capacidad que tienen los objetos para comportarse de múltiples formas.



Super

Si se necesita llamar al método padre dentro de una clase que ha reemplazado ese método, se puede hacer referencia al método padre con la palabra clave super.

```
import MiClase;
public class MiNuevaClase extends MiClase {
  public void Suma_a_i( int j ) {
      i = i + ( j/2 );
      super.Suma_a_i( j );
      }
  }
```

Clases Abstractas

Una de las características más útiles de cualquier lenguaje orientado a objetos es la posibilidad de declarar clases que definen como se utiliza solamente, sin tener que implementar métodos. Esto es muy útil cuando la implementación es específica para cada usuario, pero todos los usuarios tienen que utilizar los mismos métodos.

Cuando una clase contiene un método abstracto tiene que declararse abstracta. No obstante, no todos los métodos de una clase abstracta tienen que ser abstractos. Las clases abstractas no pueden tener métodos privados (no se podrían implementar) ni tampoco estáticos. Una clase abstracta tiene que derivarse obligatoriamente, no se puede hacer un new de una clase abstracta.

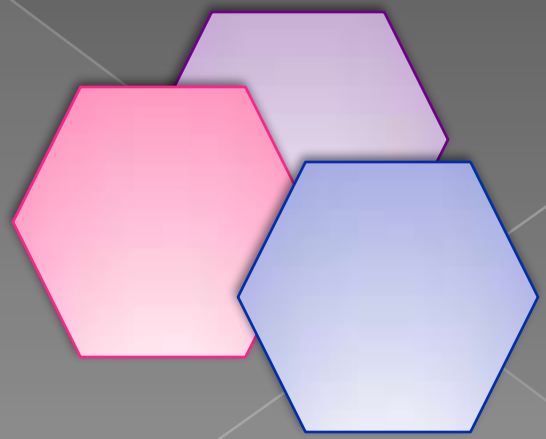
Ejemplo de Clase Abstracta

Un ejemplo de clase abstracta en Java es la clase Graphics:

```
public abstract class Graphics {
public abstract void drawLine( int x1,int y1,int x2,int y2 );
public abstract void drawOval( int x,int y,int width,int height );
public abstract void drawArc( int x,int y,int width,int height,int startAngle,int arcAngle );
    . . .
    }
```

Los métodos se declaran en la clase Graphics, pero el código que ejecutará el método está en algún otro sitio:

```
public class MiClase extends Graphics {
    public void drawLine( int x1,int y1,int x2,int y2 ) {
        <código para pintar líneas -específico de
        la arquitectura->
        }
    }
```



Interfaces

Los interfaces proporcionan un mecanismo para abstraer los métodos a un nivel superior.

Un interface contiene una colección de métodos que se implementan en otro lugar. Los métodos de una clase son public, static y final.

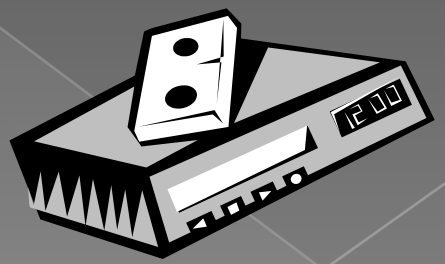
La principal diferencia entre interface y abstract es que un interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia.

Ejemplo de Interfaces

```
public interface VideoClip {
        void play(); // comienza la reproduccion del video
       void bucle(); // reproduce el clip en un bucle
        void stop(); // detiene la reproduccion
        }
```

Las clases que quieran utilizar el interface VideoClip utilizarán la palabra **implements** y proporcionarán el código necesario para implementar los métodos que se han definido para el interface:

```
class MiClase implements VideoClip {
        void play() {
           <código>
           }
        void bucle() {
           <código>
           }
        void stop() {
           <código>
           }
```



Paquetes

La palabra clave **package** permite agrupar clases e interfaces. Los nombres de los paquetes son palabras separadas por puntos y se almacenan en directorios que coinciden con esos nombres. Por ejemplo, los ficheros siguientes, que contienen código fuente Java:

Applet.java, AppletContext.java, AppletStub.java, AudioClip.java

contienen en su código la línea:

package java.applet;

Y las clases que se obtienen de la compilación de los ficheros anteriores, se encuentran con el nombre nombre_de_clase.class, en el directorio:

java/applet

Import

Los paquetes de clases se cargan con la palabra clave **import**, especificando el nombre del paquete como ruta y nombre de clase (es lo mismo que #include de C/C++). Se pueden cargar varias clases utilizando un asterisco.

import java.Date;
import java.awt.*;

Si un fichero fuente Java no contiene ningún package, se coloca en el paquete por defecto sin nombre. Es decir, en el mismo directorio que el fichero fuente, y la clase puede ser cargada con la sentencia **import**:

import MiClase;

Biblioteca Básica

java.lang...	Clases de muy bajo nivel
java.util...	Clases útiles (arrays, sets, ...)
java.text	Texto y formateo
java.math	Calculo de precisión arbitrario
java.io	E/S de streams de bytes,...
java.nio	(idem) pero nueva
java.net	Red
java.rmi...	Invocación de métodos remotos
org.omg...	CORBA
java.awt....	Gráficos, ventanas, ...
javax.swing...	(idem) pero mejor organizado
java.beans...	Para desarrollar componentes reusables
Java.applet	Soporte de applets
Java.sql	Soporte JDBC
Java.security...	Seguridad, certificados, ...

Otras Bibliotecas

- El Java API 4 tiene 135 paquetes y 2738 clases/intefaces
  - JavaBeans Activation Framework
  - InfoBus — intecambio entre JavaBeans
  - Java Communications — Serie y paralelo...
  - Java Naming and Directory Interface: LDAP
  - JavaMail
  - JavaHelp
  - JavaServlet
  - Java.Cryptography
  - javax. ... — Menos (lo ya comentado de Swing)
  - Java 3D — Gráficos 3D
  - ...

JSP

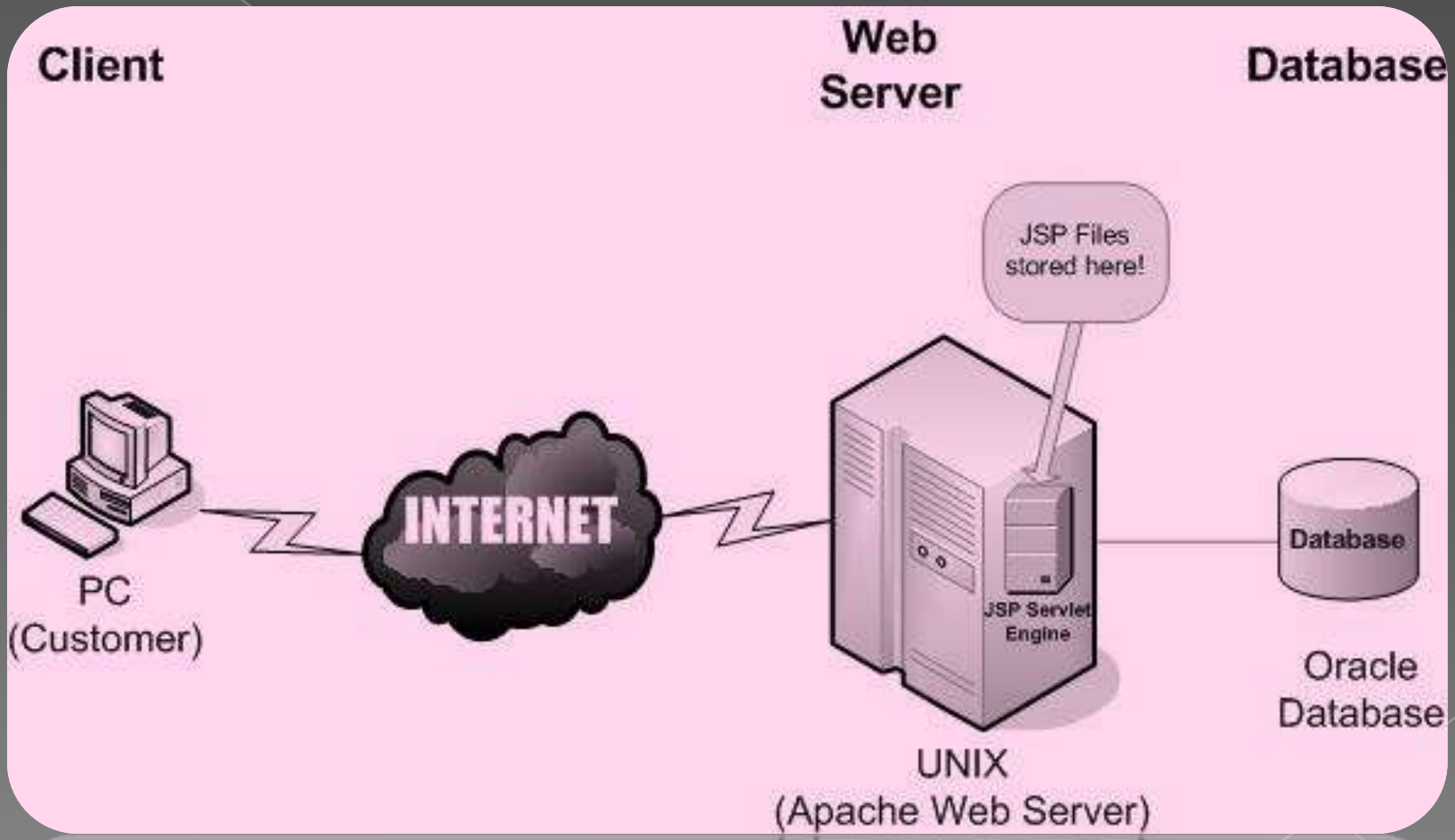
JavaServer Pages (**JSP**) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

¿Cómo funciona un JSP?

El funcionamiento general de la tecnología JSP es que el Servidor de Aplicaciones interpreta el código contenido en la página JSP para construir el código Java del servlet a generar. Este servlet será el que genere el documento (típicamente HTML) que se presentará en la pantalla del Navegador del usuario.

El denominado *contenedor JSP* (que sería un componente del servidor web) es el encargado de tomar la página, sustituir el código Java que contiene por el resultado de su ejecución, y enviarla al cliente.

Diagrama JSP-Oracle



Sintaxis Variables Implícitas

Las páginas JSP incluyen ciertas variables privilegiadas sin necesidad de declararlas ni configurarlas:

Variable	Clase
pageContext	javax.servlet.jsp.PageContext
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
out	javax.servlet.jsp.JspWriter
page	java.lang.Object
exception	java.lang.Exception

Sintaxis Variables Implícitas

Objeto	Significado
request	el objeto `HttpServletRequest` asociado con la petición
response	el objeto `HttpServletResponse` asociado con la respuesta
out	el `Writer` empleado para enviar la salida al cliente. La salida de los JSP emplea un *buffer* que permite que se envíen cabeceras HTTP o códigos de estado aunque ya se haya empezado a escribir en la salida (`out` no es un `PrintWriter` sino un objeto de la clase especial `JspWriter`).
session	el objeto `HttpSession` asociado con la petición actual. En JSP, las sesiones se crean automáticamente, de modo que este objeto está instanciado aunque no se cree explícitamente una sesión.
application	el objeto `ServletContext`, común a todos los servlets de la aplicación web.
config	el objeto `ServletConfig`, empleado para leer parámetros de inicialización.
pageContext	permite acceder desde un único objeto a todos los demás objetos implícitos
page	referencia al propio servlet generado (tiene el mismo valor que `this`).Como tal, en Java no tiene demasiado sentido utilizarla, pero está pensada para el caso en que se utilizara un lenguaje de programación distinto.
exception	Representa un error producido en la aplicación. Solo es accesible si la página se ha designado como página de error (mediante la directiva `page isErrorPage`).

Sintaxis Directivas

Son etiquetas a partir de las cuales se genera información que puede ser utilizada por el motor de JSP. No producen una salida visible al usuario sino que configura cómo se ejecutará la página JSP.

Su sintaxis es:

<% Directiva atributo="valor" %>

Las directivas disponibles son:

- **include**
- **taglib**
- **page**

Incluide, taglib y page

- **Inlude**: Incluye el contenido de un fichero en la página mediante el atributo *file*.

```
<%@ include file="cabecera" %>
```

- **taglib**: Importa bibliotecas de etiquetas (Tag Libraries)

```
<%@ taglib uri="mistags.html" prefix="html" %>
```

- **page**: Especifica atributos relacionados con la página a procesar.

page

Atributo	Significado	Ejemplo
`import`	el equivalente a una sentencia `import` de Java	`<%@ page import="java.util.Date" %>`
`contentType`	genera una cabecera HTTP `Content-Type`	`<%@ page contentType="text/plain" %>`
`isThreadSafe`	si es `false`, el servlet generado implementará el interface `SingleThreadModel` (ún único hilo para todas las peticiones). Por defecto, el valor es `true`.	
`session`	Si es `false`, no se creará un objeto `session` de manera automática. Por defecto, es `true`.	
`buffer`	Define el tamaño del *buffer* para la salida (en kb), o `none` si no se desea *buffer*. Su existencia permite generar cabeceras HTTP o códigos de estado aunque ya se haya comenzado a escribir la salida.	`<%@ page buffer="64kb" %>`
`autoflush`	Si es `true` (valor por defecto), el buffer se envía automáticamente a la salida al llenarse. Si es `false`, al llenarse el buffer se genera una excepción.	
`extends`	Permite especificar de qué clase debe descender el servlet generado a partir de la página JSP. No es habitual cambiarlo.	
`info`	define una cadena que puede obtenerse a través del método `getServletInfo`	`<%@ page info="carro de la compra" %>`
`errorPage`	especifica la página JSP que debe procesar los errores generados y no capturados en la actual.	`<%@ page errorPage="error.jsp" %>`
`isErrorPage`	Si es `true`, indica que la página actúa como página de error para otro JSP. El valor por defecto es `false`.	
`language`	Permite especificar el lenguaje de programación usado en el JSP. En la práctica, el lenguaje siempre es Java, por lo que esta directiva no se usa.	
`pageEncoding`	define el juego de caracteres que usa la página. El valor por defecto es `ISO-8859-1`.	

Etiquetas JSP

Son las etiquetas pertenecientes a la especificación JSP. Proporcionan una funcionalidad básica.

Un primer grupo de etiquetas proporciona funcionalidad a nivel de la página de una manera muy simple:

- **<jsp:forward>**, redirige la request a otra URL
- **<jsp:include>**, incluye el texto de un fichero dentro de la página
- **<jsp:plugin>**, descarga un plugin de Java (una applet o un Bean).

Etiquetas JSP

Un segundo grupo permite manipular componentes JavaBean sin conocimientos de Java.

- **<jsp:useBean>**, permite manipular un Bean*_(si no existe, se creará el Bean), especificando su ámbito (scope), la clase y el tipo.
- **<jsp:getProperty>**, obtiene la propiedad especificada de un bean previamente declarado y la escribe en el objeto response.
- **<jsp:setProperty>**, establece el valor de una propiedad de un bean previamente declarado.

* Bean: son unidades software reutilizables y auto-contenidas que pueden ser unirse visualmente en componentes compuestos, applets, aplicaciones y servlets utilizando herramientas visuales de desarrollo de aplicaciones.

Etiquetas JSTL

Son proporcionadas por Sun dentro de la distribución de JSTL.

- **core**, iteraciones, condicionales, manipulación de URL y otras funciones generales.
- **xml**, para la manipulación de XML y para XML-Transformation.
- **sql**, para gestionar conexiones a bases de datos.
- **i18n**, para la internacionalización y formateo de las cadenas de caracteres como cifras.

Etiquetas Struts TagLib

Distribuidas por Apache para funcionar junto con el Framework de Struts.

- **Bean**
- **HTML**
- **Logic**
- **Nested**
- **vjgp**

Continuará...