



DEPARTAMENTO DE INGENIERIA EN COMPUTACIÓN  
TÉCNICO EN INGENIERIA DE SISTEMAS INFORMÁTICOS

# Manual de Aplicaciones Cliente Servidor

PRIMERA EDICIÓN  
SANTA TECLA, JUNIO DEL 2010

DOCENTE : \_\_\_\_\_  
INSTRUCTOR : \_\_\_\_\_  
ALUMN@ : \_\_\_\_\_  
SECCIÓN : \_\_\_\_\_  
CICLO : \_\_\_\_\_



## **Reglamento del Centro de Computo**

1. Cada estudiante es responsable del uso adecuado del mobiliario y equipo instalado en el Centro de Cómputo.
2. Antes de iniciar su práctica verifique que su equipo esté completo y en buen estado.
3. Comunicar inmediatamente al docente o instructor sobre posibles fallas en el mobiliario y equipo.
4. Presentarse puntualmente al Centro de Cómputo con su respectivo manual de prácticas y su diskette de trabajo.
5. Portar su carné de identificación, talonario u otro documento que lo acredite como estudiante activo del ITCA.
6. Ingresar al Centro de cómputo únicamente cuando su instructor esté presente.
7. Se prohíbe totalmente la instalación o desinstalación de software, así como las modificaciones en la configuración del equipo.
8. Se prohíbe acceder al Internet durante la hora de práctica.
9. Se prohíbe fumar y/o ingresar alimentos o bebidas en el Centro de Cómputo.
10. Evitar el uso de aparatos de sonido, celulares y beepers durante la práctica.
11. Se prohíbe reproducir CD's de música en el Centro de Cómputo.
12. No se permiten los juegos de computadoras, ver pornografía ni usar salas de chat.
13. Se prohíbe ingresar partes o accesorios de computadoras.
14. No botar o esparcir basura.
15. No manipular los controles del aire acondicionado.
16. No se permite levantarse innecesariamente durante la práctica
17. Se prohíbe maquillarse o peinarse dentro del Centro de Cómputo.
18. Retirarse de la práctica en el momento que el instructor le indique.
19. Antes de retirarse del Centro de Cómputo, revise su área de trabajo, verificando que el equipo quede apagado.
20. El instructor no se hará responsable de objetos olvidados en el Centro de Cómputo.
21. El incumplimiento de las reglas anteriores conllevará a la suspensión temporal o la retribución monetaria según el daño causado.



<b>Contenido</b>	<b>Pagina</b>
<b>Clase N° 1 Conceptos Generales Aplicaciones Cliente–Servidor.</b>	<b>5</b>
<b>Guía Práctica N° 1 Instalación del IDE (Entorno de Desarrollo Integrado) y Aplicaciones básicas con Java.</b>	<b>9</b>
<b>Clase N° 2 Elementos básicos de lenguajes de programación en ambiente cliente/servidor</b>	<b>16</b>
<b>Guía Práctica N° 2 Estructuras de control de JAVA: if, swich, for, while.</b>	<b>22</b>
<b>Clase N° 3 Clases, Atributos , Métodos y Manejo de Excepciones.</b>	<b>26</b>
<b>Guía Práctica N° 3 Clases, Atributos , Métodos y Manejo de Excepciones.</b>	<b>35</b>
<b>Clase N° 4 Introducción a la interfaz Grafica y Modelo de Eventos</b>	<b>38</b>
<b>Guía Práctica N° 4 Esquema de una aplicación orientada a eventos</b>	<b>41</b>
<b>Clase N° 5 Desarrollo de Interfaces.</b>	<b>44</b>
<b>Guía Práctica N° 5 Desarrollo de Interfaces.</b>	<b>47</b>
<b>Clase N° 6 Introducción a la Tecnología JSP</b>	<b>49</b>
<b>Guía Práctica N° 6 Introducción a Java Server Pages</b>	<b>55</b>
<b>Clase N° 7 JSP con bases de datos</b>	<b>61</b>
<b>Guía Práctica N° 7 JSP con Bases de Datos</b>	<b>66</b>
<b>Clase N° 8 Manejo de sesiones y cookies con JSP</b>	<b>72</b>
<b>Guía Práctica N° 8 Manejo de sesiones y cookies con JSP</b>	<b>80</b>
<b>Clase N° 9 Introducción a Servlets</b>	<b>85</b>
<b>Guía Práctica N° 9 Servlets Básico</b>	<b>94</b>
<b>Clase N° 10 Acceso a Base de Datos con Java Servlets</b>	<b>103</b>
<b>Guía Práctica N° 10 Bases de Datos con Java Servlets (Uso de Excepciones)</b>	<b>108</b>
<b>Clase N° 11 Utilidades para programar en JAVA Servlets</b>	<b>115</b>
<b>Guía Práctica N° 11 Gráficos en Aplicaciones de Java</b>	<b>121</b>



## Sistema de Evaluación Aplicaciones Cliente Servidor



### DEPARTAMENTO DE INGENIERIA EN COMPUTACION

ACTIVIDAD	PONDERACIÓN	FECHA	CONTENIDO
EVALUACIÓN TEORICA 1	30%	Semana 4	Unidad 1
EVALUACIÓN TEORICA 2	35%	Semana 8	Unidad 2
EVALUACIÓN TEORICA FINAL	35%	Semana 14	Unidad 3
<b>TOTAL TEORIA (40%)</b>	<b>100%</b>		
EVALUACIÓN PRACTICA 1	15%	Semana 4	Unidad 1
EVALUACIÓN PRACTICA 2	15%	Semana 8	Unidad 2
EVALUACIÓN PRACTICA 3	20%	Semana 14	Unidad 3
PROYECTO	50%	Semana 16	Proyecto
<b>TOTAL PRACTICA (60%)</b>	<b>100%</b>		



## Clase N° 1 Conceptos Generales Aplicaciones Cliente-Servidor.

### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir conceptos de aplicaciones cliente servidor.
- Exponer ventajas y desventajas de aplicaciones clientes servidor.
- Identificar diferentes tecnologías de Java como lenguaje de aplicaciones cliente servidor.

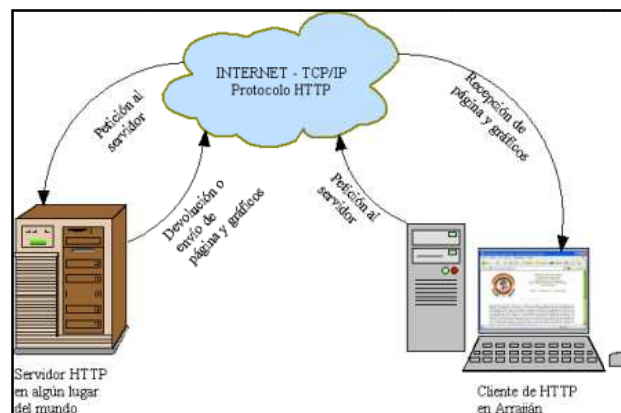
### DESARROLLO

#### CLIENTE SERVIDOR.

Modelo Cliente – servidor: divide las aplicaciones comunicantes en dos categorías, dependiendo de si la aplicación se queda en espera de conexiones (servidor) o las inicia (cliente).

En general, una aplicación que inicia una comunicación con otra se la califica como cliente. Los usuarios finales invocan aplicaciones cliente cuando utilizan un servicio de red. Cada vez que se ejecuta una aplicación cliente, esta contacta con el servidor, le envía una solicitud de servicio y espera la respuesta o resultados del servicio. El proceso cliente es el encargado de llevar a cabo la interacción con el usuario y de mostrar los resultados de las peticiones de servicio. En la mayoría de las ocasiones los clientes son mas fáciles de diseñar que los servidores, y no suelen precisar privilegios especiales del sistema para poder funcionar.

Servidor es un programa que espera peticiones de servicio por parte de un cliente. El servidor recibe la petición del cliente, ejecuta el servicio solicitado y retorna los resultados al cliente. No existe una interacción directa entre el usuario y el servidor, de esto ya se encarga la aplicación cliente.



Las aplicaciones emplean el modelo cliente-servidor donde las funciones como, los inicios de sesión y el almacenamiento de datos pueden residir en sistemas diferentes.

#### **Características de un cliente**

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación (dispositivo maestro o amo).
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario(GUI).

#### **Características de un servidor**

- Al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número de clientes (en ciertos casos el número máximo de peticiones puede estar limitado).
- No es frecuente que interactúen directamente con los usuarios finales.

#### **Ventajas**

**Centralización del control:** Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema. Esta centralización también facilita la tarea de poner al día datos u otros recursos.

**Escalabilidad:** Se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).

**Fácil mantenimiento:** Al estar distribuidas las funciones y responsabilidades entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente). Esta independencia de los cambios también se conoce como encapsulación.

#### **Desventajas**

**Congestión del tráfico:** cuando una gran cantidad de clientes envían peticiones simultáneas al mismo servidor, puede ser que cause muchos problemas para éste (a mayor número de clientes, más problemas para el servidor).

**Centralización de recursos:** cuando un servidor está fuera de línea, apagado o ha tenido algún problema para el inicio, las peticiones de los clientes no pueden ser satisfechas.

**El software y el hardware:** son generalmente muy determinantes. Un hardware regular de un ordenador personal puede no poder servir a cierta cantidad de clientes. Normalmente se necesita software y hardware específico, sobre todo en el lado del servidor, para satisfacer el trabajo. Por supuesto, esto aumentará el coste.

### **¿Qué es JAVA?**

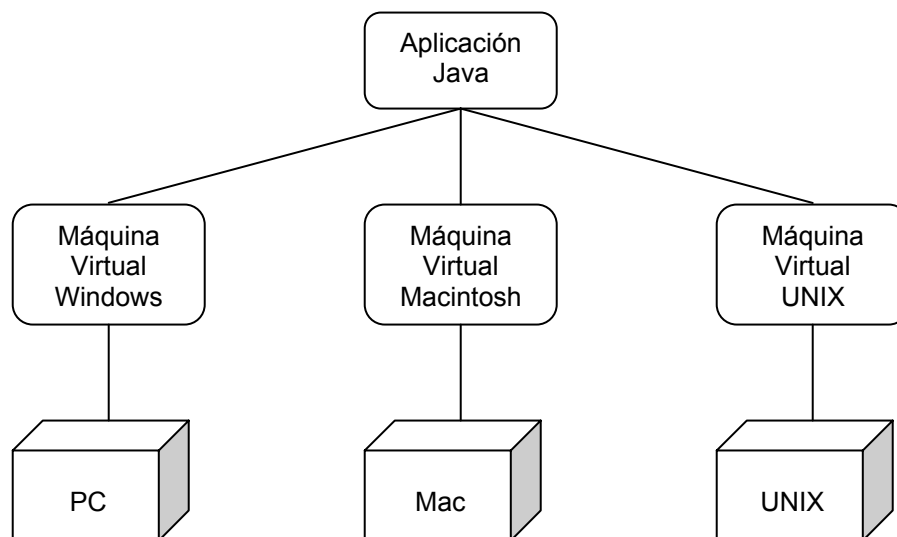
Java es un lenguaje de programación de Alto nivel independiente de la plataforma. Que se desarrolla en principios para la creación de aplicaciones que sean independientes del hardware en que se ejecuta y que poseen pocos recursos. Java elimina todas aquellas instrucciones y funciones imprescindibles que en muchas ocasiones son causantes de errores logrando mantener todas las características de un lenguaje de alto nivel.

En 1995 es lanzado como una novedad para la creación de aplicaciones pero no se queda como un lenguaje para la creación de aplicaciones de escritorio, gracias al auge del Internet se descubre otra característica que marco el rumbo del posicionamiento de java como un lenguaje para la creación de contenidos para la Web y es así que para 1996 a través de Netscape 2.0, y la Web ya no volvió a ser lo mismo.

Uno de los mensajes comerciales de Java fue "Escribir una vez, funcionar en cualquier lugar". En teoría el programa solo debe codificarse una vez, y debe funcionar en cualquier maquina con soporte Java para Windows y Unix, mientras que Apple controla la versión Macintosh. Originalmente.

### **Máquina Virtual de Java**

Uno de los acrónimos más empleados en Java es JVM, que procede del término en inglés Java Virtual Machine (Máquina Virtual Java). Entender que tiene Java de especial los programas Java no hablan directamente a la computadora, sino que lo hacen a la JVM, que a su vez se encarga de comunicarse con aquélla. La JVM es como un traductor entre el código Java y la computadora, la razón por la cual dicho código es considerado como un código interpretado en lugar compilado(es decir específico de una máquina). La JVM es un programa específico que se ejecuta en la computadora. Su único propósito es tomar programas Java y convencer a la computadora de que lo que esta ejecutando es algo que se a desarrollado específicamente para ella.



En conclusión Java puede ejecutarse en cualquier hardware, razón por la cual existe una Máquina Virtual Java, o JVM funcionando en el.

#### *Características de Java*

- Java es simple.
- Destaca por su robustez.
- Es interpretado.
- Java es distribuido.
- Ante todo es portable.
- Con una arquitectura independiente (neutral).
- Un lenguaje orientado a objetos.
- Además, es dinámico.
- Su seguridad es muy alta.
- Permite actividades simultáneas.



#### *Tecnologías aplicadas de Java:*

J2SE (Java 2 Standard Edition): Es una colección de Applets del lenguaje de programación Java útiles para muchos programas de la Plataforma Java. La Plataforma Java 2, Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre Estaciones de Trabajo.

J2ME (Java 2 Micro Edition): Es una colección de Applets de Java para el desarrollo de software para dispositivos de recursos limitados, como PDA, teléfonos móviles y otros aparatos de consumo.

J2EE (Java 2 Enterprise Edition): Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en [Java](#) con arquitectura de N niveles distribuidos, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

#### **Tarea:**

-  Investigar y hacer un resumen con todas las características de java.
-  Hacer un cuadro comparativo con las tecnologías aplicadas de java.





## Guía Práctica No 1 Instalación del IDE (Entorno de Desarrollo Integrado) y Aplicaciones básicas con Java.

### OBJETIVOS

Al finalizar la práctica, el estudiante será capaz de:

- Instalar y configurar el entorno de desarrollo de aplicaciones en Java.
- Utilizar los componentes básicos del IDE de programación para la creación de aplicaciones en Java.
- Crear programas básicos con Java.

### PROCEDIMIENTO

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI (Interfaz Gráfica de Usuario) e interprete.

Para el desarrollo de nuestras aplicaciones usaremos el NETBANS un IDE con el que se puede trabajar Java de una forma muy sencilla.

El NetBeans IDE es un entorno de desarrollo, una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

### ***Pasos para la instalación del NetBeans***

Antes de instalar el NetBeans necesitamos tener instalado el JSDK (Java Software Development Kit).

Para instalarlo daremos doble clic en el icono siguiente



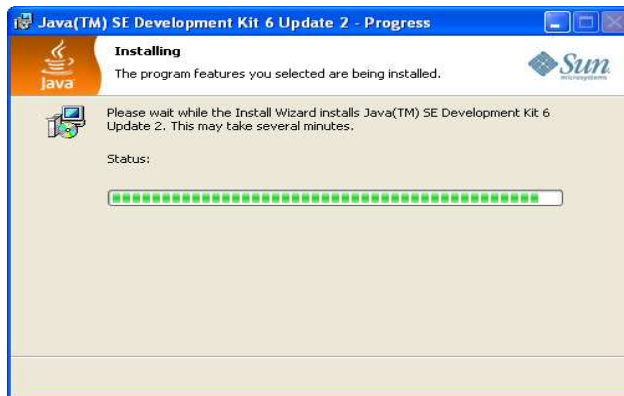
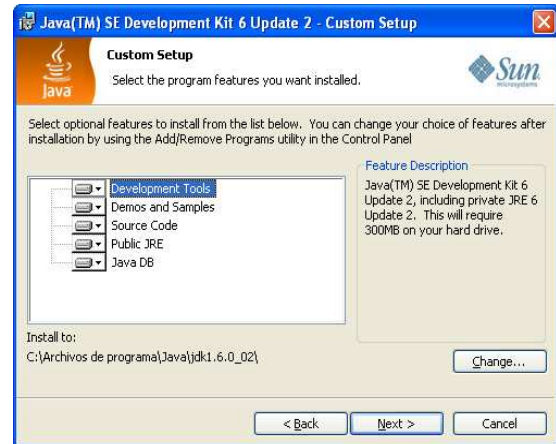
Aparecerá la pantalla de bienvenida del instalador como la que se muestra a continuación.





Después de un instante de espera aparecerá una ventana con la licencia la cual debemos aceptar dando clic en Accept.

Después de aceptar la licencia aparecerá otra ventana que indica los componentes que se instalarán y la dirección donde lo hará, solamente presionaremos Next (Si lo desea puede cambiar la dirección dando clic en Change).



Al instante aparecerá una ventana que indica el estado de la instalación, esperaremos unos instantes para que termine de instalar todos los componentes.



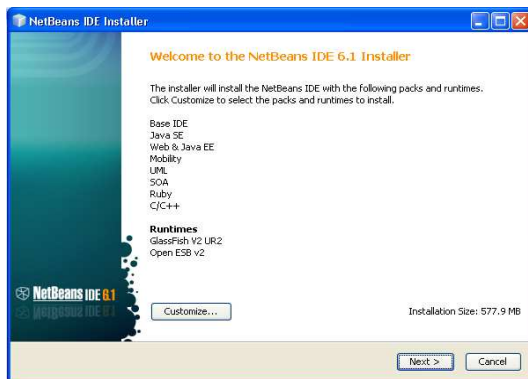
Luego solo nos resta dar clic en Finish como lo muestra en la imagen.

Ahora que ya instalamos el JSDK podemos instalar el NetBeans para ello haremos lo siguiente:



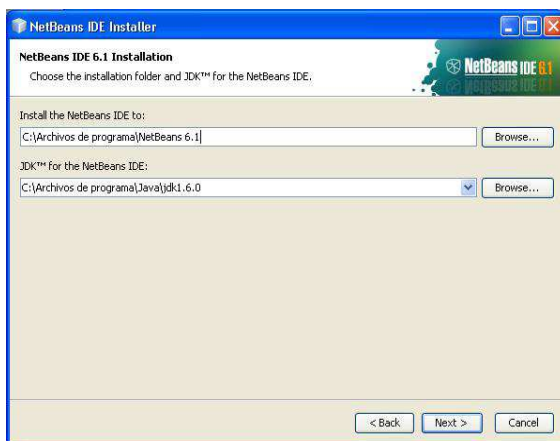
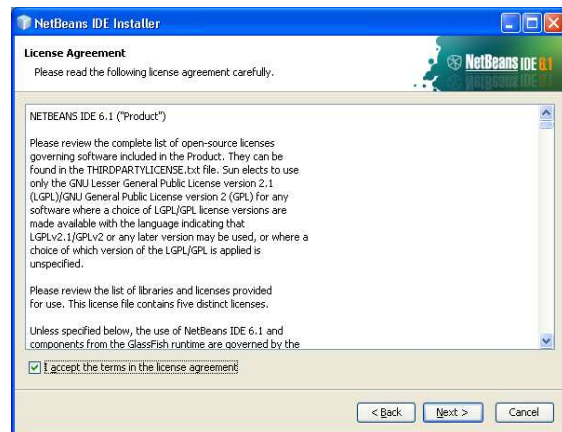
De doble clic en el instalador del NetBeans

Aparecerá una ventana indicando que se está Configurando el Instalador como la siguiente:



La ventana siguiente es la pantalla principal del instalador donde se muestra la versión que estamos instalando del NetBeans y las tecnologías que podemos trabajar con en esta pantalla debemos dar clic en Next.

Aparecerá la licencia del NetBeans la cual debemos aceptar dando clic en I accept the terms... y se activa el botón Next.

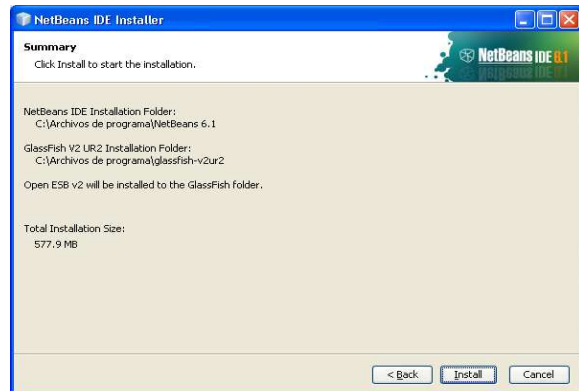
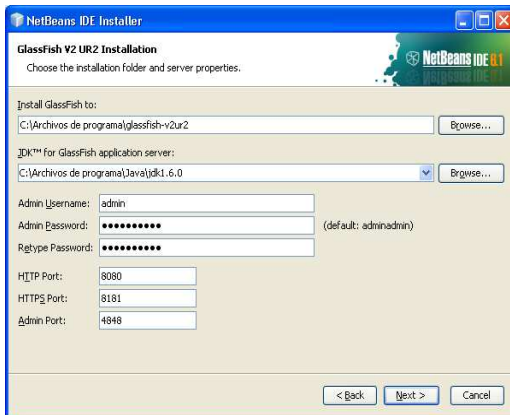


Luego nos mostrará la ruta donde se instalará el NetBeans dejaremos las direcciones que se muestran en la pantalla y demos clic en Next, (Si lo desea puede cambiar la ruta de instalación)

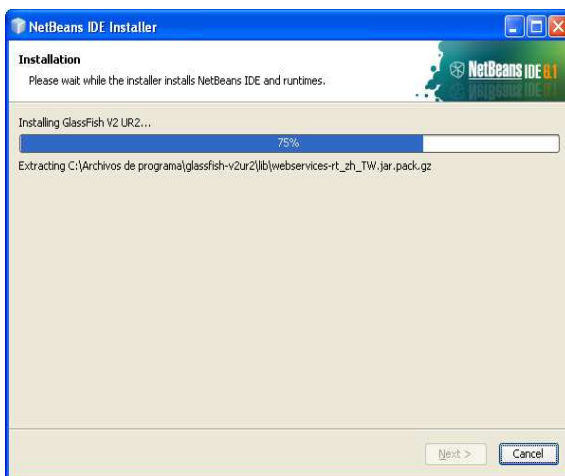
En ocasiones aparecerá una alerta de seguridad la que preguntará si deseamos bloquear Java(TM) Platform SE binary para no tener problemas posteriores daremos clic en Desbloquear.



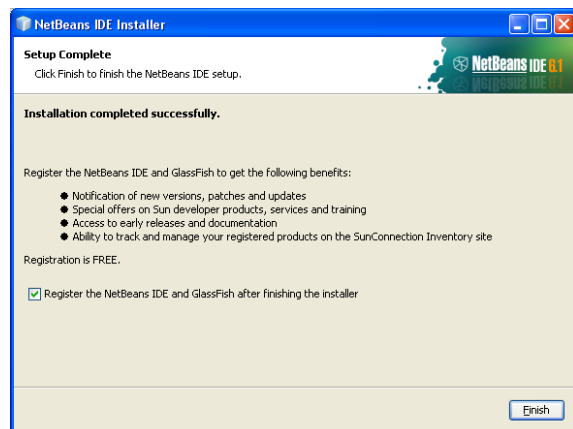
A continuación nos pedirá la configuración de los parámetros del servidor Web simplemente dejaremos las opciones tal como están y presionamos Next.



Luego nos mostrara la siguiente pantalla donde daremos clic en Install



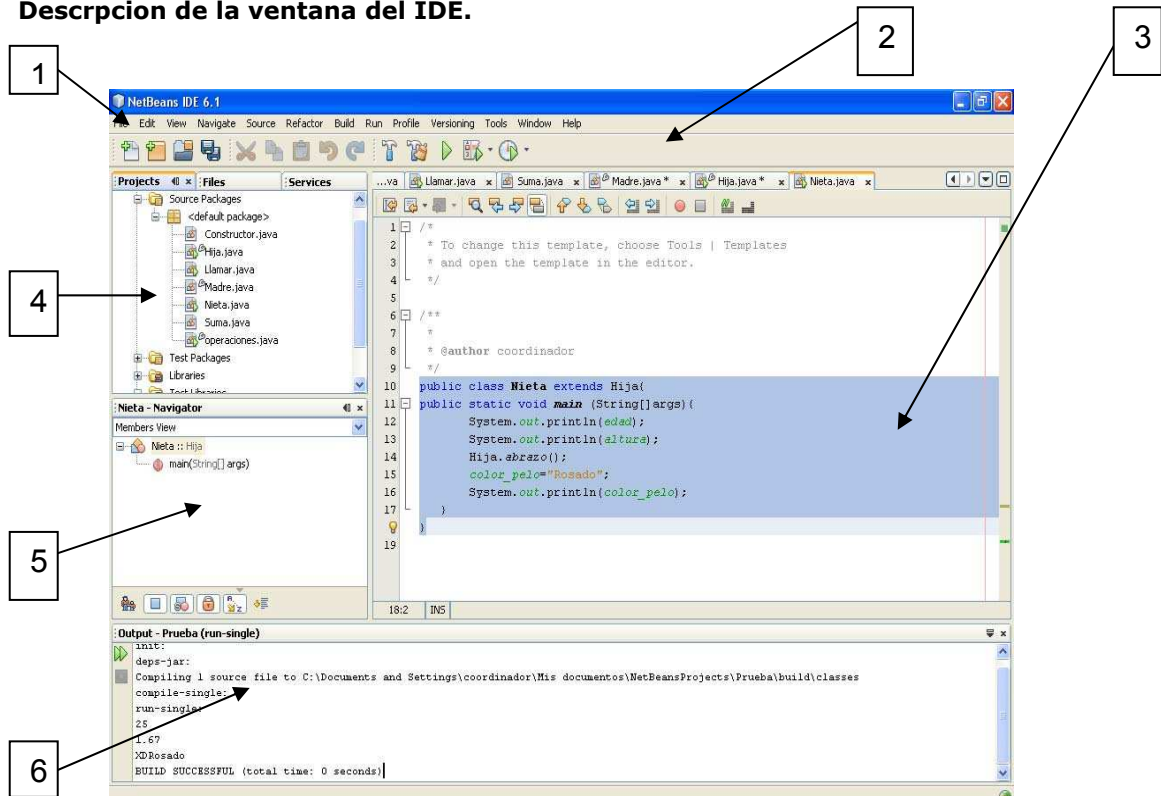
Aparecerá una ventana que nos muestra el estado de la instalación y debemos esperar ya que esto tomara un par de minutos.



Durante la instalación podría aparecer otra Alerta de seguridad simplemente has clic en Desbloquear si esto sucede. Al final daremos clic en Finish.

Terminada la instalación ya estamos preparados para utilizar el NetBeans al ejecutar la aplicación que se encuentra en **inicio > todos los programas > NetBeans>NetBeans IDE 6.1** nos aparece esta pantalla que es la ventana principal del IDE.

#### Descripción de la ventana del IDE.



1. Barra de Menú.
2. Barra de Herramientas
3. Área de Trabajo (codificación y diseño).
4. Navegador de proyectos.
5. Navegador de Elementos de la Clase.
6. Salidas o resultados.

Para la creación de aplicaciones con el IDE lo primero que crearemos será un proyecto el cual nos servirá para almacenar todas las clases y elementos que creemos para nuestra aplicación en Java. Esto lo hacemos desde **File>New Project....** o damos clic en el icono en la barra de herramientas. Lo que haremos al hacer cualquiera de estas acciones será seleccionar el tipo de proyecto (en nuestro caso será Categories:Java; projects:Java Application) definiremos el nombre y la ubicación del proyecto. Hecho esto ya podemos empezar a crear las clases que serán la base de nuestros programas.

Para Crear una clase lo haremos desde **File>New File....** O en el icono de la barra de menú al igual que el proyecto seleccionamos el tipo de archivo (en nuestro caso será Categories:Java; File Types:Java Class) y definimos el nombre. Terminado este proceso en el área de trabajo aparecerá una plantilla de una clase la cual solo nos queda definir los códigos necesarios para su funcionamiento.

#### Primer programa en Java.

Crearemos una clase denominada HolaMundo con la cual se muestra un mensaje en pantalla, el código de la clase será el siguiente.

### HolaMundo

```
public class HolaMundo {
    public static void main(String[] args){
        System.out.print("Hola Mundo");
    }
}
```

Lo único que mostrara este programa es el mensaje "Hola Mundo". Pero para hacer esto primero hay que compilar y después ejecutar la clase como se hace esto lo podemos hacer de dos maneras ya sea por medio de la barra de menú o con un metodomás facil que es por atajos del teclado.

Proceso	Barra de Menú	Atajo de Teclado
Compilar	Build> Compile "nombre de la clase.java"	F9
Ejecutar	Run > Run File > Run "nombre de la clase.java"	Mayus+F6

### Variables.

```
public class Variables {

    public static void main(String[] args){
        int dato1 = 10;
        double dato2 = 15.31213;
        float dato3 = 1.2F;
        char dato4 = 'c';
        boolean dato5 = true;
        String dato6 = "Desarrollo de aplicaciones";
        short dato7=24;
        long dato8 = 45441557864L;

        System.out.println("Tipo de dato int valor: "+dato1);
        System.out.println("Tipo de dato double valor: "+dato2);
        System.out.println("Tipo de dato float valor: "+dato3);
        System.out.println("Tipo de dato char valor: "+dato4);
        System.out.println("Tipo de dato boolean valor: "+dato5);
        System.out.println("Tipo de dato String valor: "+dato6);
        System.out.println("Tipo de dato short valor: "+dato7);
        System.out.println("Tipo de dato long valor: "+dato8);
    }
}
```

### Variables2

```
public class Variables2 {
    public static void main(String[] args){
        int dato1 = 10;
        int dato2 = 15;
        String msg="La Suma es: ";
        System.out.println(msg+(dato1+dato2));
    }
}
```

## Operaciones

```
public class Operaciones {
    public static void main(String[] args){
        int a=10,b=3;
        System.out.println("la suma de a + b = "+(a+b)); // 13
        System.out.println("la resta de a - b = "+(a-b)); // 7
        System.out.println("la multiplicacion de a * b = "+(a*b)); // 30
        System.out.println("la division de a / b = "+(a/b)); // 3
        System.out.println("el modulo de a % b = "+(a%b)); // 1
        System.out.println(" de a > b = "+(a>b)); // true
        System.out.println(" de a < b = "+(a<b)); // false
        System.out.println(" de a == b = "+(a==b)); // false
        System.out.println(" de a != b = "+(a!=b)); // true
        System.out.println(" de (a > b) && (a < b) = "+((a>b)&& (a<b))); // false
        System.out.println(" de (a > b) || (a < b) = "+((a>b)|| (a<b))); // true
        System.out.println(" de (a > b) && !(a < b) = "+((a>b)|| !(a<b))); // true
        System.out.println(" de a++ = "+ a++ + " a = "+a); // 11
        System.out.println(" de ++b = "+ ++b); // 4
    }
}
```

## Ejercicios.

- Crear una clase que imprima la suma de 2 números decimales.
- Crear una clase que calcule el área de un rectángulo.
- Crear una clase que calcule la hipotenusa de un triángulo (investigue las funciones para sacar raíz cuadradas y potencias).

Sea creativo para presentar los resultados, utilicé todas las variables y operadores que considere necesarios.



## Clase N° 2

### Elementos básicos de lenguajes de programación en ambiente cliente/servidor.

#### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Identificar elementos básicos de lenguaje Java para la programación cliente servidor.
- Definir el uso de cada elemento y su aplicación en los programas creado con Java.

#### DESARROLLO

Para todos los que los que ya hemos trabajado con un lenguaje de programación debemos saber que hay elementos básicos, para poder trabajar con dicho lenguaje y estos se apagan a estándares nos permiten tener un mayor control y orden a al hora de utilizar estos elementos para la programación. En este apartado conoceremos y estudiaremos estos electos y el tratamiento que Java hace particularmente a estos, dichos elementos son:

- Comentarios
- Identificadores
- Palabras reservadas
- Variables
- Literales
- Operadores

#### COMENTARIOS.

Nos permiten introducir notas y aclaraciones en el momento de programar para resaltar alguna acción que se realizan en ciertas partes del código, esto se hace para complementar la documentación interna de las aplicaciones creadas en Java, además es una ayuda para la modificación y estudio futuro de las aplicaciones.

El lenguaje nos permite tres tipos de comentarios los cuales son:

- Comentarios de una sola línea //
- Comentarios de bloque /\* \*/
- Comentarios de Documentación /\*\* \*/

#### Ejemplo

```
import java.util.*;

/** Un programa Java simple.
 * Imprime un mensaje y la fecha.
 * @author ITCA
 * @version 1
 */
public class EjemploComentarios {

    /** Inicio de la documentacion
     * @param args Array de Strings.
     * @return No devuelve ningun valor.
     * @throws No dispara ninguna excepción.
     */
    public static void main(String [ ] args) {
        System.out.println("Hola a todos");
        System.out.println(new Date());
    }
}
```



```
}
}
```

Además de los comentarios de documentación encontramos algunas palabras o tags que sirven para definir una mejor documentación

<b>Tipo de tag</b>	<b>Formato</b>	<b>Descripción</b>
<b>Todos</b>	<b>@see</b>	<b>Permite crear una referencia a la documentación de otra clase o método.</b>
<b>Clases</b>	<b>@version</b>	<b>Comentario con datos indicativos del número de versión.</b>
<b>Clases</b>	<b>@author</b>	<b>Nombre del autor.</b>
<b>Clases</b>	<b>@since</b>	<b>Fecha desde la que está presente la clase.</b>
<b>Métodos</b>	<b>@param</b>	<b>Parámetros que recibe el método.</b>
<b>Métodos</b>	<b>@return</b>	<b>Significado del dato devuelto por el método</b>
<b>Métodos</b>	<b>@throws</b>	<b>Comentario sobre las excepciones que lanza.</b>
<b>Métodos</b>	<b>@deprecated</b>	<b>Indicación de que el método es obsoleto.</b>

### IDENTIFICADORES

Son elementos que nos permiten nombrar variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar en una aplicación en Java. Se definen como los nombres dados a cada elemento que utilizamos dentro de la aplicación.

Estos se deben formar siguiendo algunas reglas las cuales son:

- 1- Comienzan con una letra (mayúscula o minúscula), un guión bajo ( \_ ) o un símbolo de dólar (\$).
- 2- Los caracteres siguientes pueden ser letras o dígitos
- 3- No se deben dejar espacios en blanco
- 4- No existe una longitud máxima de caracteres.

Serían identificadores válidos:

```
identificador
nombre_usuario
Nombre_apellido
_variable_del_sistema
$transaccion
```

y su uso sería, por ejemplo:

```
int contador_principal;
char _lista_de_ficheros;
float $cantidad_en_dolares;
```

### PALABRAS RESERVADAS

Son aquellas que tienen un uso especial dentro del lenguaje e identifican elementos que son generales del lenguaje y no pueden ser utilizadas como identificadores para los objetos, variables y otros objetos que nosotros creamos Estas palabras son:

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

## VARIABLES

Son elementos imprescindibles dentro de la programación y las podemos definir como posiciones de memoria, que almacenan un dato. Para utilizar una variable esta debe primero haberse declarado tomando en cuenta ciertos estándares los cuales define que la debe de poseer un tipo de dato, el identificador de dicha variable y el valor que ha de almacenar, solo que este ultimo es opcional a la hora de declarar la variable pero si es importante que se defina cuando se va a procesar. Ejemplos

```
int x=10;
double pago_dolares;
boolean valorReal = false;
String Nombre="Juan Perez";
```

Revisemos la estructura de la declaración de las variables y vemos lo siguiente

Tipos de datos: int, double, boolean, String.  
Identificador: x, pago\_dolares, valorReal, Nombre.  
Valores: 10, false, Juan Perez.

Tipos de Variables.

Además de crear variables debemos conocer su ámbito o alcance de dicho elemento lo cual nos permitirá saber en que partes de nuestros programas puede ser utilizada dicha variable. Entre estas se pueden mencionar las más importantes.

Variables Locales: estas variables solo se pueden usar dentro de bloques de códigos de los programas y cualquier intento de alguna instrucción fuera de este no tendrá acceso a los valores que almacena dicha variable.

Variables globales: este tipo permite el acceso a sus valores desde cualquier lugar dentro del código pero en Java no existen así que se sustituyen con variables de instancia y variables de clase las cuales son las que nos permitirán compartir información entre los objetos.

Variables de clase: estas poseen valores similares para la clase y para todas sus instancias. Para indicar que una variable es una variable de clase se utiliza la palabra clave static en la declaración de la variable.

Variable de instancia: es una variable que está relacionada con una sola instancia de una clase. Cada vez que una instancia de una clase se crea, el sistema crea una copia de la instancia variables relacionadas con esa categoría.

```
public class suma {
    static int y=50; // variable de clase
    static int sumar(){
        int z=4; //variable local y de instancia
        return(z+y);
    }
    public static void main(String[]args){
        System.out.print(sumar());
    }
}
```

## TIPOS DE DATOS

Otro elemento importante dentro del lenguaje Java son los tipos de datos estos nos permiten poder operar y realizar acciones en conjunto con variables como se menciona antes las variables tienen que tener un valor asignado y este valor debe ser de un tipo específico.

En Java encontraremos 8 tipos básicos de de datos los cuales se describen a continuación.

<b>Tipo de Dato</b>	<b>Tamaño</b>	<b>Valor mínimo</b>	<b>Valor máximo</b>
<b>byte</b>	<b>8 bits</b>	<b>-128</b>	<b>127</b>
<b>short</b>	<b>16 bits</b>	<b>-32768</b>	<b>32767</b>
<b>Int</b>	<b>32 bits</b>	<b>-2147483648</b>	<b>2147483647</b>
<b>long</b>	<b>64 bits</b>	<b>-9223372036854775808</b>	<b>9223372036854775807</b>
<b>float</b>	<b>32 bits</b>	<b>±1.40239846e-45</b>	<b>±3.40282347e+8</b>
<b>double</b>	<b>64 bits</b>	<b>±494,065645841246544e-324</b>	<b>±1.79769313486231570e+308</b>
<b>char</b>	<b>16 bits</b>	<b>\u0000</b>	<b>\uffff</b>
<b>boolean</b>	<b>n/a</b>	<b>true / false</b>	<b>true / false</b>

Además encontraremos un tipo de dato que no es básico ya que una librería de java que lo implementa dicho tipo es el String y este se usa para trabajar cadenas de caracteres.

## LITERALES.

Son identificadores que se definen en Java para indicarle al compilador el tipo de dato que tendrá el valor que se ha asignado a una variable y esto se usa para que dicho valor no cambie durante la ejecución del programa. Para definir los literales se usan ciertos caracteres que le dirán al compilador que maneje el dato como una constante.

<b>Literales</b>	<b>tipo</b>	<b>Ejemplo</b>
<b>True y False</b>	<b>booleano</b>	<b>x = true , y = false</b>
<b>24, 150</b>	<b>Entero</b>	<b>Edad = 24, HorasP = 150</b>
<b>2L, 34L</b>	<b>Entero largo</b>	<b>Conteo = 2L, CP=45L</b>
<b>2.3, 1.5E3</b>	<b>double</b>	<b>Desc = 2.3, cap = 1.5E5</b>
<b>23.5f, 10.75f</b>	<b>float</b>	<b>Temp = 23.5f, pi = 3.14f</b>
<b>'a', 'B', 'c'</b>	<b>char</b>	<b>Dia = 'L', Esc = 'K'</b>
<b>"Juan Perez"</b>	<b>String</b>	<b>Nombre = "Juan Perez"</b>

Además existen ciertos caracteres que poseen una función especial en Java las cuales formatean la salida de una impresión. Veamos cuales son en la siguiente tabla.

<b>Caracteres</b>	<b>Significado</b>
<b>\b</b>	<b>Backspace o retroceso</b>
<b>\ddd</b>	<b>Representación Octal.</b>
<b>\f</b>	<b>Formfeed o Avance de hoja</b>
<b>\n</b>	<b>Nueva línea</b>
<b>\r</b>	<b>Retorno de carro</b>
<b>\t</b>	<b>Tabulación</b>
<b>\udddd</b>	<b>Carácter unicode</b>

<code>\xdd</code>	Representación Hexadecimal
<code>\\</code>	Backslash
<code>\'</code>	Comilla Simple
<code>\"</code>	Comilla doble

## OPERADORES.

Estos son elementos imprescindibles de las expresiones u operaciones que se deben realizar en una aplicaron ya que el calcular cuanto es la suma de dos valores, saber si un número es mayor que otro y unir dos o más cadenas de caracteres son acciones que se repiten en todo programa y por eso es necesario contar con los operadores. Estos de agrupan en diversas categorías entre las cuales son:

- Operadores aritméticos
- Operadores booleanos.
- Operadores lógicos.
- Operadores con objetos.
- Operadores de cadena.
- Operadores de gestión de memoria.

### Operadores aritméticos:

Se usan para calcular operaciones aritméticas sobre valores numéricos estas operaciones pueden ser entre uno o mas valores ejemplo.

Operador	Accion	Ejemplo
<b>+</b>	<b>Suma</b>	<b>4 + 5=9</b>
<b>-</b>	<b>Resta o cambio de valor a negativo</b>	<b>5 - 1=6, -4</b>
<b>*</b>	<b>Multipliación</b>	<b>5 * 2=10</b>
<b>/</b>	<b>División</b>	<b>5 / 2=2.5</b>
<b>%</b>	<b>Modulo</b>	<b>5 % 2=1</b>
<b>++</b>	<b>Amento de valor</b>	<b>5++ = 6</b>
<b>--</b>	<b>Disminución de valor</b>	<b>5-- = 4</b>

### Operadores Booleanos, de comparación o Relacionales.

Se usan para devolver valores de verdad en la comparación de dos datos

Operador	Accion	Ejemplo
<b>==</b>	<b>Igual</b>	<b>5==5 = true</b> <b>6==5 = false</b>
<b>!=</b>	<b>Diferente</b>	<b>7!=5 = true</b> <b>6!=6 = false</b>
<b>&lt;</b>	<b>Menor que</b>	<b>6&lt;7 = true</b> <b>8&lt;7 = false</b> <b>6&lt;6 = false</b>
<b>&gt;</b>	<b>Mayor que</b>	<b>6&gt;7 = false</b> <b>8&gt;7 = true</b> <b>6&gt;6 = false</b>
<b>&lt;=</b>	<b>Menor o igual que</b>	<b>6&lt;=7 = true</b> <b>7&lt;=8 = false</b> <b>6&lt;=6 = true</b>
<b>&gt;=</b>	<b>Mayor o igual que</b>	<b>6&gt;=7 = false</b> <b>8&gt;=7 = true</b> <b>6&gt;=6 = true</b>

### Operadores Lógicos.



Estos evalúan expresiones formadas por operandos que a su vez están formados por expresiones y su resultado es un valor de verdad.

Operador	Accion	Ejemplo	Tabla de Verdad
&&	AND o Conjunción	(5<4) && (4==5) = false (8!=8) && (4<5) = false (8==4) && (7<5) = false (8!=4) && (4<5) = true	A B A && B V V V V F F F V F F F F
	OR o Disyunción	(5<4)    (4==5) = false (8!=8)    (4<5) = true (8==8)   (7<5) = true (8!=4)    (4<5) = true	A B A    B V V V V F V F V V F F F
!	NOT o Negación	!(6<7) = false !(6>7) = true	A !A V F F V

### Operadores de Cadena.

Las cadenas al ser una clase se pueden trabajar por medio de métodos ya definidos pero también se pueden operadores que nos ayudaran manipular estos elementos y facilitar su operación. Podemos utilizar operadores para verificar que una cadena es mayor que otra (>), concatenar cadenas (+), comparar cadenas (==).

#### Tarea

-  Investigar y crear una tabla de jerarquía de operadores en Java.
-  Investigar que es el casting y su aplicación en Java.



## Guía Práctica No 2

### Estructuras de control de JAVA: if, swich, for, while.



#### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Crear aplicaciones con estructuras condicionales.
- Crear aplicaciones con estructuras repetitivas.
- Crear aplicaciones con estructuras de control combinadas.

#### PROCEDIMIENTO

##### *Introducción.*

Las estructuras de control son una implementación de los lenguajes de programación para facilitar la creación de aplicaciones en las cuales se deben condicionar o repetir ciertas porciones de código las cuales facilitaran el flujo de cómo los datos se procesaran.

Existen 2 tipos de estructuras de control las cuales podremos implementar en nuestros programas y estos son:

- Condicionales: las cuales a partir de la verificación de ciertos elementos se toman decisiones las cuales afectaran los resultados de las acciones o las salidas de los programas.
- Repetitivas: Estas tienen la función de que repiten la ejecución de ciertos bloques de código lo que nos ayuda a

##### *Estructuras condicionales*

En java utilizaremos las condicionales if, if .....else, switch.

if.

La instrucción la utilizaremos cuando evaluamos si un caso o condición es verdadera.

```
if(x==5){
    System.out.println("Condicion Verdadera");
}
```

if.....else.

Esta es una variante de la estructura if, en esta se evalúa una condición es verdadera pero con una variante con la que obtendremos un resultado aunque la condición sea falsa.

```
if(x==5){
    System.out.println("Condicion Verdadera");
} else {
    System.out.println("Condicion Falsa");
}
```

switch.

Esta estructura se conoce como una condicional multicasos ya que para ejecutar un bloque de instrucciones se evalúa la condición con varios casos, en los cuales se ejecuta solo cuando se evalué un caso de verdad, pero si en algún momento ningún caso coincide se ejecuta un caso por defecto.

```
Int x = 1;
switch(x){
case (1):
    System.out.print("Primer dia de la Semana");
break;
case (2):
    System.out.print("Segundo dia de la Semana");
break;
case (3):
    System.out.print("Tercer dia de la Semana");
break;
case (4):
    System.out.print("Cuarto dia de la Semana");
break;
default:
    System.out.print("Otro dia de la Semana");
}
```

### *Estructuras repetitivas.*

Java cuenta con 3 tipos for, while y la variante do...while.

for.

Se usa cuando sabemos en que momento el ciclo se detendrá y la estructura es la siguiente.

```
class CicloFor {
    public static void main(String[] args){
        int i;
        for(i = 1; i < 101; ++i){
            System.out.print(i + "\t");
        }
    }
}
```

While.

En este ciclo las repeticiones se ejecutan mientras la evaluación de una condición sea verdadera.

```
class CicloWhile {
    public static void main(String[] args){
        int i=1;
        while( i < 101){
            System.out.print(i + "\t");
            ++i;
        }
    }
}
```

do...while

En esta estructura ejecuta primero las instrucciones y después se evalúa la condición para continuar o detener la ejecución de las instrucciones.

```

class CicloDo{
    public static void main(String[] args){
        int i=1;
        do{
            System.out.print(i + "\t");
            ++i;
        }while(i<101);
    }
}

```

Ejemplo aplicado.

```

import java.io.*;

public class casos{
    public static void main(String args[] )throws IOException{
        BufferedReader in =new BufferedReader(new InputStreamReader(System.in));

        int n1,n2,sum,res,div,multi;
        int op;
        System.out.print("Elige una opcion\n" );
        System.out.print("1 = Realizar Suma\n" );
        System.out.print("2 = Realizar Resta\n" );
        System.out.print("3 = Realizar una multiplicacion\n" );
        System.out.print("4 = Realizar una division\n" );
        op=Integer.parseInt(in.readLine());
        switch(op){
            case 1:
                System.out.print("\nintroduce el primer numero \n" );
                n1=Integer.parseInt(in.readLine());
                System.out.print("\nintroduce el segundo numero \n" );
                n2=Integer.parseInt(in.readLine());
                sum=n1+n2;
                System.out.println("\nLa Suma es: "+ sum);
                break;
            case 2:
                System.out.print("\nintroduce el primer numero" );
                n1=Integer.parseInt(in.readLine());
                System.out.print("\nintroduce el segundo numero " );
                n2=Integer.parseInt(in.readLine());
                res=n1-n2;
                System.out.println("\nLa Resta es: "+ res);
                break;
            case 3:
                System.out.print("\nintroduce el primer numero" );
                n1=Integer.parseInt(in.readLine());
                System.out.print("\nintroduce el segundo numero " );
                n2=Integer.parseInt(in.readLine());
                multi=n1*n2;
                System.out.print("\nLa Multiplicacion es: "+ multi);
                break;
            case 4:
                System.out.print("\nintroduce el primer numero" );
                n1=Integer.parseInt(in.readLine());
                System.out.print("\nintroduce el segundo numero " );
                n2=Integer.parseInt(in.readLine());
                if(n2==0){
                    System.out.print("\nError division entre 0 ");
                }else{
                    div=n1/n2;
                    System.out.print("\nLa Division es: "+ div);
                }
            }
        }
    }
}




```



```
}  
break;  
default:  
System.out.print("\neleccion incorrecta" );  
}  
}  
}
```

La instrucción **BufferedReader in =new BufferedReader(new InputStreamReader(System.in))** se utiliza para crear un elemento que nos permitirá hacer lecturas desde el teclado, y **in.readLine()** se utilizara para poder pedir datos por medio de la consola en nuestras aplicaciones en Java

#### Ejercicios

-  Cree una aplicación en Java que a partir del sueldo de un empleado calcule el descuento de la renta, verificar si se puede aplicar dicho descuento y mostrar en pantalla el sueldo total que recibirá el empleado.
-  Cree una aplicación en Java que permita calcular el factorial de un número entero.
-  Cree una aplicación en Java que imprima los primeros 100 números primos.



## Clase N° 3 Clases, Atributos , Métodos y Manejo de Excepciones.



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir que es una clase y las partes que la conforman.
- Definir que es un atributo de la clase
- Definir métodos de la clase
- Definir instancias y referencias en Java
- Definir mecanismos de aplicación de herencia entre clases de Java
- Definir que es una interfase en Java.
- Identificar instrucciones para el manejo de excepciones en Java

### DESARROLLO

#### Introducción

El elemento básico de la programación orientada a objetos en Java es la clase. Una clase define la forma y comportamiento de un objeto.

Para crear una clase sólo se necesita un archivo fuente que contenga la palabra clave reservada `class` seguida de un identificador legal y un bloque delimitado por dos llaves para el cuerpo de la clase.

```
class Ejemplo {
}
```

Un archivo de Java debe tener el mismo nombre que la clase que contiene, y se les suele asignar la extensión ".java". Por ejemplo la clase Ejemplo se guardaría en un fichero que se denomina Ejemplo.java. Hay que tener presente que en Java se diferencia entre mayúsculas y minúsculas; el nombre de la clase y el de archivo fuente han de ser exactamente iguales.

Una clase es un conjunto de métodos y variables relacionadas, basadas en la programación orientada a objetos (POO). Por lo tanto define la estructura de un objeto y su interfaz funcional, en forma de métodos. Cuando se ejecuta un programa en Java, el sistema utiliza definiciones de clase para crear instancias de las clases, que son los objetos reales. Los términos instancia y objeto se utilizan de manera indistinta. La forma general de una definición de clase es:

```
class Nombre_De_Clase {
    tipo_de_variable nombre_de_atributo1;
    tipo_de_variable nombre_de_atributo2;
    // ...
    tipo_devuelto nombre_de_método1( lista_de_parámetros ) {
        cuerpo_del_método1;
    }
    tipo_devuelto nombre_de_método2( lista_de_parámetros ) {
        cuerpo_del_método2;
    }
    // ...
}
```

Los tipos **tipo\_de\_variable** y **tipo\_devuelto**, han de ser tipos simples Java o nombres de otras clases ya definidas. Tanto **Nombre\_De\_Clase**, como los **nombre\_de\_atributo** y **nombre\_de\_método**, han de ser identificadores Java válidos.

Además la clase consta de dos partes fundamentales las cuales son la:

- Declaración de la clase: en esta parte se define el nombre de la clase y la definición de si heredará elementos de otras clases, y otros atributos que serán indispensables según las necesidades de las aplicaciones que creemos.
- Cuerpo de la clase: en esta parte se declaran todos los método(funciones) y atributos(variables), que permiten la ejecución de acciones y devolución de resultados de los procesos de la clase.

Los datos se encapsulan dentro de una clase declarando variables dentro de bloques de código que se distinguen por empezar por una llave de apertura, el contenido del bloque y la llave de cierre, dentro del contenido del bloque de código podremos encontrar variables y funciones.

Vistos los Elementos anteriores vamos a definir los modificadores de acceso que son elementos que indican como se comportan los objetos y si pueden compartir datos entre ellos y otras clases.

#### Modificadores de Clases

- **public** - Todas las clases puede acceder al elemento. Si es un dato miembro, Todas las clases puede ver el elemento, es decir, usarlo y asignarlo. Si es un método Todas las clases puede invocarlo.
- **private** - Sólo se puede acceder al elemento desde métodos de la clase, o sólo puede invocarse el método desde otro método de la clase.
- **protected** - es una combinación de los accesos que proporcionan los modificadores public y private. proporciona acceso público para las clases derivadas y acceso privado para el resto de clases.
- sin modificador - Se puede acceder al elemento desde cualquier clase del package donde se define la clase.

#### Modificadores de métodos variables

- **static** - que se usa para definir datos miembros o métodos como pertenecientes a una clase, en lugar de pertenecer a una instancia.
- **final** - se usa para indicar que un método, un dato miembro (variable) no se podrán redefinir dentro de otra ya sea que se utilice herencia o instancias, además se usa para definir un valor constante en el caso de las variables.
- **abstract** - Se utiliza para crear métodos o clases abstractas o sea que no tienen implementación (nada de código).
- **synchronized** - se usa para indicar que ciertas partes del código, (habitualmente, una función miembro) están sincronizadas, es decir, que solamente un subproceso puede acceder a dicho método a la vez.

### Los atributos

En java a las variables se les conoce como atributos. Se declaran igual que las variables locales de un método en concreto.

Por ejemplo, este es un programa que declara una clase Ejemplo, con dos atributos enteros llamados x e y.

```
class Ejemplo {  
  
    int x, y;  
  
}
```

Los atributos se pueden declarar con dos clases de tipos: un tipo simple Java (int, float, boolean), o el nombre de una clase (será una referencia a objeto).

Cuando se realiza una instancia de una clase (creación de un objeto) se reservará en la memoria un espacio para un conjunto de datos como el que definen los atributos de una clase. A este conjunto de variables se le denomina variables de instancia.

### *Los métodos*

Los métodos son subrutinas que definen la interfaz de una clase, sus capacidades y comportamiento.

Un método ha de tener por nombre cualquier identificador legal distinto de los ya utilizados por los nombres de la clase en que está definido. Los métodos se declaran al mismo nivel que las variables de instancia dentro de una definición de clase.

En la declaración de los métodos se define el tipo de valor que devuelven y a una lista formal de parámetros de entrada, de sintaxis tipo identificador separadas por comas. La forma general de una declaración de método es:

```
tipo_devuelto nombre_de_método( lista-formal-de-parámetros ) {  
    cuerpo_del_método;  
}
```

Por ejemplo el siguiente método devuelve la suma de dos enteros:

```
int metodoSuma( int paramX, int paramY ) {  
    return ( paramX + paramY );  
}
```

En el caso de que no se desee devolver ningún valor se deberá indicar como tipo la palabra reservada void. Así mismo, si no se desean parámetros, la declaración del método debería incluir un par de paréntesis vacíos (sin void):

```
void metodoVacio( ) { };
```

### *La instanciación de las clases: Los objetos*

#### Referencias a Objeto e Instancias

Una referencia a un objeto es el paso previo para obtener una instancia de la clase tipo del objeto. Cuando referenciamos un objeto estamos declarando un objeto (variable) de la clase tipo y le estamos asignando un valor inicial

Ejemplo Ej;

Esta es una declaración de una variable **Ej** que es una referencia a un objeto de la clase **Ejemplo**, de momento con un valor por defecto de null.

Ahora la instancia del objeto se realiza dando un valor a la variable que creamos anteriormente, el valor debe de ser un objeto al que se hace la referencia en este caso la declaración quedaría de la siguiente manera:

```
Ejemplo Ej;
```

```
Ej = new Ejemplo();
```

Esta declaración define que tipo de objeto utilizaremos y como lo llamaremos además que automáticamente implementaremos todas las variables y métodos del objeto al cual se hace la llamada por medio de la declaración **new Ejemplo()**

### Constructores

Las clases pueden implementar un método especial llamado constructor. Un constructor es un método que inicia un objeto inmediatamente después de su creación. De esta forma nos evitamos el tener que iniciar las variables explícitamente para su iniciación.

El constructor tiene exactamente el mismo nombre de la clase que lo implementa; no puede haber ningún otro método que comparta su nombre con el de su clase. Una vez definido, se llamará automáticamente al constructor al crear un objeto de esa clase (al utilizar el operador new).

El constructor no devuelve ningún tipo, ni siquiera void. Su misión es iniciar todo estado interno de un objeto (sus atributos), haciendo que el objeto sea utilizable inmediatamente; reservando memoria para sus atributos, iniciando sus valores.

Por ejemplo:

```
Class Ejemplo{
```

```
    Ejemplo( ) {  
        int x=5;  
        int y=2;  
    }  
}
```

Este constructor denominado constructor por defecto, por no tener parámetros, establece el valor 5 a la variable x y de 2 a la variable y esos valores se iniciaran automáticamente por ser parte del constructor.

El compilador, por defecto, llamará al constructor de la superclase Object() si no se especifican parámetros en el constructor.

Este otro constructor, sin embargo, recibe dos parámetros:

```
public class Datos {  
    int a,b;  
    Datos(int y, int x){  
        a=y;  
        b=x;  
    }  
}
```

La lista de parámetros especificada después del nombre de una clase en una sentencia new se utiliza para pasar parámetros al constructor.

Se llama al método constructor justo después de crear la instancia y antes de que new devuelva el control al punto de la llamada.

Así, cuando ejecutamos el siguiente programa:

```
Datos dat = new Datos(1, 5);
```

```
System.out.println("Dato 1 = " + dat.a );  
System.out.println("Dato 2 = " + dat.b );
```

```
/*
```

```
Se muestra en la pantalla:
```

```
Dato 1 = 1  
Dato 2 = 5 */
```

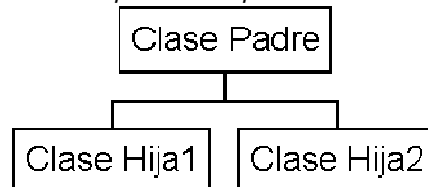
Esto indica que un constructor podrá recibir parámetros como un método cualquiera, pero surge una pregunta ¿cuántos constructores puede tener una clase? La respuesta esta en el siguiente código:

```
public Constructor() {  
    System.out.println("nada");  
}  
  
public Constructor(String tipo) {  
    System.out.println("un valor de Cadena " + tipo);  
}  
  
public Constructor(int distancia) {  
    System.out.println("Un valor entero " + distancia + " metros");  
}  
  
public Constructor(int distancia,String tipo) {  
    System.out.println("Un " + tipo + " corre a " + distancia + " metros");  
}  
}
```

Entonces pueden definirse muchos constructores dentro de una clase siempre y cuando tengan diferentes parámetros y a esto se le denomina como **sobre carga del constructor**.

### La herencia

Es el mecanismo fundamental de relación entre clases en la orientación a objetos. Relaciona las clases de manera jerárquica; una clase *padre* o *superclase* sobre otras clases *hijas* o *subclases*.

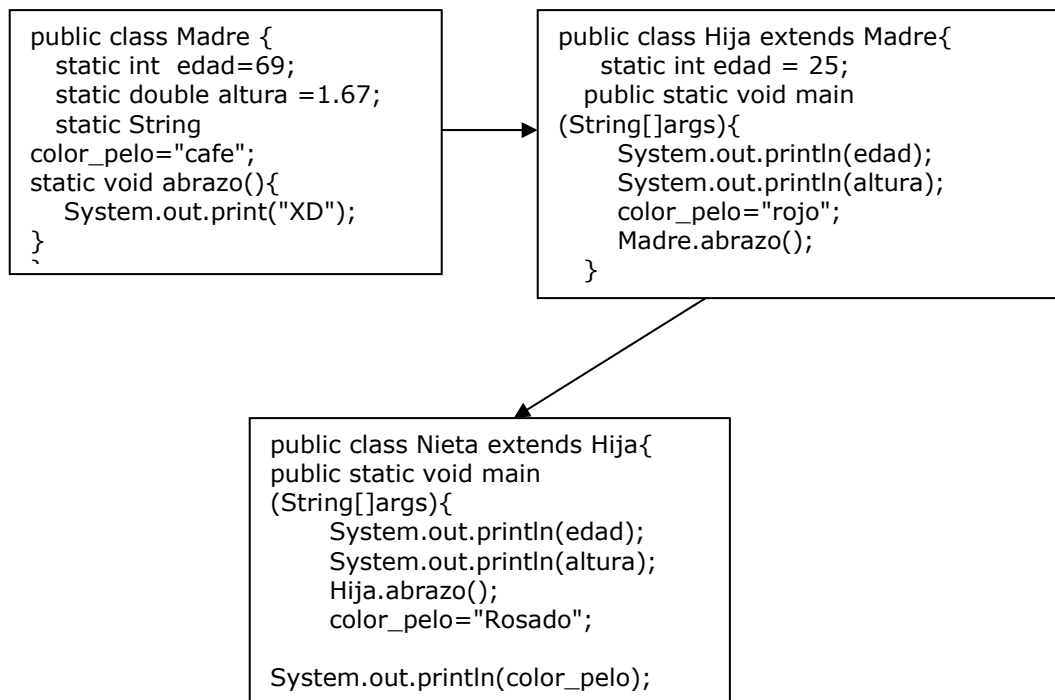


Los descendientes de una clase heredan todas las variables y métodos que sus ascendientes hayan especificado como *heredables*, además de crear los suyos propios.

La característica de herencia, nos permite definir nuevas clases derivadas de otra ya existente, que la especializan de alguna manera. Así logramos definir una jerarquía de clases, que se puede mostrar mediante un árbol de herencia.

En todo lenguaje orientado a objetos existe una jerarquía, mediante la que las clases se relacionan en términos de herencia. En Java, el punto más alto de la jerarquía es la clase *Object* de la cual derivan todas las demás clases.

En java para indicar que una clase heredará de otra se define la palabra reservada **extends**, hay que tomar muy en cuenta que solo se puede heredar de una clase a la vez en java no existe la herencia múltiple, los elementos heredados deben de ser de tipo static ya que un si no se definen de esta forma no se podrá acceder ellos. Veamos un ejemplo de cómo funciona la herencia



En el ejemplo anterior podemos observar que la herencia se hace desde una clase en este caso denominada Madre a la cual se le conoce como súper clase ya que esta no hereda de otras clases pero si comparte atributo y métodos con otras clases. Esta le pasa todos los atributos a la clase **hija** o **sub clase** pero esta a su vez puede modificar los atributos según convenga y además lo heredera también a la clase **nieta** que al igual que la clase hija podrá modificar los atributos según sea necesario. Ahora si no se quisiera heredar algún atributo o método lo único que se hace es definir el modificador de acceso **private** para que ese elemento se único de la clase en que se implementa.

### Interface

El concepto de Interface lleva un paso más adelante la idea de las clases abstractas. En Java una interface es una clase abstracta pura, es decir una clase donde todos los métodos son abstractos (no se implementa ninguno). Permite al diseñador de clases establecer la forma de una clase (nombres de métodos, listas de argumentos y tipos de retorno, pero no bloques de código).

Una interface puede también contener datos miembro, pero estos son siempre static y final. Una interface sirve para establecer un '**protocolo**' entre clases.

Para crear una interface, se utiliza la palabra clave interface en lugar de class. La interface puede definirse public o sin modificador de acceso, y tiene el mismo significado que para las clases. Todos los métodos que declara una interface son siempre public.

Para indicar que una clase implementa los métodos de una interface se utiliza la palabra clave implements. El compilador se encargará de verificar que la clase efectivamente declare e implemente todos los métodos de la interface. Una clase puede implementar más de una interface.

## Declaración y uso

Una interface se declara:

```
interface nombre_interface {
    tipo_retorno nombre_metodo ( lista_argumentos ) ;
    ...
}

interface InstrumentoMusical {
    void tocar();
    void afinar();
    String tipoInstrumento();
}
```

Y una clase que implementa la interface:

```
class InstrumentoViento extends Object implements InstrumentoMusical {
    void tocar() { ... };
    void afinar() { ... };
    String tipoInstrumento() {}
}

class Guitarra extends InstrumentoViento {
    String tipoInstrumento() {
        return "Guitarra";
    }
}
```

La clase InstrumentoViento implementa la interface, declarando los métodos y escribiendo el código correspondiente. Una clase derivada puede también redefinir si es necesario alguno de los métodos de la interface.

## Referencias a Interfaces

Es posible crear referencias a interfaces, pero las interfaces no pueden ser instanciadas. Una referencia a una interface puede ser asignada a cualquier objeto que implemente la interface. Por ejemplo:

```
InstrumentoMusical instrumento = new Guitarra();
instrumento.play();
System.out.println(instrumento.tipoInstrumento());
InstrumentoMusical i2 = new InstrumentoMusical(); //error.No se puede instanciar
```

## Extensión de interfaces

Las interfaces pueden extender otras interfaces y, a diferencia de las clases, una interface puede extender más de una interface. La sintaxis es:

```
interface nombre_interface extends nombre_interface , ... {
    tipo_retorno nombre_metodo ( lista_argumentos ) ;
    ...
}
```

## GESTIÓN DE EXCEPCIONES Y ERRORES



El control de flujo en un programa Java puede hacerse mediante las ya conocidas sentencias estructuradas (*if*, *while*, *return*). Pero Java va mucho más allá, mediante una técnica de programación denominada *gestión de excepciones*.

Mediante las excepciones se podrá evitar repetir continuamente código, en busca de un posible error, y avisar a otros objetos de una condición anormal de ejecución durante un programa.

### *Tipos de excepciones*

Existen varios tipos fundamentales de excepciones:

- **Error:** Excepciones que indican problemas muy graves, que suelen ser no recuperables y no deben casi nunca ser capturadas.
- **Exception:** Excepciones no definitivas, pero que se detectan fuera del tiempo de ejecución.
- **RuntimeException:** Excepciones que se dan durante la ejecución del programa.

Todas las excepciones tienen como clase base la clase **Throwable**, que está incluida en el paquete `java.lang`.

### *Funcionamiento*

Para que el sistema de gestión de excepciones funcione, se ha de trabajar en dos partes de los programas:

- Definir qué partes de los programas crean una excepción y bajo qué condiciones. Para ello se utilizan las palabras reservadas `throw` y `throws`.
- Comprobar en ciertas partes de los programas si una excepción se ha producido, y actuar en consecuencia. Para ello se utilizan las palabras reservadas **`try`**, **`catch`** y **`finally`**.

### *Manejo de excepciones: try - catch – finally*

Cuando el programador va a ejecutar un trozo de código que pueda provocar una excepción (pedir un dato por teclado), debe incluir este fragmento de código dentro de un bloque `try`:

```
try {  
  
    // Código posiblemente problemático  
  
}
```

Pero lo importante es cómo controlar qué hacer con la posible excepción que se cree. Para ello se utilizan las cláusulas `catch`, en las que se especifica qué acción realizar:

```
try {  
  
    // Código posiblemente problemático  
  
} catch( tipo_de_excepcion e) {  
  
    // Código para solucionar la excepción e  
  
} catch( tipo_de_excepcion_mas_general e) {  
    // Código para solucionar la excepción e  
  
}
```

En el ejemplo se observa que se pueden anidar sentencias `catch`, pero conviene hacerlo indicando en último lugar las excepciones más generales (es decir, que se encuentren más

arriba en el árbol de herencia de excepciones), porque el intérprete Java ejecutará aquel bloque de código catch cuyo parámetro sea del tipo de una excepción lanzada.

Si por ejemplo se intentase capturar primero una excepción Throwable, nunca llegaríamos a gestionar una excepción Runtime, puesto que cualquier clase hija de Runtime es también hija de Throwable, por herencia.

Si no se ha lanzado ninguna excepción el código continúa sin ejecutar ninguna sentencia catch. Pero, ¿y si quiero realizar una acción común a todas las opciones?. Para insertar fragmentos de código que se ejecuten tras la gestión de las excepciones. Este código se ejecutará tanto si se ha tratado una excepción (catch) como sino. Este tipo de código se inserta en una sentencia finally, que será ejecutada tras el bloque try o catch:

```
try {  
    } catch( Exception e ) {  
    } finally {  
        // Se ejecutara tras try o catch  
    }
```

### *Lanzamiento de excepciones: throw – throws*

Muchas veces el programador dentro de un determinado método deberá comprobar si alguna condición de excepción se cumple, y si es así lanzarla. Para ello se utilizan las palabras reservadas throw y throws.



Por una parte la excepción se lanza mediante la sentencia *throw*:

```
if ( condicion_de_excepcion == true )  
    throw new miExcepcion();
```

Se puede observar que hemos creado un objeto de la clase miExcepcion, puesto que las excepciones son objetos y por tanto deberán ser instanciadas antes de ser lanzadas. Aquellos métodos que pueden lanzar excepciones, deben indicarse cuáles son esas excepciones en su declaración. Para ello se utiliza la sentencia *throws*:

```
tipo_devuelto miMetodoLanzador() throws miExcep1, miExcep2 {  
    // Código capaz de lanzar excepciones miExcep1 y miExcep2  
}
```

### Tarea

-  investigar los métodos y atributos de la clase object.
-  Investigar los métodos y atributos de la clase Throwable.



## Guía Práctica No 3

### Clases, Atributos , Métodos y Manejo de Excepciones..

#### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Crear referencias e instancias en Java.
- Crear herencia entre clases de Java.
- Crear manipuladores de excepciones en Java.

#### PROCEDIMIENTO

##### *Introducción.*

Las clases en Java son los elementos en los cuales se agrupan los métodos y atributos necesarios para crear aplicaciones ya que un conjunto de clases pueden conformar una aplicación completa y los atributos y métodos relacionados entre estas se comparten y logran un flujo completo de información, por tal motivo es indispensable saber la mejor forma de poder implementarlos y utilizarlos para poder brindar soluciones a los problemas planteados en nuestras aplicaciones primero creemos una clase sencilla la cual llamaremos.

##### *Operaciones.java*

```
public class Operaciones {

    double potencia(double Val1,double Val2){
        return(Math.pow(Val1, Val2));
    }

    double raiz(double Val){
        return(Math.sqrt(Val));
    }

}
```

Ya creada la clase creamos una nueva clase la con la cual crearemos una referencia a la clase operaciones

##### *OpeCuadratica.java*

```
public class OpeCuadratica {
    static void OpCuadratica(double a,double b,double c){
        Operaciones op;
        op=new Operaciones();

        double valorPos, valorNeg,valPot,Op1,ValD;
        Op1=-1*b;
        valPot=op.potencia(b, 2.0)-4*(a*c);
        ValD=2*a;
        If( valPot < 1){
            System.out.print("Error los datos han devuelto una raiz negativa");
        } else {
            valorPos=(Op1+ op.raiz(valPot))/ValD;
            valorNeg=(Op1- op.raiz(valPot))/ValD;
        }
    }
}
```

```

        System.out.print("El resultado del calculo es:\nValor Positivo:"+valorPos+"\nValor
Negativo:"+valorNeg);
    }
}

public static void main(String[]args){

    OpCuadratica(2,3,1);

}
}

```

Ahora utilizaremos la clase creada antes pero esta vez heredaremos las funciones y además agregaremos un manejador de excepción con el cual podremos advertir al usuario si hay un error al introducir los datos

```

import java.io.*;
public class Cuadratica extends OpeCuadratica{

    public static void main(String[]args) throws IOException{
        BufferedReader in ;
        in=new BufferedReader(new InputStreamReader(System.in));
        double a,b,c;
        try{
            System.out.print("Digite el Valor de A:\n");
            a=Double.valueOf(in.readLine());
            System.out.print("Digite el Valor de B:\n");
            b=Double.valueOf(in.readLine());
            System.out.print("Digite el Valor de C:\n");
            c=Double.valueOf(in.readLine());
            OpCuadratica(a,b,c);
        }catch(Exception e){
            System.out.print("Error en la introducción de los Datos" +e.getMessage());
        }

    }

}
}

```

Con estas clases se puede revisar como se aplica la herencia y las referencias a clases y a la vez como se implementa un manejador de excepción.

Revisemos el siguiente ejemplo donde se aplica una sobrecarga de constructores.

mensaje.java

```

public class mensaje {
    mensaje(int val1){
        System.out.print("\nvalor almacenado "+val1);
    }
    mensaje(double val1){
        System.out.print("\nvalor almacenado " +val1);
    }

    mensaje(String val1){
        System.out.print("\nvalor almacenado " +val1);
    }
}




```

```
}
```

Constructores.java

```
public class Constructores {  
    public static void main(String[] args){  
        mensaje m1,m2,m3;  
        m1 = new mensaje(3);  
        m2 = new mensaje(2.38);  
        m3 = new mensaje("Caracteres");  
    }  
}
```

Con estas 2 clases se verifica como se sobrecarga un método constructor, y verificamos que al tener parámetros diferentes no afecta que poseen al mismo nombre.

-  Crear una clase en Java con la cual se sobre cargue 3 constructores los cuales permitan sumar 2 valores del mismo tipo.
-  Crear una clase en Java que implemente 5 métodos para realizar cálculos de áreas de diferentes polígonos (rectángulo, triángulos, rombo etc).
-  Crear una clase que haga la referencia a la clase creada anteriormente e implemente las funciones creadas realizando el calculo de 3 áreas de polígonos diferentes.



## Clase N° 4 Introducción a la interfaz Gráfica y Modelo de Eventos.



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir que es un GUI.
- Definir librerías para la creación de interfaces graficas.
- Definir características de la tecnología Swing de java.

### DESARROLLO

#### *Introducción.*

En el desarrollo de aplicaciones con Java existen 2 formas de trabajo el modo de consola y las ventanas, el trabajo a modo de consola resulta sencillo ya que en línea de comandos se muestra la información de las salidas de los programas de una forma sencilla pero a la vez resulta muy engorroso el trabajo de solo teclear comandos para que nuestros programas procesen información, por tal motivo se hace uso de interfaces graficas para los usuarios(GUI), las cuales se encargan de mostrar por medio de objetos de ventanas(botones, cajas de texto, áreas de escritura), una mejor presentación de nuestro programa y facilita al usuario el uso del mismo. Pero resulta un poco complicado para el programador crear dichos elementos, aunque la mayoría de lenguajes de programación implementa sus propias bibliotecas para la creación de GUI'S. En java podemos encontrar dos bibliotecas las cuales permiten generar las GUI las cuales se importan y permiten la implementación de las clases para la creación de objetos y manejo de eventos estas bibliotecas son:

- Java AWT(Abstract Window Toolkit).
- Java Swing.

#### *AWT*

La Abstract Window Toolkit (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC) - la API estandar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

Dentro del AWT El Contenedor de los Componentes es el **Frame** o se puede denominar como la ventana principal de la aplicación.

El AWT se encuentra desfasado con respecto a la creación de GUI's, pero la biblioteca AWT no se excluye de Java por que su uso que se le da es el del control de eventos.

Estructura del AWT

La estructura de la versión actual del AWT se puede resumir en los puntos que se exponen a continuación:

- Los Contenedores contienen Componentes, que son los controles básicos
- No se usan posiciones fijas de los Componentes, sino que están situados a través de una disposición controlada (layouts)
- El común denominador de más bajo nivel se acerca al teclado, ratón y manejo de eventos
- Alto nivel de abstracción respecto al entorno de ventanas en que se ejecute la aplicación (no hay áreas cliente, ni llamadas a X, ni hWnds, etc.)
- La arquitectura de la aplicación es dependiente del entorno de ventanas, en vez de tener un tamaño fijo
- Es bastante dependiente de la máquina en que se ejecuta la aplicación (no puede asumir que un diálogo tendrá el mismo tamaño en cada máquina)
- Carece de un formato de recursos. No se puede separar el código de lo que es propiamente interface. No hay ningún diseñador de interfaces

Ventana creada con



AWT

### Swing

Es una biblioteca gráfica para Java que forma parte de las Java Foundation Classes (JFC). Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Swing es una plataforma independiente. Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:

- **Independencia de plataforma:** Swing es una plataforma independiente en ambos términos de su expresión (java) y de su implementación (no-nativa interpretación universal de widgets).
- **Extensibilidad:** Swing es una arquitectura altamente particionada que permite la utilización de diferentes pluggins en específicos interfaces de diferentes frameworks: Los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. En general, los usuarios de swing pueden extender el framework para: extender clases existentes (framework); proveyendo alternativas de implementación para elementos esenciales.
- **Orientado a componentes:** Swing es un framework basado en componentes. La diferencia entre objetos y componentes es un punto bastante sutil: concisamente, un componente es un objeto de buena conducta con un patrón conocido y especificado característico del comportamiento.
- **Customizable:** Dado el modelo de representación programático del framework de swing, el control permite representar diferentes 'look and feel' (desde MacOS look and feel hasta Windows XP look and feel). Más allá, los usuarios pueden proveer su propia implementación look and feel, que permitirá cambios uniformes en el look and feel existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.
- **Lightweight UI:** La magia de la flexibilidad de configuración de Swing, es también debido al hecho de que no utiliza los controles del GUI del OS nativo del host para la representación, pero usa parte de los apis 2D de Java.

Ventana creada con Swing



Es muy importante entender y asimilar el hecho de que Swing es una extensión del AWT, y no un sustituto encaminado a reemplazarlo. Aunque esto sea verdad en algunos casos en que los componentes de Swing se corresponden a componentes del AWT; por ejemplo, el JButton de Swing puede considerarse como un sustituto del Button del AWT.

### Ejemplo de uso de Swing

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HolaMundoSwing {
    public static void main(String[] args) {
        JFrame frame = new JFrame("HolaMundoSwing");
```

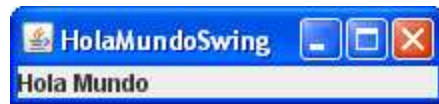
```
final JLabel label = new JLabel("Hola Mundo");
frame.getContentPane().add(label);

// listener para disparar el evento de cierre de ventana

frame.addWindowListener(new java.awt.event.WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});

frame.pack();
frame.setVisible(true);
}
```

Salida del Código



### Principales aspectos de una aplicación Swing

Como ya se dijo antes, cada aplicación Swing debe tener al menos un *top-level container* que contendrá toda la aplicación, estos pueden ser

- **javax.swing.JFrame**: Una ventana independiente.
- **javax.swing.JApplet**: Un applet.
- **Diálogos**: ventanas de interacción sencilla con el usuario como por ejemplo:
  - `java.swing.JOptionPane`: Ventana de diálogo tipo SI\_NO, SI\_NO\_CANCELAR, ACEPTAR, etc...
  - `java.swing.JFileChooser`: Ventana para elegir un archivo.
  - `java.swing.JColorChooser`
  - etc.

A un **contenedor** se le pueden agregar otros contenedores o componentes simples.

Tarea:

- 📖 Investigar la estructura de los paquetes AWT y Swing.



## Guía Práctica No 4 Esquema de una aplicación orientada a eventos.



### OBJETIVOS

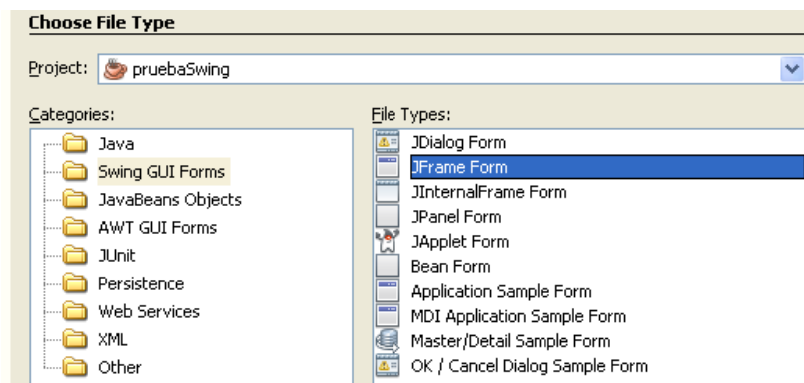
Al finalizar la Práctica, el estudiante será capaz de:

- Crear clases con componentes de interfase grafica swing.
- Crear aplicaciones para manipular datos con interfase grafica.

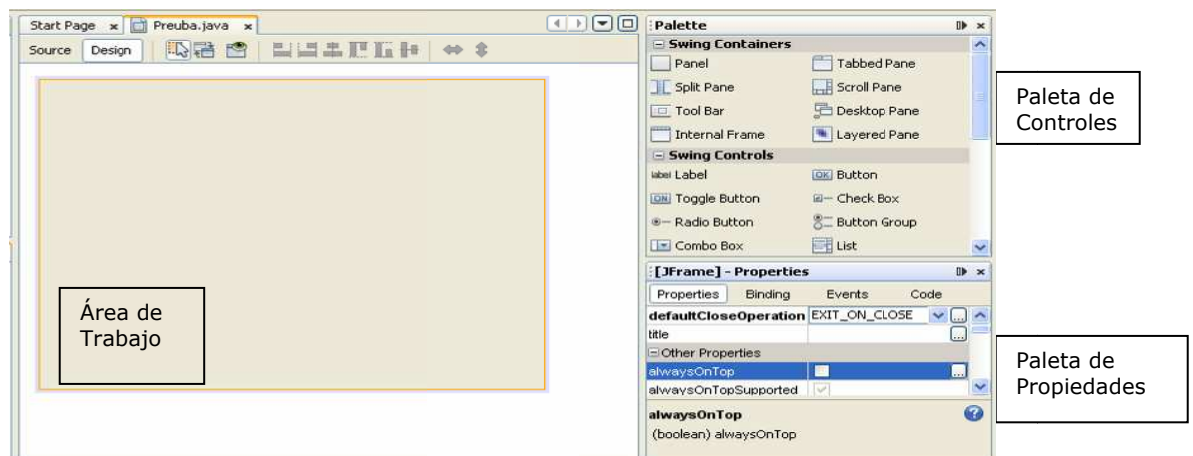
### PROCEDIMIENTO

Para crear interfaces graficas con Java NetBeans no necesitamos instalar ningun complemento adicional ya que el IDE cuenta con las herramientas necesarias para crear estos componentes. Lo que debemos tomar muy en cuenta es que no utilizaremos los archivos de clases comunes si no que debemos seleccionar específicamente los tipos de archivos de interfaces graficas que están incluidos en el IDE para ello realizaremos los siguientes pasos.

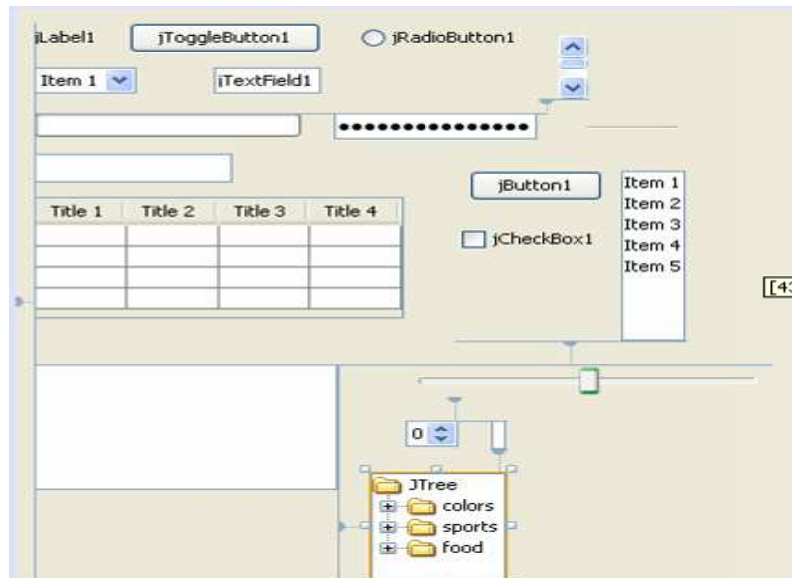
1. Agregar un nuevo archivo en su proyecto seleccionando la **categoría Swing GUI Forms** y el tipo de archivo **Jframe Form**, dar clic en siguiente.



2. Colocar el nombre de la clase y dar clic en finalizar aparecera una area de trabajo en donde podremos crear y manipular de forma visual los formularios y controles Swing.



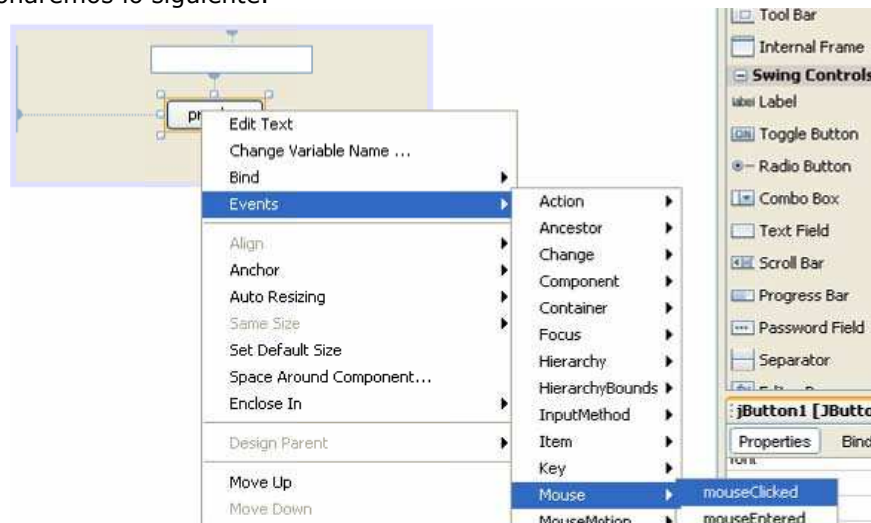
Ya creado el archivo lo único que nos resta es agregar los controles para ello el proceso que realizaremos es el arrastrar el control de la paleta hacia el área del formulario o dar clic sobre el control y ponerlo sobre el formulario.



### Ejemplos de uso de controles y eventos de un formulario.

#### Ejemplo1

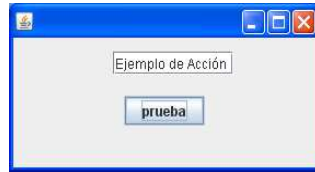
Creamos un formulario con un JTextField y un JButton y definimos una acción para el botón, la cual será mostrar un mensaje en la caja de texto. Para agregar eventos lo que haremos es dar clic derecho sobre el control al cual le queramos definir una acción y en el menú contextual seleccionaremos lo siguiente.



Al dar clic nos parecerá la ventana de codificación donde debemos agregar las instrucciones que se ejecutarán cuando demos clic sobre el botón en nuestro ejemplo serán las siguientes.

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
jTextField1.setText("Ejemplo de Acción");
}
```

Ahora probamos el resultado ejecutando la clase.



De esta forma podemos crear interfaces graficas y adicionar eventos .

#### Ejemplo2

Crearemos un formulario con 2 cajas de texto y crearemos una acción para que al teclear un texto en la caja 1 todo lo que tecleemos pase a la caja 2 el código para hacer lo siguiente.

```
private void jTextField1KeyTyped(java.awt.event.KeyEvent evt) {  
jTextField2.setText(jTextField1.getText());  
}
```

#### Ejercicios.

- Cree un formulario para calcular la suma de 2 cantidades (utilice las funciones valueOf para poder cambiar los tipos de datos.).
- Cree un formulario para la captura de datos de usuario(nombre, edad, estado civil, etc.) e imprímalos dentro de un control JLabel cuando se presione un botón en el mismo formulario.



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir Elementos para el desarrollo de interfaces.
- Definir el uso de métodos para el trabajo con interfaces graficas.
- Definir el empleo adecuado de salidas en los procesos de un programa.

### DESARROLLO

#### *Introducción.*

Para el desarrollo de Interfaces graficas es importante identificar las partes fundamentales que estas deberán poseer tanto de los controles y la disposición de los mismos, los eventos necesarios para su funcionamiento y como se comportan ya estando en funcionamiento.

Es importante entonces reconocer métodos (en el caso de los controles) para el acceso de datos y para compartirlos entre los mismos.

#### *Desarrollo de una interfase.*

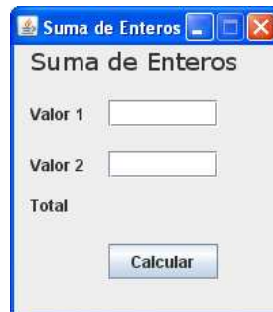
Lo primero que debemos conocer para la creación de una interfase es saber, cual es la necesidad o para que usos se crearía. Apliquemos un ejemplo básico una interfase para la suma de dos números enteros.

¿Que necesitamos para crear dicha interfase?

La respuesta a dicha pregunta es:

- Un formulario para contener los controles (JFrame).
- 2 cajas de texto (JTextField).
- Un botón (JButton).
- Y labels que se consideren necesarios (JLabel).

La interfase quedara de la siguiente forma.



Ahora tenemos que codificar el evento necesario y en el control adecuado, las acciones necesarias para que se puedan sumar los 2 valores. Tomamos en consideración aspectos necesarios para codificar de forma correcta y evitar errores a la hora de correr nuestra aplicación un aspecto del cual hablamos es el siguiente, los controles únicamente trabajan con caracteres no reconocen cantidades numéricas. Entonces para poder realizar la operación debemos de convertir los datos que se mandan con los controles usamos la función de acuerdo al tipo de datos a operar en el caso de nuestra aplicación son enteros entonces usamos la Clase **Integer** y su función **valueOf** la cual convierte los datos a números enteros, otro aspecto es el como obtendremos los datos de los controles y como enviaremos resultados hacia un control esto lo haremos con la función **getText** y **setText**, y ahora es necesario, convertir el total pero al igual

que al capturar los datos tenemos que convertir el valor por que el resultado devuelto es un tipo de dato numerico y los controles solo aceptan texto ¿Cómo haremos para convertirlo?.

El código es el siguiente es.

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
int total, val1, val2;
val1=Integer.valueOf(jTextField1.getText());
val2=Integer.valueOf(jTextField2.getText());
total=val1+val2;
jLabel5.setText(String.valueOf(total));
}
```

Ahora nos toca complementar nuestro código manejando excepciones y algunos errores que puedan ocasionarse agregamos el manejador de eventos y nuestro código final queda de la forma siguiente.

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
int total, val1, val2;
try{
val1=Integer.valueOf(jTextField1.getText());
val2=Integer.valueOf(jTextField2.getText());
total=val1+val2;
jLabel5.setForeground(new java.awt.Color(0, 0, 0));
jLabel5.setText(String.valueOf(total));
}catch(Exception e){
jLabel5.setForeground(new java.awt.Color(255, 0, 51));
jLabel5.setText("Error"+e.getMessage());
}
}
```

Donde se verifica si introduce algún valor que no corresponde al proceso para operar los datos y se enviara un mensaje el se muestra en un JLabel que es el elemento que procesara en este caso las salidas.

### Eventos

Cuando estamos trabajando GUI's el reconocer como se comporta nuestro código es importante por ello se debe de identificar claramente los eventos asociados a ello ya que Java al estar orientado a eventos no provee la facilidad de la creación o trabajo con los ya que no existe una instrucción o método que controle los eventos, se debe de haber creado desde cero. En el caso del uso de un IDE se nos facilita el trabajo ya que ya están creadas plantillas con los códigos para el uso de los eventos, donde solo debemos agregar las acciones (similar a la programación en VB.NET).

### Como se comporta un evento en java.

Un evento de Java se implementa de la siguiente forma:

- librería AWT (import java.awt.event).
- un listener (es el objeto que captura el evento, además son clases auxiliares).
- Un método asociado al listener para definir las instrucciones que se ejecutaran cuando dicho evento se desencadene.

Ejemplo Método que realiza "X" accion

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
jLabel1.setText("Hola Mundo");
}
```

Listener del control .

```
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
```

```

    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton1MouseClicked(evt);
    }
};

```

La parte de los desencadenadores de eventos o los listener es algo que se debe de tomar en cuenta ya que ellos son los que permiten la manipulación de todas las acciones que se puedan dar en una aplicación GUI, en Swing se genera un variado set de eventos, en la siguiente tabla se resumen los más comunes con sus respectivos "escuchadores".

<b>Ejemplos de eventos y sus escuchadores</b>	
<b><i>Acción que gatilla un evento</i></b>	<b>Tipo de escuchador</b>
<b><i>El usuario hace un click, presiona Return en un área de texto o selecciona un menú</i></b>	<b>ActionListener</b>
<b><i>El usuario escoge un frame (ventana principal)</i></b>	<b>WindowListener</b>
<b><i>El usuario hace un clic sobre una componente</i></b>	<b>MouseListener</b>
<b><i>El usuario pasa el mouse sobre una componente</i></b>	<b>MouseMotionListener</b>
<b><i>Una componente se hace visible</i></b>	<b>ComponentListener</b>
<b><i>Una componente adquiere el foco del teclado</i></b>	<b>FocusListener</b>
<b><i>Cambia la selección en una lista o tabla</i></b>	<b>ListSelectionListener</b>

*El modelo de creación de interfaces de java es un modelo un tanto complejo por la implementación de diferentes clases para la manipulación de los controles pero el resultado que se obtiene depende de planear previamente que elementos poseerá nuestra GUI además de los eventos importantes y que salidas manipulemos al tomar en cuenta estos 3 aspectos se nos facilitara en gran medida el proceso de desarrollo además de tener una herramienta adecuada para facilitar la codificación.*

#### **Tarea.**

- Investigar que funciones me permite cambiar el aspecto de un control (color de fondo, color de letra, tamaño etc.).
- Realice un boceto de creación de una GUI de Java, tomando en cuenta los controles que utilizara, los eventos necesarios, los métodos que se asociaran a los diferentes eventos y los datos que manipulara cada control.

## Guía Práctica No 5 Desarrollo de Interfaces.



### OBJETIVOS

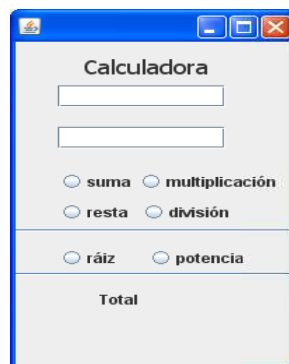
Al finalizar la Práctica, el estudiante será capaz de:

- Crear GUI en Java.
- Crear Métodos para la manipulación de datos de las GUI.

### PROCEDIMIENTO

Ya que conocemos las ventajas y características que NetBeans provee para la creación de GUI, nos facilitara la creación de una pequeña aplicación del cálculo de algunas operaciones matemáticas (operaciones aritméticas básicas, raíces cuadradas, potencias); lo primero es realizar un boceto o planeación previa de lo que necesitamos definiremos lo siguiente.

- 2 cajas de texto.
- Checkbox para cada operación.
- Labels necesarios.
- Un control Button Group.



Esta es el ejemplo del formulario con los controles ya posicionados, la lógica que definiremos para el formulario es que al dar clic sobre cualquiera de los botones de radio se realizara la operación definida tomando.

Entonces el código quedara de la siguiente forma.

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    val1=Double.valueOf(jTextField1.getText());
    val2=Double.valueOf(jTextField2.getText());
    total=val1+val2;
    jLabel3.setText(String.valueOf(total));
}

private void jButton3MouseClicked(java.awt.event.MouseEvent evt) {
    total=val1*val2;
    jLabel3.setText(String.valueOf(total));
}

private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
```

```

    total=val1-val2;
    jLabel3.setText(String.valueOf(total));
}

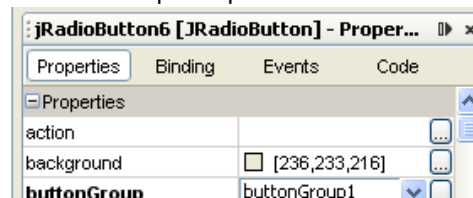
private void jButton4MouseClicked(java.awt.event.MouseEvent evt) {
    total=val1/val2;
    jLabel3.setText(String.valueOf(total));
}

private void jButton5MouseClicked(java.awt.event.MouseEvent evt) {
    total=Math.sqrt(val1);
    jLabel3.setText(String.valueOf(total));
}

private void jButton6MouseClicked(java.awt.event.MouseEvent evt) {
    total=Math.pow(val1,val2);
    jLabel3.setText(String.valueOf(total));
}
}

```

Hay que tomar algo muy en cuenta por que se usa un control radio group y por que no aparece nuestra interfase este control se utiliza para asociar todos los radio buton para que se activen o se desactiven al dar clic sobre ellos o sea para que dos radios no estén seleccionados a la vez.



Para asociar los radios a este control lo que haremos es asociar la propiedad buttonGroup en la paleta de propiedades para cada radio que tengamos en nuestro formulario. Con los códigos y ejemplos definidos podremos crear interfaces dependiendo de la complejidad debemos de ser mas cuidadosos a la hora de plantear que es lo que necesitaremos desarrollar.

#### Ejercicios.

- Con el ejemplo creado validar todas las entradas de las cajas de texto y enviar los mensajes necesarios (errores, excepciones)
- Modificar el ejemplo de la clase para operar por medio de botones.





## Clase N° 6 Introducción a la Tecnología JSP



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Conocer la Tecnología JSP, su uso y métodos de programación

### DESARROLLO

#### INTRODUCCION A JSP.

Java Server Pages (JSP) Es una tecnología similar a los Servlets que ofrece una conveniente forma de agregar contenido dinámico a un archivo HTML por utilizar código escrito en Java dentro del archivo utilizando tags especiales que son procesados por el servidor Web antes de enviarlos al cliente. La posibilidad de usar APIs de Java hacen de JSP una poderosa herramienta de desarrollo ya que se obtiene la ventaja de la programación orientada al objeto, como creación de clases especiales llamadas componentes o Java Beans, independencia de la plataforma propia de la programación en Java, etc.

La diferencia entre Servlets y JSP es que los Servlets son clases que deben implementar la clase abstracta `HttpServlet`, en especial el método `doGet()` o `doPost()` y deben ser previamente compilados, mientras que los archivos JSP contienen código Java entre código HTML utilizando los símbolos `<%` y `%>`. Por esto un archivo JSP debe ser interpretado por el servidor al momento de la petición por parte del usuario.

Un servidor Web para Servlets y JSP como Jakarta Tomcat es una aplicación escrita en Java que mantiene una Java Virtual Machine en ejecución para compilar los archivos JSP y ejecutar Servlets. El tiempo que demora en la compilación inicial de un JSP es contrarrestado por su rápido tiempo de respuesta posterior ya que para procesar un requerimiento sólo tiene que levantar un proceso liviano o thread dentro de la misma JVM para ejecutar un archivo `.class` y no crear un proceso pesado como un intérprete de Perl para programas CGI.

**Como se puede deducir** de esto, en realidad la tecnología JSP en sí no es nueva, si no que sólo es una forma distinta y más fácil para crear Servlets.

**Plantilla de Texto: HTML estático** En muchos casos, un gran porcentaje de nuestras páginas JSP consistirá en HTML estático, conocido como plantilla de texto. En casi todos los aspectos, este HTML se parece al HTML normal, sigue las mismas reglas de sintaxis, y simplemente "pasa a través" del cliente por el servlet creado para manejar la página. No sólo el aspecto del HTML es normal, puede ser creado con cualquier herramienta que usemos para generar páginas Web. Por ejemplo, podríamos utilizar Hometown de Allaire o Microsoft Frontpage.

La única excepción a la regla de que "la plantilla de texto se pasa tal y como es" es que, si queremos tener `"<%"` en la salida, necesitamos poner `"<\%"` en la plantilla de texto.

#### **ELEMENTOS DE SCRIPT**

JSP Los elementos de script nos permiten insertar código Java dentro del servlet que se generará desde la página JSP actual. Hay tres formas:

Expresiones de la forma `<%= expresión %>` que son evaluadas e insertadas en la salida.

Scriptlets de la forma `<% código %>` que se insertan

dentro del método `service` del servlet, y Declaraciones de la forma `<%! código %>` que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente.

#### **Expresiones JSP**

Una expresión JSP se usa para insertar valores Java directamente en la salida. Tiene la siguiente forma:

```
<%= expresión Java %>
```

La expresión Java es evaluada, convertida a un string, e insertada en la página. Esta evaluación se ejecuta durante la ejecución (cuando se solicita la página) y así tiene total acceso a la información sobre la solicitud. Por ejemplo, esto muestra la fecha y hora en que se solicitó la página:

```
Current time: <%= new java.util.Date() %>
```

Para simplificar estas expresiones, hay un gran número de variables predefinidas que podemos usar. Estos objetos implícitos se describen más adelante con más detalle, pero para el propósito de las expresiones, los más importantes son:

**1. request, el HttpServletRequest;**

**2. response, el HttpServletResponse;**

**3. session, el HttpSession asociado con el request (si existe), y**

**4. out, el PrintWriter (una versión con buffer del tipo JspWriter) usada para enviar la salida al cliente.**

Aquí tenemos un ejemplo:

```
Tu Servidor es: <%= request.getRemoteHost() %>
```

### Scriptlets JSP

Si queremos hacer algo más complejo que insertar una simple expresión, los scriptlets JSP nos permiten insertar código arbitrario dentro del método servlet que será construido al generar la página. Los Scriptlets tienen la siguiente forma:

```
<% Código Java %>
```

Los Scriptlets tienen acceso a las mismas variables predefinidas que las expresiones. Por eso, por ejemplo, si queremos que la salida aparezca en la página resultante, tenemos que usar la variable out:

```
<%
String queryData = request.getQueryString();
out.println("Datos Adjuntos al método GET: " + queryData);
%>
```

Observa que el código dentro de un scriptlet se insertará exactamente como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias print. Esto significa que los scriptlets no necesitan completar las sentencias Java, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets. Por ejemplo, el siguiente fragmento JSP, contiene una mezcla de texto y scriptlets:

```
<% if (Math.random() < 0.5) { %>
Tendrás un <B>Buen</B> día!
<% } else { %>
Tendrás un <B>Mal</B> día!
<% } %>
```

El ejemplo anterior se convertirá en algo como esto:

```
if (Math.random() < 0.5) {
out.println("Tendrás un <B>Buen</B> día!");
}
```

```
} else {  
out.println("Tendrás un <B>Mal</B> día!");  
}
```

### **Declaraciones JSP**

Una declaración JSP nos permite definir métodos o campos que serán insertados dentro del cuerpo principal de la clase servlet (fuera del método service que procesa la petición). Tienen la siguiente forma:

```
<%! Código Java%>
```

Como las declaraciones no generan ninguna salida, normalmente se usan en conjunción con expresiones JSP o escriptlets. Por ejemplo, aquí tenemos un fragmento de JSP que imprime el número de veces que se ha solicitado la página actual desde que el servidor se arrancó (o la clase del servlet se modificó o se recargó):

```
<%! private int accessCount = 0; %>  
Accesos a la Página desde que el Servidor Inició:  
<%= ++accessCount %>
```

### **Directivas JSP**

Una directiva JSP afecta a la estructura general de la clase servlet. Normalmente tienen la siguiente forma:

```
<%@ directive attribute="value" %>
```

Sin embargo, también podemos combinar múltiples selecciones de atributos para una sola directiva, de esta forma:

```
<%@ directive attribute1="value1"  
attribute2="value2"  
...  
attributeN="valueN" %>
```

Hay dos tipos principales de directivas: page, que nos permite hacer cosas como importar clases, personalizar la superclase del servlet, etc. include, que nos permite insertar un fichero dentro de la clase servlet en el momento que el fichero JSP es traducido a un servlet.

#### **La directiva page**

La directiva page nos permite definir uno o más de los siguientes atributos sensibles a las mayúsculas:

```
import="package.class" o import="package.class1,...,package.classN".
```

Esto nos permite especificar los paquetes que deberían ser importados. Por ejemplo:

```
<%@ page import="java.util.*" %>
```

El atributo import es el único que puede aparecer múltiples veces.

```
contentType="MIME-Type"
```

Esto especifica el tipo MIME de la salida. El valor por defecto es text/html. Por ejemplo, la directiva:

```
<%@ page contentType="text/plain" %>
```

tiene el mismo valor que el scriptlet

```
<% response.setContentType("text/plain"); %>
```

Ejemplo1.jsp

```
<HTML><HEAD><TITLE>Utilizando Java Server Pages</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE"
VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
<TR><TH CLASS="TITLE">
Utilizando Java Server Pages</TABLE>
</CENTER>
<P>
Algunos Contenidos Dinámicos utilizando mecanismos de JSP:
<UL>
<LI><B>Expresión.</B><BR>
Tu Servidor es: <%= request.getRemoteHost() %>.
<LI><B>Un Scriptlet.</B><BR>
<% out.println("Parámetros Adjuntos al Método GET: " +
request.getQueryString()); %>
<LI><B>Declaración.</B><BR>
<%! public int CUENTA = 0; %>
Número de Accesos a esta Página desde que inició Servidor: <%= ++CUENTA %>
<LI><B>Directiva.</B><BR>
<%@ page import = "java.util.*" %>
Current date: <%= new Date() %>
</UL>
</BODY>
</HTML>
```

### La directiva include JSP

Esta directiva nos permite incluir ficheros en el momento en que la página JSP es traducida a un servlet. La directiva se parece a esto:

```
<%@ include file = "url relativa" %>
```

La URL especificada normalmente se interpreta como relativa a la página JSP a la que se refiere, pero, al igual que las URLs relativas en general, podemos decirle al sistema que interpreta la URL relativa al directorio home del servidor Web empezando la URL con una barra invertida. Los contenidos del fichero incluido son analizados como texto normal JSP, y así pueden incluir HTML estático, elementos de script, directivas y acciones.

Por ejemplo, muchas sites incluyen una pequeña barra de navegación en cada página. Debido a los problemas con los marcos HTML, esto normalmente se implementa mediante una pequeña tabla que cruza la parte superior de la página o el lado izquierdo, con el HTML repetido para cada página de la site. La directiva include es una forma natural de hacer esto, ahorrando a los desarrolladores el mantenimiento engorroso de copiar realmente el HTML en cada fichero separado. Aquí tenemos un código representativo:

```
<HTML>
```

```

<HEAD>
<TITLE>Ejemplo de JavaServer Pages Utilizando Include</TITLE>
</HEAD>
<BODY>
<%@ include file="Otroejemplo.jsp" %>
</BODY>
</HTML>

```

### Variables Predefinidas

Para simplificar el código en expresiones y scriptlets JSP, tenemos ocho variables definidas automáticamente, algunas veces llamadas objetos implícitos. Las variables disponibles son: request, response, out, session, application, config, pageContext, y page. A continuación una descripción de aquellas más utilizadas:

- **request**

Este es el HttpServletRequest asociado con la petición, y nos permite mirar los parámetros de la petición (mediante getParameter), el tipo de petición (GET, POST, HEAD, etc.), y las cabeceras HTTP entrantes (cookies, Referer, etc.).

- **response**

Este es el HttpServletResponse asociado con la respuesta al cliente. Observa que, como el stream de salida (ver out más abajo) tiene un buffer, es legal seleccionar los códigos de estado y cabeceras de respuesta, aunque no está permitido en los servlets normales una vez que la salida ha sido enviada al cliente.

- **out**

Este es el PrintWriter usado para enviar la salida al cliente. Sin embargo, para poder hacer útil el objeto response (ver la sección anterior), esta es una versión con buffer de PrintWriter llamada JspWriter.

Ejemplo:

FormularioSimple.html

```

<HTML><head><title>Ejemplo Utilizando JSP</title></head>
<BODY><CENTER> </CENTER>
<h1 align="center">Ejemplo Utilizando JSP</h1>
<h3 align="center">Programación IV</h3>
<hr>
<p>
<H3>Por Favor, Introduzca la siguiente Información</H3>
<FORM action="RecuperaDatos.jsp" method="get">
Nombre y Apellido: <INPUT type="text" name="Nombre" size="20">
<INPUT type="text" name="Apellido" size="20"><BR>
Sexo: <INPUT type="radio" checked name="sexo" value="Masculino">Masculino
<INPUT type="radio" name="sexo" value="Femenino">Femenino
<INPUT type="radio" name="sexo" value="Alienígena">Alienígena<BR>
<P>
¿Cual es tu lenguaje de Programación favorito?:
<SELECT name="Lenguaje">
<option>Visual Basic</option>
<option>Visual FoxPro</option>
<option>Visual C</option>
<option>Delphi</option>
<option>Java</option>
<option>Power Builder</option>

```

```
<option>Otro</option>
</SELECT>
```

Los datos del formulario anterior, son enviados al siguiente archivo jsp:

RecuperaDatos.jsp

```
<HTML><HTML><head><title>Resultado del Ejemplo JSP</title></head>
<BODY>
<CENTER> </CENTER>
<h1 align="center">Resultado del Ejemplo JSP</h1>
<hr>
<%
// Recuperando las variables del formulario
String Nombre = request.getParameter("Nombre");
String Apellido = request.getParameter("Apellido");
String sexo = request.getParameter("sexo");
String Lenguaje = request.getParameter("Lenguaje");
%>
<%-- Imprimiendo las variables --%>
<H2>Saludos, <%=Nombre%> <%=Apellido%>!!</H2>
Tu Sexo es <i><%=sexo%></i>. Tu lenguaje de Programación Preferido es:
<b><%=Lenguaje%></b>, Excelente Elección.
</BODY></HTML>
```



## Guía Práctica No 6 Introducción a Java Server Pages



### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Utilizar el Lenguaje de Java Servlets para crear Java Server Pages.
- Verificar las ventajas que ofrece JSP sobre los lenguajes de programación orientados al Web, incluyendo los mismos Servlets.
- Utilizar Expresiones de Java en los llamados Scriptlets.
- Utilizar Directivas de JSP en las aplicaciones para el Web.

### PROCEDIMIENTO

#### ¿Qué es JSP?

Java Server Pages (JSP) es una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente. Muchas páginas Web que están construidas con programas CGI son casi estáticas, con la parte dinámica limitada a muy pocas localizaciones. Pero muchas variaciones CGI, incluyendo los servlets, hacen que generemos la página completa mediante nuestro programa, incluso aunque la mayoría de ella sea siempre lo mismo. JSP nos permite crear dos partes de forma separada. Aquí tenemos un ejemplo:

EjemploSencillo.jsp

```
<HTML><head><title>JSP mis inicios</title><head>
<h1>Ejemplo Sencillo de JSP <h1><hr><BODY><P>
<%
// Este es un comentario
out.println("<MARQUEE>FINALIZANDO LAS CLASES DE PROGRAMACION IV, "+
"ESTA MARQUESINA ESTA HECHA CON JSP</MARQUEE>");
%></BODY></HTML>
```

¿Cuáles son las Ventajas de JSP?

#### • **Contra Active Server Pages (ASP).**

ASP es una tecnología similar de Microsoft. Las ventajas de JSP estan duplicadas. Primero, la parte dinámica está escrita en Java, no en Visual Basic, otro lenguaje específico de MS, por eso es mucho más poderosa y fácil de usar. Segundo, es portable a otros sistemas operativos y servidores Web

#### • **Contra los Servlets.**

JSP no nos da nada que no pudiéramos en principio hacer con un servlet. Pero es mucho más conveniente escribir (y modificar!) HTML normal que tener que hacer un billón de sentencias println que generen HTML. Además, separando el formato del contenido podemos poner diferentes personas en diferentes tareas: los expertos en diseño de páginas Web pueden construir el HTML, dejando espacio para que los programadores de servlets inserten el contenido dinámico.

#### • **Contra Server-Side Includes (SSI).**

SSI es una tecnología ampliamente soportada que incluye piezas definidas externamente dentro de una página Web estática. JSP es mejor

porque nos permite usar servlets en vez de un programa separado para generar las partes dinámicas. Además, SSI, realmente está diseñado para inclusiones sencillas, no para programas "reales" que usen formularios de datos, hagan conexiones a bases de datos, etc.

- **Contra JavaScript.**

JavaScript puede generar HTML dinámicamente en el cliente. Esta es una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente. Con la excepción de las cookies, el HTTP y el envío de formularios no están disponibles con JavaScript. Y, como se ejecuta en el cliente, JavaScript no puede acceder a los recursos en el lado del servidor, como bases de datos, catálogos, información de precios, etc.

Normalmente daremos a nuestro fichero una extensión .jsp, y normalmente lo instalaremos en el mismo sitio que una página Web normal.

Aunque lo que escribamos frecuentemente se parezca a un fichero HTML normal en vez de un servlet, detrás de la escena, la página JSP se convierte en un servlet normal, donde el HTML estático simplemente se imprime en el stream de salida estándar asociado con el método service del servlet. Esto normalmente sólo se hace la primera vez que se solicita la página, y los desarrolladores pueden solicitar la página ellos mismos cuando la instalan si quieren estar seguros de que el primer usuario real no tenga un retardo momentáneo cuando la página JSP sea traducida a un servlet y el servlet sea compilado y cargado. Observa también, que muchos servidores Web nos permiten definir alias para que una URL que parece apuntar a un fichero HTML realmente apunte a un servlet o a una página JSP.

Además del HTML normal, hay tres tipos de construcciones JSP que embeberemos en una página: elementos de script, directivas y acciones.

Los elementos de script nos permiten especificar código Java que se convertirá en parte del servlet resultante, las directivas nos permiten controlar la estructura general del servlet, y las acciones nos permiten especificar componentes que deberían ser usados, y de otro modo controlar el comportamiento del motor JSP. Para simplificar los elementos de script, tenemos acceso a un número de variables predefinidas como request, response y out. Ejemplo:

Fomulario.html

```
<HTML>
<BODY bgcolor="#B9E3FF">
<H1>Por favor, Introduzca un listado de Nombre</H1>
<FORM action="MultiParametros.jsp" method="get">
<INPUT type="text" name="nonmbre" size="20"><BR>
<INPUT type="text" name="nombres" size="20"><BR>
<INPUT type="text" name="nombres" size="20"><BR>
<INPUT type="text" name="nombres" size="20"><BR>
<INPUT type="text" name="nombres" size="20"><p><BR>
<INPUT type="submit"> </p>
</FORM>
</BODY>
</HTML>
MultiParametros.jsp
<HTML>
<BODY bgcolor="#B9E3FF">
<font face=verdana color=blue>Los nombres introducidos son: </font>
<hr>
<PRE>
<%
// Obteniendo los valores de los nombres
String arraynombres[] = request.getParameterValues("nombres");
out.println("<Lo>");
```



```

for (int i=0; i < arraynombres.length; i++)
{
out.println("<Li><i>" + arraynombres[i]);
}
out.println("</Lo>");
%>
</PRE>
</BODY>
</HTML>

```

El siguiente archivo .jsp Crea un formulario y se auto-envía los parámetros que son evaluados por el mismo archivo jsp, ya que algunos de estos son obligatorios que el usuario los introduzca.

#### CamposRequeridos.jsp

```

<HTML>
<BODY>
<%
    String Nombre = request.getParameter("nombre");
    if (Nombre == null) Nombre = "";
    String Apellido = request.getParameter("apellido");
    if (Apellido == null) Apellido = "";
    String Direccion = request.getParameter("direccion");
    if (Direccion == null) Direccion = "";
    String Ciudad = request.getParameter("ciudad");
    if (Ciudad == null) Ciudad = "";
    String Departamento = request.getParameter("departamento");
    if (Departamento == null) Departamento = "";
    String Telefono = request.getParameter("telefono");
    if (Telefono == null) Telefono = "";
    String formatOption = request.getParameter("formatoption");
    if (formatOption == null) formatOption = "";
    // Algunos de los parametros son requeridos,Damos un estilo
    // por defecto para estos datos"requeridos"
    String NombreColorRequerido = "black";
    String ApellidoColorRequerido = "black";
    String TelefonoColorRequerido = "black";
    String ColorNotificarRequerido = "red";
    // Cuando esta pagina es ejecutada, Realiza una petición HTTP GET

    // Pero en la opción METHOD del tag FORM, El formulario
    // Envía los datos por HTTP POST.
    // Cuando el boton submit es presionado...
    if (request.getMethod().equals("POST"))
    {
        boolean CamposRequeridosPresentes = true;
        // Verificamos si los campos requeridos estan en blanco
        if (Nombre.length() == 0)
        {
            NombreColorRequerido = ColorNotificarRequerido;
            CamposRequeridosPresentes = false;
        }
        if (Apellido.length() == 0)
        {
            ApellidoColorRequerido = ColorNotificarRequerido;
            CamposRequeridosPresentes = false;
        }
        if (Telefono.length() == 0)
        {

```

```

TelefonoColorRequerido = ColorNotificarRequerido;
CamposRequeridosPresentes = false;
}
// Si el usuario no introdujo los campos requeridos, digamosle que estan
// marcados en un color diferente.
if (!CamposRequeridosPresentes)
{
    %>
    Usted No ha introducido todos los campos requeridos.<br> Debe introducir
    todos los campos que estan marcados en color <b><font color="<%=ColorNotificarRequerido%>">
    Rojo</font></b>.
    <%
    }
    else
    {
        // Desplegar el nombre y Dirección que ha sido introducida
        String nameString = Nombre+" "+Apellido+
        "<BR>"+ Direccion+"<BR>"+Ciudad+", "+
        Departamento+".<br> Teléfono: "+Telefono;
        out.println("El Registro Actual es:<P>");
        if (formatOption.equals("Negrita"))
        {
            out.println("<B>"+nameString+"</B>");
        }
        else if (formatOption.equals("Cursiva"))
        {
            out.println("<I>"+nameString+"</I>");
        }
        else
        {
            out.println(nameString);
        }
        out.println("<P>");
    }
    %>
    <FORM action="CamposRequeridos.jsp" method="POST">
    <TABLE>
    <TR><TD>Nombre:<TD><INPUT type="text" name="nombre" value="<%=Nombre%>">
    <TD><FONT color="<%=NombreColorRequerido%>">requerido</FONT>
    <TR><TD>Apellido:<TD><INPUT type="text" name="apellido" value="<%=Apellido%>">
    <TD><FONT color="<%=ApellidoColorRequerido%>">requerido</FONT>
    <TR><TD>Dirección:<TD><textarea rows="4" name="direccion" cols="26"><%=Direccion%></textarea>
    <TR><TD>Ciudad<TD><INPUT type="text" name="ciudad" value="<%=Ciudad%>">
    <TR><TD>Departamento:<TD><INPUT type="text" name="departamento"
    value="<%=Departamento%>">
    <TR><TD>Teléfono de Contacto:<TD><INPUT type="text" name="telefono" value="<%=Telefono%>">
    <TD><FONT color="<%=TelefonoColorRequerido%>">requerido</FONT>
    </TABLE>
    <P>
    Opciones de Formato:<BR>
    <SELECT name="formatoption">
    <OPTION value="Normal">Normal</OPTION>
    <OPTION value="Negrita">Negrita</OPTION>
    <OPTION value="Cursiva">Cursiva</OPTION>
    </SELECT>
    <P>
    <INPUT type="submit" value="Hacer Clic Aquí">
    </FORM>

```

```
</BODY>  
</HTML>
```

## DIRECTIVAS JSP

### ¿Qué es la Directiva Include?

La directiva include se usa para insertar un fichero dentro de una página JSP cuando se compila la página JSP. El texto del fichero incluido se añade a la página.

### ¿Qué clases de ficheros se pueden incluir?

El fichero incluido puede ser un fichero JSP, un fichero HTML, o un fichero de texto. También ser un fichero de código escrito en lenguaje Java.

Hay que ser cuidadoso en que el fichero incluido no contenga las etiquetas <html>, </html>, <body>, or </body>. Porque como todo el contenido del fichero incluido se añade en esa localización del fichero JSP, estas etiquetas podrían entrar en conflicto con las etiquetas similares del fichero JSP.

### Incluir Ficheros JSP

Si el fichero incluido es un fichero JSP, las etiquetas JSP son analizadas y sus resultados se incluyen (junto con cualquier otro texto) en el fichero JSP.

Sólo podemos incluir ficheros estáticos. Esto significa que el resultado analizado del fichero incluido se añade al fichero JSP justo donde está situada la directiva. Una vez que el fichero incluido es analizado y añadido, el proceso continúa con la siguiente línea del fichero JSP llamante.

### ¿Qué es un fichero Estático?

Un include estático significa que el texto del fichero incluido se añade al fichero JSP. Además en conjunción con otra etiqueta JSP, <jsp:include>: podemos incluir ficheros estáticos o dinámicos:

- Un fichero estático es analizado y si contenido se incluye en la página JSP llamante.
- Un fichero dinámico actúa sobre la solicitud y envía de vuelta un resultado que es incluido en la página JSP.

### ¿Cuál es la Sintaxis para Incluir un Fichero?

Podemos incluir un fichero en la localización específica del fichero JSP usando la directiva include con la siguiente sintaxis:

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

Aquí la URL puede ser una URL relativa indicando la posición del fichero a incluir dentro del servidor.

### Acción jsp:include

Esta acción nos permite insertar ficheros en una página que está siendo generada. La sintaxis se parece a esto:

```
<jsp:include page="relative URL" flush="true" />
```

Al contrario que la directiva include, que inserta el fichero en el momento de la conversión de la página JSP a un Servlet, esta acción inserta el fichero en el momento en que la página es solicitada. Esto se paga un poco en la eficiencia, e imposibilita a la página incluida de contener código JSP general (no puede seleccionar cabeceras HTTP, por ejemplo), pero se obtiene una significativa flexibilidad. Por ejemplo, aquí tenemos una página JSP que inserta cuatro puntos diferentes dentro de una página Web

"Noticias de Ultima Hora?". Cada vez que cambian las líneas de cabeceras, los autores sólo tienen que actualizar los cuatro ficheros, pero pueden dejar como estaba la página JSP principal.

Noticias.jsp

```
<HTML>
<HEAD>
<TITLE>Noticias Frescas</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE"
VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
<TR><TH CLASS="TITLE">
Ultimas Noticias en t JspNoticias.com</TABLE>
</CENTER>
<P>
Este es un Resumen de las Noticias Más Recientes:
<OL>
<LI><jsp:include page="noticias/Item1.html" flush="true"/>
<LI><jsp:include page="noticias/Item2.html" flush="true"/>
<LI><jsp:include page="noticias/Item3.html" flush="true"/>
<LI><jsp:include page="noticias/Item4.html" flush="true"/>
</OL>
</BODY>
</HTML>
```

Para que este archivo pueda ejecutarse, debe crear en un directorio virtual llamando "noticias" los archivos item1, item2, item3, item4 con extensión HTML o puede acceder a cualquier archivo de texto cambiando el valor del atributo page de Noticias.jsp.

### EJERCICIOS PROPUESTOS

1. Crear un formulario que pida Nombre, Apellido, Dirección, Teléfono, Dirección de Email, Fecha de Nacimiento, Pasatiempos Favoritos. Estos serán leídos por un archivo JSP, que desplegará los parámetros recibidos (utilizar método POST).

2. Crear una pagina JSP, donde puedas incluir en una tabla, el contenido de cuatro archivos HTML, uno en cada celda, como se muestra en la figura.

Archivo1.html	Archivo2.html
Archivo3.html	Archivo4.html

3. Realizar el ejercicio #2 de la guía practica 8 utilizando JSP, el enunciado es el siguiente:

Crear un formulario en HTML que simule la pantalla de acceso a una aplicación Web con acceso restringido, el formulario pedirá al usuario: Su nombre de Usuario y Contraseña (el formulario debe estar validado de tal forma que obligue al usuario a escribir los datos requeridos). Estos datos serán enviados a una pagina JSP que validará la entrada a la aplicación. Esta contendrá 2 arreglos uno de Usuarios y otro de Contraseñas, de tal forma que Usuario[2] poseerá su clave en contraseña[2]. Si el usuario y contraseña son validos la página mostrará un mensaje de bienvenida al usuario, de lo contrario desplegará un mensaje de Usuario y/o contraseña no validos.

Tomar como datos de los arreglos:

- USUARIO {Administrador, Usuario1, Usu02, Operador}
- PASSWORD {admin0101, nimodo, clave02, ok}



## Clase N° 7 JSP con bases de datos



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Conocer aspectos de Seguridad y conexión de BD para Internet

### DESARROLLO

#### JSP CON BASES DE DATOS.

##### *Mover el Cursor en una Hoja de Resultados*

Una de las nuevas características del API JDBC 2.0 es la habilidad de mover el cursor en una hoja de resultados tanto hacia atrás como hacia adelante. También hay métodos que nos permiten mover el cursor a una fila particular y comprobar la posición del cursor. La hoja de resultados Scrollable hace posible crear una herramienta GUI (Interface Gráfico de Usuario) para navegar a través de ella, lo que probablemente será uno de los principales usos de esta característica. Otro uso será movernos a una fila para actualizarla.

Antes de poder aprovechar estas ventajas, necesitamos crear un objeto **ResultSet** Scrollable:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT NOM_PROD, PRECIO FROM PRODUCTOS");
```

Este código es similar al utilizado anteriormente, excepto en que añade dos argumentos al método **createStatement**. El primer argumento es una de las tres constantes añadidas al API **ResultSet** para indicar el tipo de un objeto **ResultSet**: **TYPE\_FORWARD\_ONLY**, **TYPE\_SCROLL\_INSENSITIVE**, y **TYPE\_SCROLL\_SENSITIVE**. El segundo argumento es una de las dos constantes de **ResultSet** para especificar si la hoja de resultados es de sólo lectura o actualizable: **CONCUR\_READ\_ONLY** y **CONCUR\_UPDATABLE**. Lo que debemos recordar aquí es que si especificamos un tipo, también debemos especificar si es de sólo lectura o actualizable. También, debemos especificar primero el tipo, y como ambos parámetros son **int**, el compilador no comprobará si los hemos intercambiado.

Especificando la constante **TYPE\_FORWARD\_ONLY** se crea una hoja de resultados no desplazable, es decir, una hoja en la que el cursor sólo se mueve hacia adelante. Si no se especifican constantes para el tipo y actualización de un objeto **ResultSet**, obtendremos automáticamente una **TYPE\_FORWARD\_ONLY** y **CONCUR\_READ\_ONLY**.

Obtendremos un objeto **ResultSet** desplazable si utilizamos una de estas constantes: **TYPE\_SCROLL\_INSENSITIVE** o **TYPE\_SCROLL\_SENSITIVE**. La diferencia entre estas dos es si la hoja de resultados refleja los cambios que se han hecho mientras estaba abierta y si se puede llamar a ciertos métodos para detectar estos cambios. Generalmente hablando, una hoja de resultados **TYPE\_SCROLL\_INSENSITIVE** no refleja los cambios hechos mientras estaba abierta y en una hoja **TYPE\_SCROLL\_SENSITIVE** si se reflejan. Los tres tipos de hojas de resultados harán visibles los resultados si se cierran y se vuelve a abrir. En este momento, no necesitamos preocuparnos de los puntos delicados de las capacidades de un objeto **ResultSet**, entraremos en más detalle más adelante. Aunque deberíamos tener en mente el hecho de que no importa el tipo de hoja de resultados que especifiquemos, siempre estaremos limitados por nuestro controlador de base de datos y el driver utilizados.

Una vez que tengamos un objeto **ResultSet** desplazable, **srs** en el ejemplo anterior, podemos utilizarlo para mover el cursor sobre la hoja de resultados. Recuerda que cuando creábamos un objeto **ResultSet** anteriormente, tenía el cursor posicionado antes de la primera fila. Incluso aunque una hoja de resultados se seleccione desplazable, el cursor también se posiciona inicialmente delante de la primera fila. En el API JDBC 1.0, la única forma de mover el cursor era llamar al método **next**. Este método todavía es apropiado si queremos acceder a las filas una a una, yendo de la primera fila a la última, pero ahora tenemos muchas más formas para mover el cursor.

La contrapartida del método **next**, que mueve el cursor una fila hacia delante (hacia el final de la hoja de resultados), es el nuevo método **previous**, que mueve el cursor una fila hacia atrás (hacia el inicio de la hoja de resultados). Ambos métodos devuelven **false** cuando el cursor se sale de la hoja de resultados (posición antes de la primera o después de la última fila), lo que hace posible utilizarlos en un bucle **while**. Ya hemos utilizado un método **next** en un bucle **while**, pero para refrescar la memoria, aquí tenemos un ejemplo que mueve el cursor a la primera fila y luego a la siguiente cada vez que pasa por el bucle **while**. El bucle termina cuando alcanza la última fila, haciendo que el método **next** devuelva **false**. El siguiente fragmento de código imprime los valores de cada fila de **srs**, con cinco espacios en blanco entre el nombre y el precio:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT NOM_PROD, PRECIO FROM PRODUCTOS");
while (srs.next())
{
String name = srs.getString("NOM_PROD ");
float price = srs.getFloat("PRECIO");
System.out.println(name + " " + price);
}
```

Al igual que en el fragmento anterior, podemos procesar todas las filas de **srs** hacia atrás, pero para hacer esto, el cursor debe estar detrás de la última fila. Se puede mover el cursor explícitamente a esa posición con el método **afterLast**. Luego el método **previous** mueve el cursor desde la posición detrás de la última fila a la última fila, y luego a la fila anterior en cada interacción del bucle **while**. El bucle termina cuando el cursor alcanza la posición anterior a la primera fila, cuando el método **previous** devuelve **false**.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet srs = stmt.executeQuery("SELECT NOM_PROD, PRECIO FROM PRODUCTOS");
srs.afterLast();
while (srs.previous())
{
String name = srs.getString("NOM_PROD ");
float price = srs.getFloat("PRECIO");
System.out.println(name + " " + price);
}
```

Se puede mover el cursor a una fila particular en un objeto **ResultSet**. Los métodos **first**, **last**, **beforeFirst**, y **afterLast** mueven el cursor a la fila indicada en sus nombres. El método **absolute** moverá el cursor al número de fila indicado en su argumento. Si el número es positivo, el cursor se mueve al número dado desde el principio, por eso llamar a **absolute(1)** pone el cursor en la primera fila. Si el número es negativo, mueve el cursor al número dado desde el final, por eso llamar a **absolute(-1)** pone el cursor en la última fila. La siguiente línea de código mueve el cursor a la cuarta fila de **srs**:

```
srs.absolute(4);
```

Si **srs** tuviera 500 filas, la siguiente línea de código movería el cursor a la fila 497:

```
srs.absolute(-4);
```

Tres métodos mueven el cursor a una posición relativa a su posición actual. Como hemos podido ver, el método **next** mueve el cursor a la fila siguiente, y el método **previous** lo mueve a la fila anterior. Con el método **relative**, se puede especificar cuántas filas se moverá desde la fila actual y también la dirección en la que se moverá. Un número positivo mueve el cursor hacia adelante el número de filas dado; un número negativo mueve el cursor hacia atrás el número de filas dado. Por ejemplo, en el siguiente fragmento de código, el cursor se mueve a la cuarta fila, luego a la primera y por último a la tercera:

```
srs.absolute(4); // cursor está en la cuarta fila
...
srs.relative(-3); // cursor está en la primera fila
...
srs.relative(2); // cursor está en la tercera fila
```

El método **getRow** permite comprobar el número de fila donde está el cursor. Por ejemplo, se puede utilizar **getRow** para verificar la posición actual del cursor en el ejemplo anterior:

```
srs.absolute(4);
int rowNum = srs.getRow(); // rowNum debería ser 4
srs.relative(-3);
int rowNum = srs.getRow(); // rowNum debería ser 1
srs.relative(2);
int rowNum = srs.getRow(); // rowNum debería ser 3
```

Existen cuatro métodos adicionales que permiten verificar si el cursor se encuentra en una posición particular. La posición se indica en sus nombres: **isFirst**, **isLast**, **isBeforeFirst**, **isAfterLast**. Todos estos métodos devuelven un **boolean** y por lo tanto pueden ser utilizados en una sentencia condicional. Por ejemplo, el siguiente fragmento de código comprueba si el cursor está después de la última fila antes de llamar al método **previous** en un bucle **while**. Si el método **isAfterLast** devuelve **false**, el cursor no estará después de la última fila, por eso se llama al método **afterLast**. Esto garantiza que el cursor estará después de la última fila antes de utilizar el método **previous** en el bucle **while** para cubrir todas las filas de **srs**.

```
if (srs.isAfterLast() == false) {
    srs.afterLast();
}
while (srs.previous()) {
    String name = srs.getString("NOM_PROD");
    float price = srs.getFloat("PRECIO");
    System.out.println(name + " " + price);
}
```

### **Ejemplo de Java Server Pages Con Bases de Datos.**

Este es también un caso común con elementos de una tabla, sin embargo es también fácil de resolver.

Es necesario recordar primero algunas cosas elementales:

1) Recordar que el número de columna en una tabla empieza en 1, esto es que para realizar alguna operación por ejemplo la columna edad del ejemplo siguiente, su número de columna es la número 3.

2) La operación que se plantea se puede realizar con todos los renglones de la tabla o con un solo renglón de la tabla(del resultset).

3) En el ejemplo se realiza la operación con todos los renglones de la tabla y no olvidar que se tiene que usar la instrucción SQL Update para que la nueva información se actualice en disco, recordar que los cambios que se hacen a la tabla, es realmente al resultset, que a su vez es una tabla o base de datos en la memoria de la maquina del cliente o usuario, y estos cambios hay que actualizarlos o pasarlos o UPDATE a la base de datos en disco.

El siguiente programa le aumenta 5 a todas las edades.

### EjemploGuia13.jsp

```
<html>
<head>
<title>Ejemplo JSP con BD</title>
</head>
<body bgcolor="#F0F0FF">
<p align="center"><b><font face="Verdana" size="4">Ejemplo de Conexión con bases
de Datos.</font></b></p>
<hr>
<p align="center">&nbsp;</p>
<p></p>
<p></p>
<center><table border="1" cellpadding="0" cellspacing="0" bordercolor="#111111"
width="62%">
<tr>
<td width="100%" align="center" height="100">
<font face="Verdana" color="#800000"><p align="center"><b><u>Los datos de la tabla se
han
actualizado.</u></b></font></p>
</td>
</tr>
</table>
</center>
<%@ page import="java.sql.*" %>
<%
int edad, clave;
String q,nombre;
Connection canal = null;
ResultSet tabla= null;
Statement instruccion=null;
String sitiobase = "c:/inetpub/wwwroot/llevar/base/mibase.mdb";
String strcon= "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=" + sitiobase;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
canal=DriverManager.getConnection(strcon);
instruccion = canal.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
}
catch(SQLException e)
{
out.println("EXCEPCION : "+e.getMessage());
};
try
{
tabla = instruccion.executeQuery("select * from mitabla");
tabla.last();
int ultimo=tabla.getInt(1);
```



```

tabla.first();%>
<font face='Verdana' color='#800000'><p>Los Datos de la Tabla son:</p>
<center><table border="1" cellpadding="0" cellspacing="0" bordercolor="#111111"
width="62%">
<th> CLAVE <th> NOMBRE <th> EDAD
<%
for(int r=1; r<=ultimo; r++)
{
clave = tabla.getInt(1);
nombre = tabla.getString(2);
edad =tabla.getInt(3);
edad=edad+5;
q="update mitabla set edad= "+edad+ " where clave = "+clave+"";
instruccion.executeUpdate(q);
%>
<tr><td align=center> <%=clave%><td> <%=nombre%><td align=center> <%=edad%>
<%
clave=clave+1;
tabla = instruccion.executeQuery("select * from mitabla");
tabla.absolute(clave);
}; %>
</table>
</center>
<%
}
finally
{
try
{
if (tabla != null)
{
tabla.close();
}
if (instruccion != null)
{
instruccion.close();
}
if (canal != null)
{
canal.close();
}
}
catch (Exception e)
{
out.println("EXCEPCION : "+e.getMessage());
}
}
%>
</body>
</html>

```



## Guía Práctica No 7 JSP con Bases de Datos



### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Utilizar distintos formatos para desplegar la fecha del sistema.
- Realizar Consultas a Bases de Datos por medio de SQL, utilizando JSP.
- Utilizar diferentes tipos de conexión ODBC con bases de Datos.
- Crear acceso a Aplicaciones Web con JSP.

### PROCEDIMIENTO

#### La clase Date.

En ocasiones, los programas necesitan trabajar con la fecha y hora en curso. La clase para manejo de fechas en Java, *java.util.date*, proporciona métodos para representar y manipular fechas y horas. Si un programa llama al constructor de la clase *date* sin parámetros, se crea una instancia que se inicializa con la fecha y hora en curso.

Como las fechas se representan de diferentes formas en distintas partes del mundo, la clase Date proporciona un método que da formato a la fecha en el estilo local. El método *toLocaleString* (a cadena local) toma la información de la fecha de la instancia y crea un objeto String. Por ejemplo las siguientes instrucciones de JSP utilizan los métodos de la clase Date para desplegar la fecha y hora del sistema.

#### Fecha.jsp

```
<HTML>
hola hoy es:<BR>
<%= new java.util.Date().toLocaleString() %>
</HTML>
```

El siguiente ejemplo JSP, muestra el encabezado de una aplicación Web incluyendo la fecha del sistema en formato largo. Para ejecutarlo debe incluir en la URL el parámetro NOMBRE, por ejemplo:

**<http://localhost/MiAplicación/FechaLarga.jsp?NOMBRE=JOSE PEREZ>.**  
**FechaLarga.jsp**

```
<%@page import="java.util.*,java.text.*"%>

<%DateFormat dtLong = DateFormat.getDateInstance(
DateFormat.LONG, DateFormat.LONG); //Obteniendo Formato de Fecha
%>
<html>
<head>
<base target=contents>
<title>Encabezado de la Aplicación</title>
</head>
<body rightMargin=2 leftMargin=2 topMargin=2 text=#FFFFFF>
<table border=0 cellpadding=0 cellspacing=0 width=100% height=65>
<tr align= center>
<td bgcolor=#006699 height=58><p align=left>
<img src='../Imagenes/Logo.gif' width=69 height=82></td>
```

```

<td height=78 bgcolor=#006699>
<font face='verdana' size=4>
<b> SISTEMA DE CONTROL DE ACCESOS </font>
<br><font face='verdana' size='2'>
BIENVENID@ <%= request.getParameter("NOMBRE")%>
</b>
</font>
</td>
</tr>
<tr>
<td valign='top' align='right' colspan=2 bgcolor=#000080><b><font face=Verdana size=1
color=#ffffff>
<%
//Obteniendo La fecha Actual, supriendo la Hora ***
String Fecha = dtLong.format(new Date()).toString();
int Contar = Fecha.indexOf(":");// Buscando los 2 puntos de la hora
Fecha = Fecha.substring(0,Contar-3); //recuperando solo la fecha en formato largo
out.println(Fecha); //Imprimiendo Fecha Actual ***
%>
</font></b></td>
</tr>
</table>
</body>
</html>

```

## JDBC SQL RESULTSET

El modelo de datos de java descansa en una serie de objetos especializados que facilitan el procesamiento de una base de datos.

- El problema es comunicar un programa o aplicación con una base de datos y más que comunicar se pretende que el programa o aplicación realice una serie de procesos u operaciones con la base de datos o mejor aun con el conjunto de tablas que contiene una base de datos.

- La primera nota a recordar es que una base de datos puede estar físicamente en el servidor y en algún folder o directorio del disco duro de dicha maquina servidora por ejemplo, **c:\prograiv\misitio\mibase.mbd**, como se observa la base que se construyó en access (mibase.mbd) se almaceno en el disco c en el folder **prograiv** y dentro del subfolder **misitio**.

- El modo de comunicarse entre nuestro programa o aplicación y la base de datos (*ya sea física o un dbserver*) implica que ambos manejen un lenguaje de programación común, es decir no se puede mandar una instrucción en Basic o pascal, a la base de datos y además esperar que esta ultima la entienda (*para entender esto, una razón muy sencilla es que la base de datos tendría que conocer o comprender todos los lenguajes de programación*), para resolver este problema de comunicación es que se usa un lenguaje común de bases de datos que tanto los lenguajes de programación existentes como las bases de datos entienden, este lenguaje común de bases de datos es el SQL (*structured query lenguaje*) o lenguaje estructurado de consultas.

Ahora para mandar las instrucciones SQL a la base de datos, la respuesta son los siguientes OBJETOS.

— **OBJETO JDBCDBCDBDRIVER:** Objeto que se utiliza para traducir las instrucciones del lenguaje SQL a las instrucciones del lenguaje original de la base de datos.

\_ **OBJETO CONNECTION:** Objeto que se utiliza para establecer una conexión o enlace a la base de datos.

\_ **OBJETO RESULTSET:** Es la representación en memoria de una de las tablas de la base de datos en disco, se puede entender como una tabla virtual, recordar que generalmente todos los procesos que se realicen con la tabla (insertar registros, eliminar registros, etc) se realizarán realmente contra un *resultset* y no provocarán ningún cambio en la tabla física en disco, *resultset* tiene un conjunto de métodos muy útiles y muy usados para el proceso de los renglones de la tabla virtual.

\_ **OBJETO STATEMENT:** Este objeto y sus dos métodos *executequery* (*solo consultas de Selección*) y *executeupdate* (*Solo para consultas de Acción*) son los métodos que se utilizarán para comunicarse con la tabla física en disco.

Ejemplo:

```
Connection con = null;
ResultSet rs= null;
Statement stmt=null;
String sitiobase = "c:/prograiv/base/mibase.mdb";
String strcon= "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=" + sitiobase;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection(strcon);
stmt = canal.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
}
catch(java.lang.ClassNotFoundException e)
{
//instrucciones Catch
}
catch(SQLException e)
{
//instrucciones Catch
};
```

Para ejecutar el siguiente ejemplo debe incluir en la URL el parámetro EDAD, por ejemplo:

#### Consulta1.jsp

```
<HTML><HEAD><TITLE>EJEMPLO DE CONEXIÓN A BASES DE DATOS</TITLE></HEAD>
<BODY><H2 ALIGN=CENTER> INFORMACIÓN ALMACENADA EN LA BASE DE DATOS
</H2><HR><P>
<%@ page import="java.io.*, java.util.*, java.net.*, java.sql.*" %>
<%
Connection canal = null;
ResultSet tabla= null;
Statement instruccion=null;
String sitiobase = "c:/inetpub/wwwroot/MiAplicación/base/mibase.mdb";
String strcon= "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=" + sitiobase;
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
canal=DriverManager.getConnection(strcon);
instruccion = canal.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
}
catch(java.lang.ClassNotFoundException e)
{
out.println("EXCEPCION CLASE NO ENCONTRADA: "+e.getMessage());
```

```

}
catch(SQLException e)
{
out.println("EXCEPCION1 SQL: "+e.getMessage());
};
int edad = Integer.parseInt(request.getParameter("EDAD"));
String q="select * from mitabla where edad >="+edad;
try
{
tabla = instruccion.executeQuery(q);
out.println("<CENTER><TABLE Border=10 CellPadding=5><TR>");
out.println("<th bgcolor=Green>CLAVE</th><th bgcolor=White>NOMBRE</th><th bgcolor=Red>EDAD</th></TR>");
while(tabla.next())
{
out.println("<TR>");
out.println("<TD>"+tabla.getString(1)+"</TD>");
out.println("<TD>"+tabla.getString(2)+"</TD>");
out.println("<TD>"+tabla.getString(3)+"</TD>");
out.println("</TR>");
}; // fin while
out.println("</TABLE></CENTER></HTML>");
tabla.close();
} //fin try no usar ; al final de dos o mas catches
catch(SQLException e)
{
out.println("EXCEPCION2 SQL: "+e.getMessage());
};
try
{
canal.close();
}
catch(SQLException e)
{
out.println("EXCEPCION3 SQL: "+e.getMessage());
};
}%>

```

## IMAGENES EN APLICACIONES CON BASES DE DATOS.

Campos de gráficos o de imágenes, se han convertido en elementos importantes de cualquier base de datos.

Para manejar este elemento con java-jsp puedes utilizar el siguiente método:

Primero subir las imágenes (de preferencia jpg) con un ftp normal a tusitio o directorio donde guardarás las imágenes y después usar el tag **<img src>** de html y además agregar un campo de texto llamado fotourl o foto a la tabla en Access y grabar la dirección o path de la imagen en este campo, por ejemplo **http://programacionfacil.com/tusitio/pato.jpg** o simplemente **/tusitio/pato.jpg** Después solo cargar este tag imageurl en la página que se construirá que no es otra cosa que el programa de búsqueda con el despliegue del campo extra, como lo muestra el programa ejemplo

Para ejecutar el siguiente ejemplo debe incluir en la URL el parámetro CLAVE, por ejemplo:

**http://localhost/MiAplicación/Consulta2.jsp?CLAVE=1.**

Se asume además que todas las imágenes se guardan en el directorio "Base" y que tiene extensión jpg. Se recomienda analizar detenidamente el ejemplo.

### Consulta2.jsp

```

<HTML><HEAD><TITLE>EJEMPLO DE REGISTROS CON IMAGENES</TITLE></HEAD><BODY>
<H2 ALIGN=Center> INFORMACIÓN DEL USUARIO CON FOTOGRAFIA </H2><HR><P>
<%@ page import="java.io.*, java.util.*, java.net.*, java.sql.*" %>
<%

```

```

String foto;
Connection canal = null;
ResultSet tabla= null;
Statement instruccion=null;
String sitiobase = "c:/inetpub/wwwroot/llevar/base/mibase.mdb";
String strcon= "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=" + sitiobase;
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
canal=DriverManager.getConnection(strcon);
instruccion = canal.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
}
catch(java.lang.ClassNotFoundException e)
{
out.println("EXCEPCION CLASE NO ENCONTRADA: "+e.getMessage());
}
catch(SQLException e)
{
out.println("EXCEPCION1 SQL: "+e.getMessage());
};
int clave = Integer.parseInt(request.getParameter("CLAVE"));
String q="select * from mitabla where clave="+clave;
try {
tabla = instruccion.executeQuery(q);
out.println("<center><TABLE Border=10 CellPadding=5><TR>");
out.println("<th bgcolor=Green>CLAVE</th><th bgcolor=White>NOMBRE</th><th
bgcolor=Red>EDAD</th><th
bgcolor=gray>FOTOGRAFIA</th></TR>");
while(tabla.next())
{
out.println("<TR>");
out.println("<TD>"+tabla.getString(1)+"</TD>");
out.println("<TD>"+tabla.getString(2)+"</TD>");
out.println("<TD>"+tabla.getString(3)+"</TD>");
foto=tabla.getString(4);
out.println("<TD><img src=base/"+ foto+".jpg width=100 height=120>");
out.println("</TR>");
}; // fin while
out.println("</TABLE></CENTER></HTML>");
tabla.close();
}
catch(SQLException e)
{
out.println("EXCEPCION2 SQL: "+e.getMessage());
};
try
{
canal.close();
}
catch(SQLException e)
{
out.println("EXCEPCION3 SQL: "+e.getMessage());
};
%>

```

**EJERCICIOS**

Para la realización de los siguientes ejercicios deberás crear una base de datos en Access con el nombre de **MiAplicación.mdb**.

1. Construir una tabla (notas) en que traiga carnet, nombre, apellido, calif1, calif2, calif3 y promedio, cargar en Access unos 5 renglones de alumnos, no cargar promedio, el promedio lo deberán calcular con un programa en JSP.

**NOTA.** CALIF1 equivale al 30% de la nota final, CALIF2 Y CALIF3 cada una 35%.

2. Crear un programa en JSP que muestre la información almacenada en la tabla anterior.

3. Construir un proceso de búsqueda de un alumno por medio de carnet, nombre o apellido y que pueda mostrar los datos de sus calificaciones y promedio final.

4. Crear la interfaz de entrada de una aplicación web, que pida el identificador de usuario y contraseña, la validez de estos parámetros se verificará con la información almacenada en la base de datos. Si el usuario es válido se desplegará la pantalla de entrada del sistema en una interfaz web compuesta por 3 marcos (frames); un encabezado que mostrará un mensaje de bienvenida y la fecha del sistema, un menú principal, y una pagina principal, como se muestra en la figura.

ENCABEZADO	
M E N	PAGINA PRINCIPAL

**TABLA USUARIOS**

Idusuario
Nombre
Apellido
Contraseña

## Clase N° 8

### Manejo de sesiones y cookies con JSP



#### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir que es el estado de sesión.
- Manejar el estado de sesión utilizando sesiones y cookies con JSP.

#### DESARROLLO

##### Introducción

El protocolo HTTP permite acceder a páginas web y enviar datos de un formulario pero tiene una limitación que consiste en que no puede almacenar cuando se cambia de servidor o de página dentro de un mismo servidor. Por esta razón a este protocolo se le conoce como protocolo sin estado.

Cuando se solicita una página independientemente del tipo que sea, el servidor abre una conexión por la que envía los datos y luego ésta es cerrada una vez que ha terminado.

##### Qué es una sesión

Una sesión es una serie de comunicaciones entre un cliente y un servidor en la que se realiza un intercambio de información. Por medio de una sesión se puede hacer un seguimiento de un usuario a través de la aplicación.

El tiempo de vida de una sesión comienza cuando un usuario se conecta por primera vez a un sitio web pero su finalización puede estar relacionada con tres circunstancias:

- **Cuando se abandona el sitio web.**

- **Cuando se alcanza un tiempo de inactividad que es previamente establecido, en este caso la sesión es automáticamente eliminada. Si el usuario siguiera navegando se crearía una nueva sesión.**

- **Se ha cerrado o reiniciado el servidor**

Una posible aplicación de las sesiones es en el comercio electrónico. En este caso una sesión permite ir eligiendo una serie de productos e irlos añadiendo a nuestro "carrito" y así hasta finalizar la compra. Sin el uso de sesiones no se podría hacer porque al ir navegando de una página a otra se iría perdiendo toda la información.

También se utilizan para la identificación de usuarios, en la que se deben de introducir un *login* y un *password*. Después de haber hecho esto el usuario tendrá una serie de permisos sobre las páginas que va a visitar, de tal forma que si un usuario intenta pasar a una página si haberse identificado, el sistema comprobará que no se ha identificado y sería redireccionado a la página de identificación. Para poder realizarse estas operaciones es necesario almacenar en una sesión la información necesaria para saber que el usuario se ha identificado correctamente.

Para poder hacer uso de las sesiones en JSP hay que poner el atributo *session* de la directiva *page* a *true*, de esta forma se notifica al contenedor que la página interviene en un proceso que utiliza las sesiones del protocolo HTTP:

```
<%@page session='true'%>
```

El manejo de las sesiones impide el intercambio de datos entre ellas ya que se trata información específica para cada usuario e incluso si se trata del mismo usuario.



## Manejo de las sesiones

En JSP las acciones que se pueden realizar sobre las sesiones se lleva a cabo mediante la interface `HttpSession` y los métodos que implementa. Esta interfaz está incluida dentro del paquete `javax.servlet.http` y es utilizada por el contenedor de páginas JSP para crear una sesión entre el servidor y el cliente.

Para obtener la sesión de un usuario se utiliza el método `getSession()` que devuelve una interfaz de tipo `HttpSession`.

```
<%  
HttpSession sesion=request.getSession();  
%>
```

Una vez creado el objeto de tipo sesión es posible acceder a una serie de datos sobre la misma. Uno de estos datos es `idSession` que devuelve un identificador único asociado a una sesión:

```
<%  
HttpSession sesion=request.getSession();  
out.println("IdSesion: "+sesion.getId());  
%>
```

Es posible conocer el momento en el que se creó la sesión:

```
<%@page import="java.util.*" session="true"%>  
<%  
HttpSession sesion=request.getSession();  
out.println("Creación: "+sesion.getCreationTime());  
Date momento=new Date(sesion.getCreationTime());  
out.println("<BR>Creación: "+momento);  
%>
```

En el primer caso se muestra el dato tal cual lo devuelve el método `getCreationTime()`, que es una fecha en formato long, mientras que en el segundo caso se formatea para que tenga un aspecto más común.

También se puede conocer la fecha y hora de la última vez que el cliente accedió al servidor con el que se creó la sesión, utilizando el método `getLastAccessedTime()`:

```
<%  
Date acceso=new Date(sesion.getLastAccessedTime());  
out.println("Último acceso: "+acceso+"<br>");  
%>
```

Teniendo en cuenta el momento en el que se creó la sesión y la última vez que se accedió al servidor, se puede conocer el tiempo que lleva el cliente conectado al servidor, o lo que es lo mismo el tiempo que lleva el usuario navegando por la páginas JSP:

```
<%  
long longDuracion=sesion.getLastAccessedTime()  
sesion.getCreationTime();  
Date duracion=new Date(longDuracion);  
out.println("Duracion:  
"+duracion.getMinutes()+"min."+duracion.getSeconds()+"seg")  
;  
%>
```

La interfaz `HttpSession` ofrece el método `isNew()` mediante el cual es posible saber si la sesión creada es nueva o se está tomando de una previamente creada:

```
<%
HttpSession sesion=request.getSession();
out.println("nueva: "+sesion.isNew());
%>
```

Si se ejecuta el ejemplo la primera vez el método devolverá true, ya que previamente no había ninguna sesión y ha sido creada en ese instante. Si se recarga la página devolverá false ya que la sesión ya ha sido creada.

### Guardar objetos en una sesión

Para guardar un objeto en una sesión se utiliza el método `setAttribute()`, que ha sustituido al método `putValue()`. Este método utiliza dos argumentos:

- El primero es el nombre que identificará a esa variable.
- El segundo es el dato que se va a guardar.

### setAttribute(java.lang.String name, java.lang.Object value)

Un ejemplo de cómo guardar una cadena de texto en la sesión:

```
<%@page import="java.util.*" session="true" %>
<%
HttpSession sesion=request.getSession();
sesion.setAttribute("trabajo","Paginas de JSP");
%>
```

Si se quiere pasar un parámetro que no sea un objeto es necesario realizar una conversión:

```
<%@page import="java.util.*" session="true" %>
<%
HttpSession sesion=request.getSession();
Integer edad=new Integer(26);
sesion.setAttribute("edad",edad);
%>
```

Si se hubiera utilizado el valor entero en vez del objeto Integer, el resultado habría sido similar al siguiente.

Incompatible type for meted. Can't convert int to java.lang.Object.

En el primer ejemplo este no sucedería puesto que una cadena es un objeto de tipo String, no así un entero . Así habría sido igual si en el primer caso ponemos:

```
<%@page import="java.util.*" session="true" %>
<%
HttpSession sesion=request.getSession();
String nombre=new String("Paginas de JSP.");
sesion.setAttribute("trabajo",nombre);
%>
```

En caso de tratarse objeto de tipo Vector (parecido a un array con dos diferencias: la primera es que puede almacenar todo tipo de objetos, y la segunda es que no es necesario establecer de forma previa el tamaño que va a tener) que almacene los 7 días de la semana. El código sería el siguiente:

```
<%@page import="java.util.*" session="true" %>
<%
HttpSession sesion=request.getSession();
```

```

Vector v=new Vector();
v.addElement(new String("Lunes"));
v.addElement(new String("Martes"));
v.addElement(new String("Miercoles"));
v.addElement(new String("Jueves"));
v.addElement(new String("Viernes"));
v.addElement(new String("Sábado"));
v.addElement(new String("Domingo"));
sesion.setAttribute("diasSemana",v);
%>

```

### Recuperar objetos de una sesión

Los datos que se guardan en la sesión permanecen ahí a la espera de ser utilizados. Para ello es necesario realizar el proceso contrario a cuando se graban, comenzando por la recuperación del objeto de la sesión para empezar a ser tratado.

Para poder realizar este paso se utiliza el método `getAttribute()` (anteriormente se utilizaba el método `getValue()`, pero este método se encuentra en desuso), utilizando como argumento el nombre que identifica al objeto que se quiere recuperar.

```
getAttribute(java.lang,String nombre)
```

Un ejemplo de recuperación de objetos almacenados en la sesión:

```

<%
HttpSession sesion=request.getSession();
Sesion.getAttribute("nombre");
%>

```

Cuando este método devuelve el objeto no establece en ningún momento de qué tipo de objeto se trata (`String`, `Vector`...)

Por ello si se conoce previamente el tipo de objeto que puede devolver tras ser recuperado de la sesión es necesario realizar un *casting*, para convertir el objeto de tipo genérico al objeto exacto que se va a usar. Para realizar esta operación se añade el tipo de objeto al lado de tipo `HttpSession` que utiliza el método `getAttribute()` para obtener el objeto que devuelve:

```

<%
HttpSession sesion=request.getSession();
String nombre=(String)sesion.getAttribute("nombre");
out.println("Contenido de nombre: "+nombre);
%>

```

Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el método `getAttribute()`, el valor devuelto será *null*. Por ello habrá que prestar especial atención ya que si se realiza el *casting* de un valor *null* el contenedor JSP devolverá un error. Lo mejor en estos casos es adelantarse a los posibles errores que pueda haber.

```

<%
if(sesion.getAttribute("nombre")!=null)
{
String nombre=(String)sesion.getAttribute("nombre");
out.println("Contenido de nombre: "+nombre);
}
%>

```

Por último, el ejemplo del vector guardado en la sesión tiene un tratamiento similar al de los casos anteriores. El primer paso es recuperar el objeto de la sesión:

```
<%@page import="java.util.*" session="true" %>
<%
HttpSession sesion=request.getSession();
sesion.getAttribute("diasSemana");
%>
```

Como se sabe que el objeto es de tipo Vector se puede recuperar y convertir en un solo paso:

```
Vector v= (Vector) sesion.getAttribute("diasSemana");
```

A partir de este momento se puede acceder a los elementos del vector independientemente de si venía de una sesión o ha sido creado. Para ello se utiliza el método `size()` que devuelve el tamaño del vector para ir leyendo cada uno de sus elementos:

```
<%
for(int i=0; i<v.size(); i++)
{
out.println("<b>Dia: </b>" + (String)v.get(i) + "<br>");
}
%>
```

### Cómo se destruye una sesión

Como se ha visto, los datos almacenados por las sesiones pueden destruirse en tres casos:

- El usuario abandona aplicación web (cambia de web o cierra el navegador) - Se alcanza el tiempo máximo permitido de inactividad de un usuario (*timeout*).
- El servidor se para o se reinicia.

Pero la situación más probable es querer iniciar las sesiones o dar por finalizada una si se ha cumplido una o varias condiciones. En este caso no es necesario esperar a que ocurra alguno de los tres casos citados anteriormente, ya que mediante el método **invalidate()** es posible destruir una sesión concreta.

En el siguiente caso la sesión "sesión" se destruye al invocar el método `invalidate()`; y por la tanto el valor u objeto que está asociado a la misma.

```
<%
[...]
sesion.invalidate();
%>
```

### Cookies

Las sesiones vistas anteriormente basan su funcionamiento en los *cookies*. Cuando se hace uso de la interfaz `HttpSession` de forma interna y totalmente transparente al programador se está haciendo uso de los *cookies*. De hecho cuando a través de una página JSP se comienza una sesión, se crea un *cookie* llamado `JSESSIONID`. La diferencia es que este *cookie* es temporal y durará el tiempo que permanezca el navegador ejecutándose, siendo borrada cuando el usuario cierre el navegador.

### Crear una cookie

Un *cookie* almacenado en el ordenador de un usuario está compuesto por un nombre y un valor asociado al mismo. Además, asociada a este *cookie* pueden existir una serie de atributos que definen datos como su tiempo de vida, alcance, dominio, etc.

Cabe reseñar que los *cookies*, no son más que ficheros de texto, que no pueden superar un tamaño de 4Kb, además los navegadores tan sólo pueden aceptar 20 *cookies* de un mismo servidor web (300 *cookies* en total).

Para crear un objeto de tipo Cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar. El siguiente ejemplo crearía un objeto Cookie que contiene el nombre "nombre" y el valor "objetos".

```
<%  
Cookie miCookie=new Cookie("nombre","objetos");  
%>
```

También es posible crear *cookies* con contenido que se genere de forma dinámica. El siguiente código muestra un *cookie* que guarda un texto que está concatenado a la fecha/hora en ese momento:

```
<%@page contentType="text/html; charset=iso-8859-1"  
session="true" language="java" import="java.util.*" %>  
<%  
Cookie miCookie=null;  
Date fecha=new Date();  
String texto= "Este es el texto que vamos a guardar en el cookie"+fecha;  
miCookie=new Cookie("nombre",texto);  
%>
```

Por defecto, cuando creamos un *cookie*, se mantiene mientras dura la ejecución del navegador. Si el usuario cierra el navegador, los *cookies* que no tengan establecido un tiempo de vida serán destruidos.

Por tanto, si se quiere que un *cookie* dure más tiempo y esté disponible para otras situaciones es necesario establecer un valor de tiempo (en segundos) que será la duración o tiempo de vida del *cookie*. Para establecer este atributo se utiliza el método `setMaxAge()`. El siguiente ejemplo establece un tiempo de 31 días de vida para el *cookie* "unCookie":

```
<%  
unCookie.setMaxAge(60*60*24*31);  
%>
```

Otros de los atributos que se incluye cuando se crea un *cookie* es el *path* desde el que será visto, es decir, si el valor del *path* es "/" (raíz), quiere decir que en todo el *site* se podrá utilizar ese *cookie*, pero si el valor es "/datos" quiere decir que el valor del *cookie* sólo será visible dentro del directorio "datos". Este atributo se establece mediante el método `setPath()`.

```
<%  
unCookie.setPath("/");  
%>  
Para conocer el valor de path, se puede utilizar el método getPath().  
<%  
out.println("cookie visible en: "+unCookie.getPath());  
%>
```

Existe un método dentro de la clase Cookie que permite establecer el dominio desde el cual se ha generado el *cookie*. Este método tiene su significado porque un navegador sólo envía al

servidor los *cookies* que coinciden con el dominio del servidor que los envió. Si en alguna ocasión se requiere que estén disponibles desde otros subdominios se especifica con el método `setDomain()`. Por ejemplo, si existe el servidor web en la página `www.paginasjsp.com`, pero al mismo tiempo también existen otros subdominios como `usuario1.paginasjsp.com`, `usuario2.paginasjsp.com`, etc.

En el siguiente ejemplo hace que el *cookie* definido en el objeto "unCookie" esté disponible para todos los dominios que contengan el nombre ".paginasjsp.com". Un nombre de dominio debe comenzar por un punto.

```
<%
unCookie.setDomain(".paginasjsp.com");
%>
```

Igualmente, para conocer el dominio sobre el que actúa el *cookie*, basta con utilizar el método `getDomain()` para obtener esa información.

Una vez que se ha creado el objeto *Cookie*, y se ha establecido todos los atributos necesarios es el momento de crear realmente, ya que hasta ahora sólo se tenía un objeto que representa ese *cookie*.

Para crear el fichero *cookie* real, se utiliza el método `addCookie()` de la interfaz `HttpServletResponse`:

```
<%
response.addCookie(unCookie);
%>
```

Una vez ejecutada esta línea es cuando el *cookie* existe en el disco del cliente que ha accedido a la página JSP.

Es importante señalar que si no se ejecuta esta última línea el *cookie* no habrá sido grabado en el disco, y por lo tanto, cualquier aplicación o página que espere encontrar dicho *cookie* no lo encontrará.

### **Recuperar un cookie**

El proceso de recuperar un *cookie* determinado puede parecer algo complejo, ya que no hay una forma de poder acceder a un *cookie* de forma directa. Por este motivo es necesario recoger todos los *cookies* que existen hasta ese momento e ir buscando aquél que se quiera, y que al menos, se conoce su nombre.

Para recoger todos los *cookies* que tenga el usuario guardados se crea un array de tipo *Cookie*, y se utiliza el método `getCookies()` de la interfaz `HttpServletRequest` para recuperarlos:

```
<%
Cookie [] todosLosCookies=request.getCookies();
/* El siguiente paso es crear un bucle que vaya leyendo
todos los cookies. */
for(int i=0;i<todosLosCookies.length;i++)
{
Cookie unCookie=todosLosCookies[i];
/* A continuación se compara los nombres de cada uno de
los cookies con el que se está buscando. Si se encuentra un
cookie con ese nombre se ha dado con el que se está
buscando, de forma que se sale del bucle mediante break. */
if(unCookie.getName().equals("nombre"))
break;
}
/* Una vez localizado tan sólo queda utilizar los
métodos apropiados para obtener la información necesaria
que contiene. */
out.println("Nombre: "+unCookie.getName()+"<BR>");
out.println("Valor: "+unCookie.getValue()+"<BR>");
out.println("Path: "+unCookie.getPath()+"<BR>");
```

```
out.println("Tiempo de vida:"+unCookie.getMaxAge()+"<BR>");  
out.println("Dominio: "+unCookie.getDomain()+"<BR>");  
%>
```



## Guía Práctica No 8 Manejo de sesiones y cookies con JSP



### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Desarrollar aplicaciones utilizando sesiones con JSP.
- Crear aplicaciones utilizando cookies con JSP.

### PROCEDIMIENTO

#### Administración de usuarios.

Un caso práctico donde poder usar las sesiones es en las páginas a las que se debe acceder habiendo introducido previamente un usuario y una clave. Si no se introducen estos datos no se podrán visualizar y de igual modo si alguien intenta entrar directamente a una de estas páginas sin haberse identificado será redirigido a la página principal para que se identifique y, de este modo, no pueda acceder de forma anónima.

La primera página de la aplicación JSP es en la que el usuario se debe identificar con un nombre de usuario y una clave por lo que su aspecto será el de un formulario.

La página JSP contiene el formulario el cual especifica la página destino cuando el usuario pulse el botón de enviar los datos. Además se ha añadido una comprobación en la que en caso de recibir un parámetro llamado "error" se muestra el mensaje que contenga. De esta forma el usuario ve qué tipo de error se ha producido.

#### **login.jsp**

```
<%@page contentType="text/html; charset=iso-8859-1"
session="true" language="java" import="java.util.*" %>
<html>
<head><title>Proceso de login</title>
</head>
<body>
<b>Proceso de identificación</B>
<p>
<%
if(request.getParameter("error")!=null)
```

Esta página es la encargada de recoger del usuario y la clave enviados desde el formulario. Una vez recibidos se almacenan en dos variables("usuario" y "clave") de tipo String. A continuación se comparan con los valores correctos del usuario y la clave.

Si esta comprobación es correcta se crea un objeto de tipo session y se guarda el valor en la variable "usuario" en la sesión mediante el método setAttribute().

A continuación y mediante la opción estándar <jsp: forward> se redirecciona al usuario a la página final en la que se encuentra el menú de opciones al que se accede después de haber completado de forma satisfactoria el proceso de identificación.

En caso que la comprobación de usuario y clave no se cumpla se redirecciona al usuario hacia la página de inicio, para que vuelva a identificarse incluyendo esta vez un parámetro llamado "error" con un mensaje que avisará de qué es lo que le ha ocurrido.



**checklogin.jsp**

```

<%@ page session="true" %>
<%
String usuario = "";
String clave = "";
if (request.getParameter("usuario") != null)
usuario = request.getParameter("usuario");
if (request.getParameter("clave") != null)
clave = request.getParameter("clave");
if (usuario.equals("spiderman") &&
clave.equals("librojsp")) {
HttpSession sesionOk = request.getSession();
sesionOk.setAttribute("usuario",usuario);
%>
<jsp:forward page="menu.jsp" />
<%
} else {
%>
<jsp:forward page="login.jsp">
<jsp:param name="error" value="Usuario y/o clave
incorrectos.<br>Vuelve a intentarlo."/>
</jsp:forward>
<%
}
%>

```

**menu.jsp**

```

<%@ page session="true" %>
<%
String usuario = "";
HttpSession sesionOk = request.getSession();
if (sesionOk.getAttribute("usuario") == null) {
%>
<jsp:forward page="login.jsp">
<jsp:param name="error" value="Es
obligatorio identificarse"/>
</jsp:forward>
<%
} else {
usuario = (String)sesionOk.getAttribute("usuario");
}
%>
<html>
<head><title>Proceso de login</title>
</head>
<body>
<b>PROCESO DE IDENTIFICACIÓN</b><p>
<b>Menú de administración</b><br>
<b>Usuario activo:</b> <%=usuario%><p>
<li> <a href="opc1.jsp">Crear nuevo usuario</a>
<li> <a href="opc2.jsp">Borrar un usuario</a>
<li> <a href="opc3.jsp">Cambiar clave</a>
<p>
<li> <a href="cerrarsesion.jsp">Cerrar sesión</a>
</body>
</html>

```

La última opción que incorpora el menú es la de "Cerrar sesión", que será de gran utilidad

cuando se haya finalizado el trabajo y queremos estar seguro que nadie realiza ninguna acción con nuestro usuario y clave.

Al pulsar este enlace, se recupera de nuevo la sesión y mediante el método `invalidate()` se da por finalizada la sesión.

### ***cerrarsesion.jsp***

```
<%@ page session="true" %>
<%
HttpSession sesionOk = request.getSession();
sesionOk.invalidate();
%>
<jsp:forward page="login.jsp"/>
```

### ***Utilizar cookies***

Para realizar un ejemplo práctico se va a seguir con el ejemplo de Sesiones. El objetivo será modificar las páginas necesarias para que si el usuario selecciona un campo de tipo *checkbox* (que será necesario añadir) el nombre de usuario le aparezca por defecto cuando vuelva a entrar a esa página. Este nombre de usuario estará guardado en un *cookie* en su ordenador.

El primer paso es añadir el *checkbox* en la página `login.jsp`:

```
<%@ page session="true" import="java.util.*"%>
<%
String usuario = "";
String fechaUltimoAcceso = "";
/*Búsqueda del posible cookie si existe para recuperar
su valor y ser mostrado en el campo usuario */
Cookie[] todosLosCookies = request.getCookies();
for (int i=0; i<todosLosCookies.length; i++) {
Cookie unCookie = todosLosCookies[i];
if (unCookie.getName().equals("cookieUsu")) {
usuario = unCookie.getValue();
}
}
/* Para mostrar la fecha del último acceso a la página.
Para ver si el cookie que almacena la fecha existe, se busca en los
cookies existentes. */
for (int i=0; i<todosLosCookies.length; i++) {
Cookie unCookie = todosLosCookies[i];
if (unCookie.getName().equals("ultimoAcceso")) {
fechaUltimoAcceso = unCookie.getValue();
}
}
/* Se comprueba que la variable es igual a vacío, es decir
no hay ningún cookie llamado "ultimoAcceso", por lo que se
recupera la fecha, y se guarda en un nuevo cookie. */
if (fechaUltimoAcceso.equals(""))
{
Date fechaActual = new Date();
fechaUltimoAcceso = fechaActual.toString();
Cookie cookieFecha = new
Cookie("ultimoAcceso",fechaUltimoAcceso);
cookieFecha.setPath("/");
cookieFecha.setMaxAge(60*60*24);
response.addCookie(cookieFecha);
}
%>
<html>
<head><title>Proceso de login</title>
</head>
```

```

<body>
<b>PROCESO DE IDENTIFICACIÓN</b>
<br>Última vez que accedió a esta
página: <br><%=fechaUltimoAcceso%>
<p>
<%
if (request.getParameter("error") != null) {
out.println(request.getParameter("error"));
}
%>
<form action="checklogin.jsp" method="post">
usuario: <input type="text" name="usuario" size="20"
value="<%=usuario%>"><br>
clave: <input type="password" name="clave" size="20"><br>
Recordar mi usuario: <input type="checkbox"
name="recordarUsuario" value="on"><br>
<input type="submit" value="enviar">
</form>
</body>
</html>

```

El siguiente paso es modificar la página checklogin.jsp que recoge el usuario y clave introducidos y por lo tanto ahora también la nueva opción de "Recordar mi usuario". Dentro de la condición que se cumple si el usuario y la clave son correctos, y después de crear la sesión, escribimos el código que creará el *cookie* con el usuario. El primer paso es comprobar que el usuario ha activado esta opción, es decir, ha seleccionado el *checkbox*. También se realiza la comprobación de que el campo "recordarUsuario" no llegue con el valor nulo y produzca un error en la aplicación, en caso de que el usuario deje sin seleccionar el *checkbox*:

```

<%@ page session="true" import="java.util.*"%>
<%
String usuario = "";
String clave = "";
if (request.getParameter("usuario") != null)
usuario = request.getParameter("usuario");
if (request.getParameter("clave") != null)
clave = request.getParameter("clave");
if (usuario.equals("spiderman") &&
clave.equals("librojsp")) {
out.println("checkbox: " +
request.getParameter("recordarUsuario") + "<br>");
HttpSession sesionOk = request.getSession();
sesionOk.setAttribute("usuario",usuario);
if ((request.getParameter("recordarUsuario") != null) &&
(request.getParameter("recordarUsuario").equals("on")))
{
out.println("entra");
Cookie cookieUsuario = new Cookie
("cokieUsu",usuario);
cookieUsuario.setPath("/");
cookieUsuario.setMaxAge(60*60*24);
response.addCookie(cookieUsuario);
}
/* Se realiza un proceso similar a la creación de cookie de
recordar el usuario. En este caso se trata de crear un nuevo cookie
con el nuevo valor de la fecha y guardarlo con el mismo nombre. De
esta forma será borrado el anterior y prevalecerá el valor del último.
*/
Date fechaActual = new Date();

```

```
String fechaUltimoAcceso = fechaActual.toString();
Cookie cookieFecha = new
Cookie("ultimoAcceso", fechaUltimoAcceso);
cookieFecha.setPath("/");
cookieFecha.setMaxAge(60*60*24);
response.addCookie(cookieFecha);
%>
<jsp:forward page="menu.jsp" />
<%
} else {
%>
<jsp:forward page="login.jsp">
<jsp:param name="error" value="Usuario y/o clave
incorrectos.<br>Vuelve a intentarlo."/>
</jsp:forward>
<%
}
%>
```

#### EJERCICIOS

1. Crear una aplicación que valide un usuario utilizando una base de datos con el nombre empleado y una tabla con el nombre usuarios con los campos siguientes: login y contraseña, Si el usuario es correcto, entonces se creara la sesión que guarde el login del usuario y se direccionara a una pagina jsp.
2. Crear la pagina jsp donde se re direccionara en el ejercicio #1, la cual permita validar si existe la sesión, en caso que no exista se direccionara a la pagina de validación.
3. Realizar los ejercicios anteriores utilizando cookies.



## OBJETIVOS

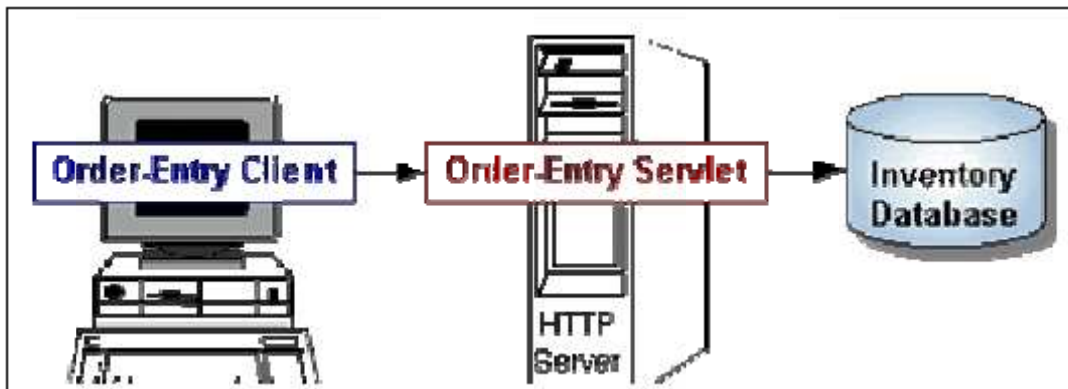
Al finalizar la clase, el alumno será capaz de:

- Definir los conceptos básicos de la programación orientada a objetos.
- Nombrar las tecnologías de Java para web y la estructura básica de éstas.
- Identificar los métodos que existen en la programación para el web, para el envío de información.

## DESARROLLO

### INTRODUCCIÓN A LOS SERVLETS

Podemos decir que los Servlets son programas o módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java. Por ejemplo, un servlet podría ser responsable de tomar los datos de un formulario de entrada de pedidos en HTML y aplicarle la lógica de negocios utilizada para actualizar la base de datos de pedidos de la compañía.



Los Servlets son para los servidores lo que los applets son para los navegadores. Sin embargo, al contrario que los applets, los servlets no tienen interfase gráfico de usuario.

Los servlets pueden ser incluidos en muchos servidores diferentes porque el API Servlet, el que se utiliza para escribir Servlets, no asume nada sobre el entorno o protocolo del servidor. Los servlets se están utilizando ampliamente dentro de servidores HTTP; muchos *servidores Web soportan el API Servlet*.

### UTILIZAR SERVLETS EN LUGAR DE SCRIPTS CGI!

Los Servlets son un reemplazo efectivo para los scripts CGI. Proporcionan una forma de generar documentos dinámicos que son fáciles de escribir y rápidos en ejecutarse. Los Servlets también solucionan el problema de hacer la programación del lado del servidor con APIs específicos de la plataforma: están desarrollados con el API Java Servlet, una extensión estándar de Java.

Por eso se utilizan los servlets para manejar peticiones de cliente HTTP. Por ejemplo, tener un servlet procesando datos POSTeados sobre HTTP utilizando un formulario HTML, incluyendo datos del pedido o de la tarjeta de crédito. Un servlet como este podría ser parte de un sistema de procesamiento de pedidos, trabajando con bases de datos de productos e inventarios, y quizás un sistema de pago on-line.

## Otros usos de los Servlets

- Permitir la colaboración entre la gente. Un servlet puede manejar múltiples peticiones concurrentes, y puede sincronizarlas. Esto permite a los servlets soportar sistemas como conferencias on-line
- Reenviar peticiones. Los Servlets pueden reenviar peticiones a otros servidores y servlets. Con esto los servlets pueden ser utilizados para cargar balances desde varios servidores que reflejan el mismo contenido, y para particionar un único servicio lógico en varios servidores, de acuerdo con los tipos de tareas o la organización compartida.

## ¿Qué son los Servlets Java?

Los Servlets son la respuesta de la tecnología Java a la programación CGI. Son programas que se ejecutan en un servidor Web y construyen páginas Web. Construir páginas Web dinámicas es útil (y comunmente usado) por un número de razones:

- **La página Web está basada en datos enviados por el usuario.** Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde sites de comercio electrónico también.
- **Los datos cambian frecuentemente.** Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.
- **Las páginas Web que usan información desde bases de datos corporativas u otras fuentes.** Por ejemplo, usaríamos esto para hacer una página Web en una tienda on-line que liste los precios actuales y el número de artículos en stock.

## ¿Cuáles son las Ventajas de los Servlets sobre el CGI "Tradicional"?

Los Servlets Java son más eficientes, fáciles de usar, más poderosos, más portables, y más baratos que el CGI tradicional y otras muchas tecnologías del tipo CGI.

- **Eficiencia.** Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP. Si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los Servlets, la máquina Virtual Java permanece arrancada, y cada petición es manejada por un thread Java de peso ligero, no un pesado proceso del sistema operativo. De forma similar, en CGI tradicional, si hay N peticiones simultáneas para el mismo programa CGI, el código de este problema se cargará N veces en memoria. Sin embargo, con los Servlets, hay N threads pero sólo una copia de la clase Servlet. Los Servlet también tienen más alternativas que los programas normales CGI para optimizaciones como los caches de cálculos previos, mantener abiertas las conexiones de bases de datos, etc.

- **Conveniencia.** ¿Por qué aprender otro lenguaje? Junto con la conveniencia de poder utilizar un lenguaje familiar, los Servlets tienen una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y seleccionar cabeceras HTTP, manejar cookies, seguimiento de sesiones, y muchas otras utilidades.

- **Potencia.** Los Servlets Java nos permiten fácilmente hacer muchas cosas que son difíciles o imposibles con CGI normal. Por algo, los servlets pueden hablar directamente con el servidor Web. Esto simplifica las operaciones que se necesitan para buscar imágenes y otros datos almacenados en situaciones estándar. Los Servlets también pueden compartir los datos entre ellos, haciendo las cosas útiles como almacenes de conexiones a bases de datos fáciles de implementar. También pueden mantener información de solicitud en solicitud, simplificando cosas como seguimiento de sesión y el caché de cálculos anteriores.

- **Portable.** Los Servlets están escritos en Java y siguen un API bien estandarizado. Consecuentemente, los servlets escritos, digamos en el servidor I-Planet Enterprise, se pueden ejecutar sin modificarse en Apache, Microsoft IIS, o WebStar. Los Servlets están soportados directamente o mediante plug-in en la mayoría de los servidores Web.

- **Barato.** Hay un número de servidores Web gratuitos o muy baratos que son buenos para el uso "personal" o el uso en sites Web de bajo nivel. Sin embargo, con la excepción de Apache, que es gratuito, la mayoría de los servidores Web comerciales son relativamente caros. Una vez

que tengamos un servidor Web, no importa el coste del servidor, añadirle soporte para Servlets (si no viene preconfigurado para soportarlos) es gratuito o muy barato.

### ¿Dónde puedo ejecutar Servlets y qué necesito?

En la actualidad la mayoría de servidores web tanto comerciales como de licencia libre tienen la capacidad de ejecutar servlets a través de plug-ins o módulos. A continuación señalaremos unos cuantos:

- \_ Apache web server
- \_ Netscape FastTrack 2.0
- \_ Microsoft IIS
- \_ WebLogic
- \_ Lotus Domino Go Web Server
- \_ IBM Internet Conexión Server
- \_ Java Web Server

Con respecto a este último cabe destacar que ejecuta servlets de forma nativa sin necesidad de módulos adicionales. Señalaremos dos módulos de ejecución de servlets Allaire's JRun y Jakarta Tomcat ambos gratuitos y descargables desde su página web si no es para usos comerciales.

Como dato adicional el JSDK 2.1 incluye una herramienta llamada *servletrunner* análoga a *appletviewer* para la ejecución y depuración de servlet con unas capacidades muy limitadas por lo que solo se debe usar para comprobar la exactitud del servlet.

### Estructura de un servlet

El API Servlet consiste básicamente en dos paquetes:

·**javax.servlet** En este paquete se definen 6 interfaces y 3 clases para la implementación de servlets genéricos, sin especificación de protocolo. Hoy en día no tienen utilidad práctica más que para servir de base en la jerarquía de clases de los servlets.

Conforme pase el tiempo se supone que constituirán la base para la implementación de otros protocolos distintos de http.

·**javax.servlet.http** Ofrece la implementación específica de servlets para el protocolo http. En estos paquetes se definen todas las clases e interfaces necesarias para la escritura de applets. De hecho cuando se usen los servlets para gestionar conexiones http usaremos las clases del paquete **javax.servlet.http**.

El ciclo de ejecución de un servlet es análogo al de un applet con ligeras diferencias. Inicialmente el servlet debe extender a la clase **HttpServlet**:

```
import javax.servlet;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class MiServlet extends HttpServlet{  
    ....  
}
```

Para dotar de funcionalidad a un servlet se han de redefinir una serie de métodos que guardan una analogía con los métodos de funcionamiento de un applet (**init()**, **start()**, **stop()**, **destroy()**). **public void init(ServletConfig config)**

Cada vez que se inicia el servlet el servidor web llama a este método pasando un parámetro de la clase **ServletConfig** que guarda información de la configuración del servlet y del contexto del servidor web en el que se ejecuta. A través de **ServletConfig** se accede a los parámetros de inicialización del servlet que se establecieron al configurar el servlet y a través de la interfaz **ServletContext** (obtenido a partir del método **getServletContext()** de **ServletConfig**) se accede a la información del servidor web.

El siguiente es un ejemplo simple de un servlet que escribe información en un fichero de registro (el formato, ubicación y nombre de este es dependiente del servidor web):

#### **MiServlet.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class MiServlet extends HttpServlet{
    public void init(ServletConfig config)
    { config.getServletContext().log("Iniciado MiServlet a las" +new Date()); }
}
```

En este método se han de realizar todas las operaciones únicas en el ciclo de vida del servlet tal como conexión a BD de forma persistente y otras tareas de inicialización. Dado que el servlet se carga en memoria al iniciar el servidor web o al recibir la primera petición (dependiendo de la configuración) el método `init()` es llamado *solo* una vez, no cada vez que se realice una petición.

#### · **public void destroy()**

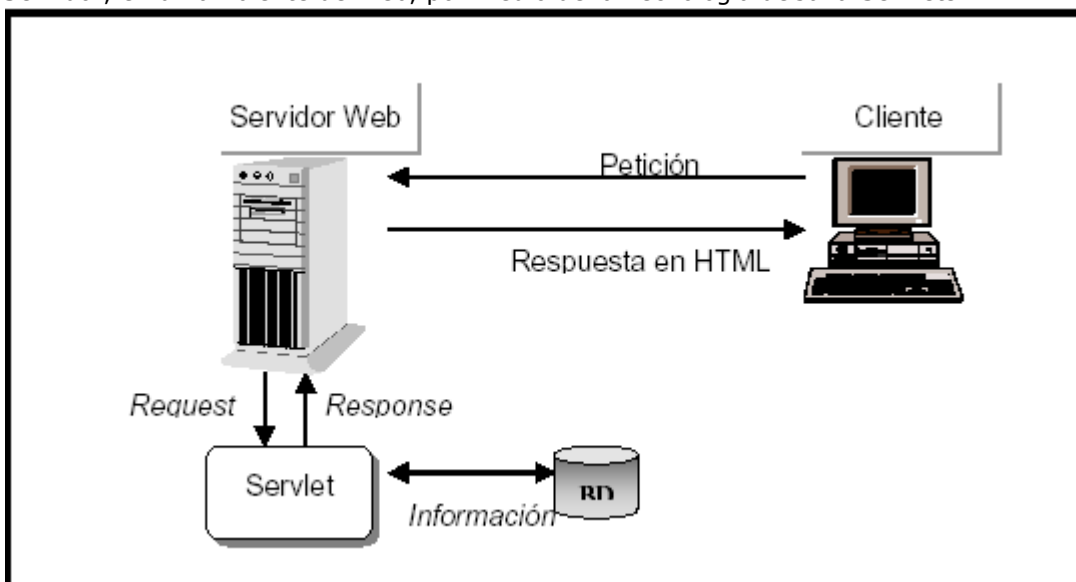
Este método es análogo al método **init()** solo que será llamado por el servidor web cuando el servlet esta a punto de ser descargado de memoria (no cuando termina una petición). En este método se han de realizar las tareas necesarias para conseguir una finalización apropiada como cerrar archivos y flujos de entrada de salida externos a la petición, cerrar conexiones persistentes a bases de datos, etc. Un punto importante es que se puede llamar a este método cuando todavía esta ejecutándose alguna petición por lo que podría producirse un fallo del sistema y una inconsistencia de datos tanto en archivos como en BD. Por eso debe retrasarse la desaparición del servlet hasta que todas las peticiones hayan sido concluidas.

#### · **public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException.**

En este metodo se encuentra la mayor parte de la funcionalidad del servlet. Cada vez que se realice una petición se llamará a este metodo pasándole dos parámetros que nos permite obtener información de la petición y un flujo de salida para escribir la respuesta.

#### **COMUNICACIÓN CON EL CLIENTE (USUARIO).**

Como se explicó anteriormente, los Servlets se han usado más en la generación de paginas web dinámicas, y cuando se habla de dinamismo se refiere a dinamismo en la información *no en la interfaz*. En la figura siguiente se muestra como se establece la comunicación entre el Cliente y el Servidor, en un ambiente de Web, por medio de la Tecnología de *Java Servlets*.





## PROGRAMACION CON JAVA SERVLETS.

### Interacción con el Cliente

Cuando un servlet acepta una llamada de un cliente, recibe dos objetos:

- \_ Un **ServletRequest**, que encapsula la comunicación desde el cliente al servidor.
- \_ Un **ServletResponse**, que encapsula la comunicación de vuelta desde el servlet hacia el cliente.

**ServletRequest** y **ServletResponse** son interfaces definidos en el paquete **javax.servlet**.

#### El Interface ServletRequest

El Interface **ServletRequest** permite al servlet acceder a:

Información como los nombres de los parámetros pasados por el cliente, el protocolo (esquema) que está siendo utilizado por el cliente, y los nombres del host remote que ha realizado la petición y la del server que la ha recibido.

El stream de entrada, **ServletInputStream**. Los Servlets utilizan este stream para obtener los datos desde los clientes que utilizan protocolos como los métodos POST y PUT del HTTP.

Los interfaces que extienden el interface **ServletRequest** permiten al servlet recibir más datos específicos del protocolo. Por ejemplo, el interface **HttpServletRequest** contiene métodos para acceder a información de cabecera específica HTTP.

#### El Interface ServletResponse

El Interface **ServletResponse** le da al servlet los métodos para responder al cliente.

Permite al servlet seleccionar la longitud del contenido y el tipo MIME de la respuesta.

Proporciona un stream de salida, **ServletOutputStream**, y un **Writer** a través del cual el servlet puede responder datos.

Los interfaces que extienden el interface **ServletResponse** le dan a los servlets más capacidades específicas del protocolo. Por ejemplo, el interface **HttpServletResponse** contiene métodos que permiten al servlet manipular información de cabecera específica HTTP.

Un Servlet HTTP maneja peticiones del cliente a través de su método **service**. Este método soporta peticiones estándar de cliente HTTP despachando cada petición a un método designado para manejar esa petición. Por ejemplo, el método **service** llama al método **doGet** mostrado en el siguiente ejemplo:

```
public class SimpleServlet extends HttpServlet
{
/**
 * Maneja el método GET de HTTP para construir una sencilla página Web.
 */
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
    PrintWriter out;
    String title = "Salida de un Servlet Sencillo";
    // primero selecciona el tipo de contenidos y otros campos de cabecera de la respuesta
    response.setContentType("text/html");
    // Luego escribe los datos de la respuesta
    out = response.getWriter();
    out.println("<HTML><HEAD><TITLE>");
    out.println(title);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + title + "</H1>");
    out.println("<P>This is output from SimpleServlet.");
    out.println("</BODY></HTML>");
    out.close();
} //Fin del Método doGet
} //Fin de la clase SimpleServlet
```

Del ejemplo anterior, **SimpleServlet** extiende la clase **HttpServlet**, que implementa el interface **Servlet**.

**SimpleServlet** sobrescribe el método **doGet** de la clase **HttpServlet**. Este método es llamado cuando un cliente hace un petición GET (el método de petición por defecto de HTTP), y resulta en una sencilla página HTML devuelta al cliente.

#### **Dentro del método doGet**

- \_ La petición del usuario está representada por un objeto **HttpServletRequest**.
- \_ La respuesta al usuario esta representada por un objeto **HttpServletResponse**.

Como el texto es devuelto al cliente, el respuesta se envía utilizando el objeto **Writer** obtenido desde el objeto **HttpServletResponse**.

#### **Peticiones y Respuestas**

Como se explico en el apartado anterior los métodos de la clase **HttpServlet** que manejan peticiones de cliente toman dos argumentos:

- \_ Un objeto **HttpServletRequest**, que encapsula los datos *desde* el cliente.
- \_ Un objeto **HttpServletResponse**, que encapsula la respuesta *hacia* el cliente.

#### **Objetos HttpServletRequest**

Un objeto **HttpServletRequest** proporciona acceso a los datos de cabecera HTTP, como cualquier cookie encontrada en la petición, y el método HTTP con el que se ha realizado la petición. El objeto **HttpServletRequest** también permite obtener los argumentos que el cliente envía como parte de la petición.

#### *Para acceder a los datos del cliente*

El método **getParameter** devuelve el valor de un parámetro nombrado. Si nuestro parámetro pudiera tener más de un valor, deberíamos utilizar **getParameterValues** en su lugar. El método **getParameterValues** devuelve un array de valores del parámetro nombrado. (El método **getParameterNames** proporciona los nombres de los parámetros.

#### **Manejar Peticiones GET y POST**

Para manejar peticiones HTTP en un servlet, extendemos la clase **HttpServlet** y sobrescribimos los métodos del servlet que manejan las peticiones HTTP que queremos soportar. Este apartado ilustra el manejo de peticiones GET y POST. Los métodos que manejan estas peticiones son **doGet** y **doPost**.

#### **\_ Manejar Peticiones GET**

Manejar peticiones GET implica sobrescribir el método **doGet**. El siguiente ejemplo muestra a **BookDetailServlet** haciendo esto.

Los métodos explicados en Peticiones y Respuestas se muestran en **negrita**:

```
public class BookDetailServlet extends HttpServlet
{
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // selecciona el tipo de contenido en la cabecera antes de acceder a Writer
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Luego escribe la respuesta
        out.println("<html>" +
            "<head><title>Book Description</title></head>" +
            "...");
        //Obtiene el identificador del libro a mostrar
        String bookId = request.getParameter("bookId");
```

```
if (bookId != null)
{
// Obtiene la información sobre el libro y la imprime
...
}
out.println("</body></html>");
out.close();
}
... }
```

### **\_ Manejar Peticiones POST**

Manejar peticiones POST implica sobreescibir el método **doPost**. El siguiente ejemplo muestra a ReceiptServlet haciendo esto.

Nuevamente, los métodos explicados en Peticiones y Respuestas se muestran en **negrita**:

```
public class ReceiptServlet extends HttpServlet
{
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
...
// selecciona la cabecera de tipo de contenido antes de acceder a Writer
response.setContentType("text/html");
PrintWriter out = response.getWriter();
// Luego escribe la respuesta
out.println("<html>" +
"<head><title> Receipt </title>" +
...);
out.println("<h3>Thank you for purchasing your books from us " +
request.getParameter("cardname") +
...);
out.close();
}
...
}
```

El servlet extiende la clase **HttpServlet** y sobreescibe el método **doPost**. Dentro del método **doPost**, el método **getParameter** obtiene los argumentos esperados por el servlet.

Para responder al cliente, el método **doPost** utiliza un **Writer** del objeto **HttpServletResponse** para devolver datos en formato texto al cliente. Antes de acceder al writer, el ejemplo selecciona la cabecera del tipo de contenido. Al final del método **doPost**, después de haber enviado la respuesta, el **Writer** se cierra.

### **Manejar Datos de Formularios**

Si alguna vez has usado un motor de búsqueda Web, visitado un tienda de libros on-line, etc., probablemente habrás encontrado URLs de búsqueda con varios parámetros como:

**<http://host/path?user=Marty+Hall&origin=bwi&dest=lax>**.

La parte posterior a la interrogación (**user=Marty+Hall&origin=bwi&dest=lax**) es conocida como *datos de formulario*, y es la forma más común de obtener datos desde una página Web para un programa del lado del servidor. Puede añadirse al final de la URL después de la interrogación (como arriba) para peticiones **GET** o enviada al servidor en una línea separada, para peticiones **POST**.

Extraer la información necesaria desde estos datos de formulario es tradicionalmente una de las partes más tediosas de la programación CGI.

1. Primero de todo, tenemos que leer los datos de una forma para las peticiones **GET** (en CGI tradicional, esto se hace

mediante **QUERY\_STRING**), y de otra forma para peticiones **POST** (normalmente leyendo la entrada estándar).

2. Segundo, tenemos que separar las parejas de los ampersands (&), luego separar los nombres de los parámetros (a la izquierda de los signos igual) del valor del parámetro (a la derecha de los signos igual).

3. Tercero, tenemos que decodificar los valores. Los valores alfanuméricos no cambian, pero los espacios son convertidos a signos más y otros caracteres se convierten como %XX donde XX es el valor ASCII (o ISO Latin-1) del carácter, en hexadecimal.

Por ejemplo, si alguien introduce un valor de "*~hall, ~gates, y ~mcnealy*" en un campo de texto con el nombre "users" en un formulario HTML, los datos serían enviados como **"users=%7Ehall%2C+%7Egates%2C+y+%7Emcnealy"**.

4. Finalmente, la cuarta razón que hace que el análisis de los datos de formulario sea tedioso es que los valores pueden ser omitidos (por ejemplo, **param1=val1&param2=&param3=val3**) y un parámetro puede tener más de un valor y que el mismo parámetro puede aparecer más de una vez (por ejemplo: **param1=val1&param2=val2&param1=val3**).

Una de las mejores características de los servlets Java es que todos estos análisis de formularios son manejados automáticamente.

Simplemente llamamos al método **getParameter** de **HttpServletRequest**, y suministramos el nombre del parámetro como un argumento. Observa que los nombres de parámetros son sensibles a la mayúsculas. Hacemos esto exactamente igual que cuando los datos son enviados mediante **GET** o como si los enviáramos mediante **POST**. El valor de retorno es un **String** correspondiente al valor de la primera ocurrencia del parámetro. Se devuelve un **String** vacío si el parámetro existe pero no tiene valor, y se devuelve **null** si no existe dicho parámetro. Si el parámetro pudiera tener más de un valor, como en el ejemplo anterior, deberíamos llamar a **getParameterValues** en vez de a **getParameter**. Este devuelve un array de strings. Finalmente, aunque en aplicaciones reales nuestros servlets probablemente tengan un conjunto específico de nombres de parámetros por los que buscar. Usamos **getParameterNames** para esto, que devuelve una **Enumeration**, cada entrada puede ser forzada a **String** y usada en una llamada a **getParameter**.

#### **Ejemplo: Leer Tres Parámetros**

Aquí hay un sencillo ejemplo que lee tres parámetros llamados **param1**, **param2**, y **param3**, listando sus valores en una lista marcada.

Observamos que, aunque tenemos que especificar selecciones de respuesta (content type, status line, otras cabeceras HTTP) antes de empezar a generar el contenido, no es necesario que leamos los parámetros de petición en un orden particular.

También observamos que podemos crear fácilmente servlets que puedan manejar datos **GET** y **POST**, simplemente haciendo que su método **doPost** llame a **doGet** o sobrescribiendo **service** (que llama a **doGet**, **doPost**, **doHead**, etc.). Esta es una buena práctica estándar, ya que requiere muy poco trabajo extra y permite flexibilidad en el lado del cliente.

#### **TresParametros.java**

Nota: este ejemplo también usa la clase Utilidad.java.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
public class TresParametros extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    { response.setContentType("text/html");
```

```
PrintWriter out = response.getWriter();
String title = "Lectura de Tres parámetros";
out.println(Utilidad.headConTitle(title) +
"<BODY>\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<UL>\n" +
" <LI>param1: "
+ request.getParameter("param1") + "\n" +
" <LI>param2: "
+ request.getParameter("param2") + "\n" +
" <LI>param3: "
+ request.getParameter("param3") + "\n" +
"</UL>\n" +
"</BODY></HTML>");
}
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
doGet(request, response);
}
}
```



## Guía Práctica No 9 Servlets Básico



### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Conocer la jerarquía de clases y paquetes utilizados para la implementación de Servlets de Java.
- Conocer la estructura básica de programas Java Servlets.
- Utilizar HTML en programas de Java.
- Crear formularios en HTML que envíen parámetros a servlets de java.
- Crear Clases de Java servlets que utilicen los métodos en envío y recepción de información (doGet y doPost)

### PROCEDIMIENTO

#### DIFERENCIAS ENTRE LAS TECNOLOGÍAS CGI Y SERVLET

La tecnología **Servlet** proporciona las mismas ventajas del lenguaje **Java** en cuanto a **portabilidad** ("write once, run anywhere") y **seguridad**, ya que un **servlet** es una **clase** de **Java** igual que cualquier otra, y por tanto tiene en ese sentido todas las características del lenguaje. Esto es algo de lo que carecen los **programas CGI**, ya que hay que compilarlos para el sistema operativo del servidor y no disponen en muchos casos de técnicas de comprobación dinámica de errores en tiempo de ejecución.

Otra de las principales ventajas de los **servlets** con respecto a los **programas CGI**, es la del rendimiento, y esto a pesar de que **Java** no es un lenguaje particularmente rápido. Mientras que los es necesario cargar los **programas CGI** tantas veces como peticiones de servicio existan por parte de los clientes, los **servlets**, una vez que son llamados por primera vez, **quedan activos en la memoria del servidor hasta que el programa que controla el servidor los desactiva**. De esta manera se minimiza en gran medida el tiempo de respuesta.

El **HttpServletRequest** tiene métodos que nos permiten encontrar información entrante como datos de un FORM, cabeceras de petición HTTP, etc. El **HttpServletResponse** tiene métodos que nos permiten especificar líneas de respuesta HTTP (200, 404, etc.), cabeceras de respuesta (Content-Type, Set-Cookie, etc.), y, todavía más importante, nos permiten obtener un **PrintWriter** usado para enviar la salida de vuelta al cliente. Para servlets sencillos, la mayoría del esfuerzo se gasta en sentencias `println` que generan la página deseada. Tenemos que importar las clases de los paquetes `java.io` (para `PrintWriter`, etc.), `javax.servlet` (para `HttpServletRequest`, etc.), y `javax.servlet.http` (para `HttpServletRequest` y `HttpServletResponse`).

#### Un Sencillo Servlet que Genera Texto Normal

Aquí tenemos un servlet que sólo genera texto normal. La siguiente sección mostrará el caso más usual donde se generará HTML.

##### Ejemplo1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Ejemplo1 extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
```

```
out.println("Este es mi Primer Servlet en Programación IV");
}
}
```

### Un Servlet que Genera HTML

La mayoría de los servlets generan HTML, no texto normal como el ejemplo anterior. Para hacer esto, necesitamos dos pasos adicionales, decirle al navegador que estamos devolviendo HTML. y modificar la sentencia `println` para construir una página Web legal.

El primer paso se hace configurando la cabecera de respuesta **Content-Type**. En general, las cabeceras de respuesta se configuran mediante el método **setHeader** de **HttpServletResponse**, pero seleccionar el tipo de contenido es una tarea muy común y por eso tiene un método especial **setContentType** sólo para este propósito. Observa que necesitamos configurar las cabeceras de respuesta antes, de devolver algún contenido mediante **PrintWriter**. Aquí hay un ejemplo:

#### Ejemplo2.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Ejemplo2 extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"Transitional//EN\">\n" +
            "<HTML>\n" +
            "<HEAD><TITLE>Programación IV</TITLE></HEAD>\n" +
            "<BODY>\n" +
            "<center><H1>Ejemplo No 2 de Java Servlets</H1></center>\n" +
            "</BODY></HTML>");
    }
}
```

La línea DOCTYPE es técnicamente requerida por la especificación HTML, y aunque la mayoría de los navegadores Web la ignoran, es muy útil cuando se envían páginas a validadores de formato HTML. Estos validadores comparan la sintaxis HTML de las páginas comparándolas con la especificación formal del HTML, y usan la línea DOCTYPE para determinar la versión de HTML con la que comparar.

En muchas páginas web, la línea HEAD no contiene nada más que el TITLE, aunque los desarrolladores avanzados podrían querer incluir etiquetas META y hojas de estilo. Pero para el caso sencillo, crearemos un método que crea un título y devuelve las entradas DOCTYPE, HEAD, y TITLE como salida. Aquí está el código:

#### Utilidad.java

```
public class Utilidad
{
    public static final String DOCTYPE = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
    Transitional//EN\">";
    public static String headConTitle(String title)
    {
        return(DOCTYPE + "\n" +
            "<HTML>\n" +
```

```
"<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
}
}
```

Aquí tenemos un nuevo ejemplo que instancia a la Clase Utilidad:

### Ejemplo3.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Ejemplo3 extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println(Utilidad.headConTitle("Ejemplo Utilizando dos Clases") +
            "<BODY>\n" +
            "<H1>Este es el Ejemplo No. 3</H1>\n" +
            "</BODY></HTML>");
    }
}
```

### METODO SERVICE.

**public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException.**

En este método se encuentra la mayor parte de la funcionalidad del servlet. Cada vez que se realice una petición se llamará a este método pasándole dos parámetros que nos permite obtener información de la petición y un flujo de salida para escribir la respuesta.

Análogamente tenemos otra serie de métodos que realizan la implementación de respuesta a métodos de comunicación del protocolo http 1.1 como son GET y POST. Estos son respectivamente:

– **public void doGet(HttpServletRequest request, HttpServletResponse response)**  
 – **public void doPost(HttpServletRequest request, HttpServletResponse response)**

Los dos parámetros que recibe **service()** son esenciales para el funcionamiento del servlet por lo que pasaremos a verlos con mas profundidad:

Los dos parámetros que recibe **service()** son **HttpServletRequest** y **HttpServletResponse**

### HttpServletRequest

Esta interfaz derivada de **ServletRequest** proporciona los métodos para recuperar la información de la petición del usuario así como del propio usuario. Señalaremos los más importantes:

– **public abstract String getRemoteHost().** Devuelve el nombre del ordenador que realizó la petición  
 – **public abstract String getParameter(String parameter).** Devuelve el valor del parámetro *parameter* o null si dicho parámetro no existe.  
 – **public abstract String[] getParameterValues(String parameter).** Devuelve un array con los valores del parámetro especificado por *parameter* o null si dicho parámetro no existe.  
 – **public abstract Enumeration getParameterNames().** Devuelve una Enumeration de los nombres de los parámetros empleados en la petición.



**HttpServletResponse**

Se trata de un interfaz derivada de `ServletResponse` que proporciona los métodos para realizar la respuesta al cliente que originó la petición. Señalaremos los más importantes:

**public abstract PrintWriter getWriter().** Permite obtener un objeto **PrintWriter** para escribir la respuesta.

**public abstract setContentType(String).** Permite establecer el tipo MIME de la respuesta

**EJEMPLOS DE SERVLETS CON LECTURA DE PARAMETROS**

A continuación realizaremos un sencillo de ejemplo de un servlet que recibirá como parámetro un nombre y saludará al cliente que realizó la petición. Para ello construiremos una página web con un formulario que nos servirá para enviar la petición al servlet.

```
<html>
<head> <title>Ejemplo de servlet con Parametros</title> </head>
<body>
<h1>Introduzca su nombre y pulse el botón de enviar</h1><hr>
<FORM ACTION="/servlet/HolaServlet" METHOD="post">
Nombre:<INPUT TYPE="text" NAME="nombre" size="30">
<INPUT TYPE="submit" NAME="enviar" VALUE="Enviar">
</form>
</body>
</html>
```

A continuación se muestra el código del servlet. Este código fuente se compilaría y se situaría en el directorio configurado en el servidor web para la ejecución de servlets(en nuestro caso sera /servlet):

**HolaServlet.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaServlet extends HttpServlet
{
    /*
    * En este caso se ha optado por redefinir el metodo doPost(), pudiéndose
    * igualmente haberse optado por redefinir service().Lo que seria incorrecto
    * es redefinir doGet() ya que la petición se realizará por el método post
    */
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        //Se obtiene el valor del parametro enviado
        String name = request.getParameter("nombre");
        //Se establece el contenido MIME de la respuesta
        response.setContentType("text/html");
        //Se obtiene un flujo de salida para la respuesta
        PrintWriter out;
        out = response.getWriter();
        //Se escribe la respuesta en HTML estandar
        out.println("<html>");
        out.println("<head>");
        out.println("<title> Respuesta de HolaServlet</title>");
        out.println("<head>");
        out.println("<body>");
        out.println("<h1>El servlet ha generado la pagina de Respuesta</h1><hr>");
        out.println("<br>");
        out.println("<font color='red'>");
        out.println("<h2>Hola " + name + "</h2>");
    }
}
```

```

out.println("</font>");
out.println("</body>");
out.println("</html>");
// Se fuerza la descarga del buffer y Se cierra el canal
out.flush();
out.close();
} //fin doPost()
} //fin clase

```

### Ejemplo 2 usando formularios de HTML.

El formulario contendrá dos campos de tipo TEXT donde el visitante introducirá su **nombre** y **apellidos**. A continuación, deberá indicar la opinión que le merece la página visitada eligiendo una entre tres posibles (**Buena**, **Regular** y **Mala**). Por último, se ofrece al usuario la posibilidad de escribir un **comentario** si así lo considera oportuno.

El código correspondiente a la **página HTML** que contiene este formulario es el siguiente:

```

<HTML>
<HEAD>
<TITLE>Envíe su opinión</TITLE>
</HEAD>
<BODY>
<H2>Por favor, envíenos su opinión acerca de este sitio web</H2>
<FORM ACTION="/servlet/ServletOpinion" METHOD="POST">
Nombre: <INPUT TYPE="TEXT" NAME="nombre" SIZE=15><BR>
Apellidos: <INPUT TYPE="TEXT" NAME="apellidos" SIZE=30><P>
Opinión que le ha merecido este sitio web<BR>
<INPUT TYPE="RADIO" CHECKED NAME="opinion" VALUE="Buena">Buena<BR>
<INPUT TYPE="RADIO" NAME="opinion" VALUE="Regular">Regular<BR>
<INPUT TYPE="RADIO" NAME="opinion" VALUE="Mala">Mala<P>
Comentarios <BR>
<TEXTAREA NAME="comentarios" ROWS=6 COLS=40>
</TEXTAREA><P>
<INPUT TYPE="SUBMIT" NAME="botonEnviar" VALUE="Enviar">
<INPUT TYPE="RESET" NAME="botonLimpiar" VALUE="Limpiar">
</FORM>
</BODY>
</HTML>

```

El servlet que gestionará toda la información del formulario se llamará **ServletOpinion**. Este servlet se limitará a responder al usuario con una página HTML con la información introducida en el formulario, dejando para un posterior apartado el estudio de cómo se almacenarían dichos datos. El código fuente de la clase **ServletOpinion** es el siguiente:

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletOpinion extends HttpServlet
{
// Declaración de variables miembro correspondientes a
// los campos del formulario
private String nombre=null;
private String apellidos=null;
private String opinion=null;
private String comentarios=null;
// Este método se ejecuta una única vez (al ser inicializado el servlet)
// Se suelen inicializar variables y realizar operaciones costosas en
// tiempo de ejecución (abrir ficheros, bases de datos, etc)
public void init(ServletConfig config) throws ServletException

```

```

{
// Llamada al método init() de la superclase (GenericServlet)
// Así se asegura una correcta inicialización del servlet
super.init(config);
System.out.println("Iniciando ServletOpinion...");
} // fin del método init()
// Este método es llamado por el servidor web al "apagarse" (al hacer
// shutdown). Sirve para proporcionar una correcta desconexión de una
// base de datos, cerrar ficheros abiertos, etc.
public void destroy()
{
System.out.println("No hay nada que hacer...");
} //fin del método destroy()
// Método llamado mediante un HTTP POST. Este método se llama
// automáticamente al ejecutar un formulario HTML
public void doPost (HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException
{
// Adquisición de los valores del formulario a través del objeto req
nombre=req.getParameter("nombre");
apellidos=req.getParameter("apellidos");
opinion=req.getParameter("opinion");
comentarios=req.getParameter("comentarios");
// Devolver al usuario una página HTML con los valores adquiridos
devolverPaginaHTML(resp);
} // fin del método doPost()
public void devolverPaginaHTML(HttpServletResponse resp)
throws ServletException, IOException
{
// En primer lugar se establece el tipo de contenido MIME de la respuesta
resp.setContentType("text/html");
// Se obtiene un PrintWriter donde escribir (sólo para mandar texto)
PrintWriter out = null;
out=resp.getWriter();
// Se genera el contenido de la página HTML
out.println("<html>");
out.println("<head>");
out.println("<title>Valores recogidos en el formulario</title>");
out.println("</head>");
out.println("<body>");
out.println("<b><font size=+2>Valores recogidos del ");
out.println("formulario: </font></b>");
out.println("<p><font size=+1><b>Nombre: </b>" + nombre + "</font>");
out.println("<br><fontsize=+1><b>Apellido: </b>" +
apellidos + "</font><b><font size=+1></font></b>");
out.println("<p><font size=+1> <b>Opinión: </b><i>" + opinion +
"</i></font>");
out.println("<br><font size=+1><b>Comentarios: </b>" + comentarios +
"</font>");
out.println("</body>");
out.println("</html>");
// Se fuerza la descarga del buffer y se cierra el PrintWriter,
// liberando recursos de esta forma. IMPORTANTE
out.flush();
out.close();
} // fin de devolverPaginaHTML()
// Función que permite al servidor web obtener una pequeña descripción del
// servlet, qué cometido tiene, nombre del autor, comentarios
// adicionales, etc.

```

```

public String getServletInfo()
{
return "Este servlet lee los datos de un formulario" +
" y los muestra en pantalla";
} // fin del método getServletInfo()
}

```

### Ejemplo 3: Listar todos los Datos del Formulario

Aquí hay un ejemplo que busca **todos** los nombres de parámetros que fueron enviados y los pone en una tabla. Ilumina los parámetros que tienen valor cero así como aquellos que tienen múltiples valores. Primero busca todos los nombres de parámetros mediante el método **getParameterNames** de **HttpServletRequest**. Esto devuelve una **Enumeration**. Luego, pasa por la **Enumeration** de la forma estándar, usando **hasMoreElements** para determinar cuando parar y usando **nextElement** para obtener cada entrada. Como **nextElement** devuelve un **Object**, fuerza el resultado a **String** y los pasa a **getParameterValues**, obteniendo un array de **Strings**. Si este array sólo tiene una entrada y sólo contiene un string vacío, el parámetro no tiene valores, y el servlet genera una entrada "No Value" en itálica. Si el array tiene más de una entrada, el parámetro tiene múltiples valores, y se muestran en una lista bulleteada. De otra forma, el único valor principal se sitúa en la tabla.

### MostrarParametros.java

Nota: este servlet también usa Utilidad.java, mostrado en la guía anterior.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
/** Muestra todos los parámetros vía
 * GET o POST. Especialmente los que no poseen valor o que poseen
 * valores Múltiples.
 */
public class MostrarParametros extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Reading All Request Parameters";
out.println(Utilidad.headConTitle(title) +
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<TABLE BORDER=1 ALIGN=CENTER>\n" +
"<TR BGCOLOR=\"#FFAD00\">\n" +
"<TH>Parameter Name<TH>Parameter Value(s)");
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements())
{
String paramName = (String)paramNames.nextElement();
out.println("<TR><TD>" + paramName + "\n<TD>");
String[] paramValues = request.getParameterValues(paramName);
if (paramValues.length == 1)
{
String paramValue = paramValues[0];
if (paramValue.length() == 0)
out.print("<I>No Value</I>");
else
out.print(paramValue);
}
else

```

```

{
out.println("<UL>");
for(int i=0; i<paramValues.length; i++) {
out.println("<LI>" + paramValues[i]);
}
out.println("</UL>");
} //fin del while
}
out.println("</TABLE>\n</BODY></HTML>");
} //Fin de doGet
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
doGet(request, response);
}
}

```

Aquí tenemos un formulario HTML que envía un número de parámetros a este servlet. Usa **POST** para enviar los datos (como deberían hacerlo *todos* los formularios que tienen entradas **PASSWORD**), demostrando el valor de que los servlets incluyan tanto **doGet** como **doPost**.

#### EnviarParametros.html

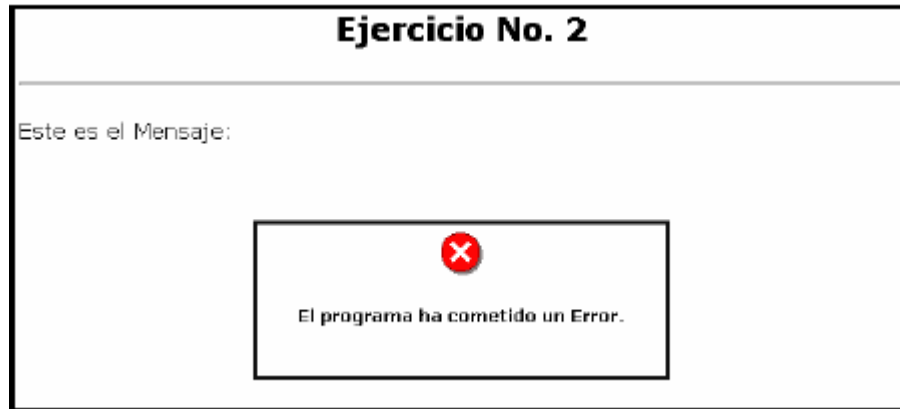
```

<HTML>
<HEAD> <TITLE>A Sample FORM using POST</TITLE> </HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Un Ejemplo de Formulario Utilizando POST</H1>
<hr color="#800000" size="3">
<p>
<FORM ACTION="/servlet/MostrarParametros" METHOD="POST">
Código de Producto: <INPUT TYPE="TEXT" NAME="itemNum" size="20"><BR>
Cantidad: <INPUT TYPE="TEXT" NAME="quantity" size="20"><BR>
Precio Unitario: <INPUT TYPE="TEXT" NAME="price" VALUE="$" size="20"><BR>
<HR>
Nombre: <INPUT TYPE="TEXT" NAME="firstName" size="20"><BR>
Apellido: <INPUT TYPE="TEXT" NAME="lastName" size="20"><BR>
Iniciales: <INPUT TYPE="TEXT" NAME="initial" size="20"><BR>
Dirección: <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
Tarjeta de Crédito:<BR>
<INPUT TYPE="RADIO" NAME="cardType" VALUE="Visa">Visa<BR>
<INPUT TYPE="RADIO" NAME="cardType" VALUE="Master Card">Master Card<BR>
<INPUT TYPE="RADIO" NAME="cardType" VALUE="Amex">American Express<BR>
<INPUT TYPE="RADIO" NAME="cardType" VALUE="Discover">Discover<BR>
<INPUT TYPE="RADIO" NAME="cardType" VALUE="Java SmartCard">Java SmartCard<BR>
Número de Tarjeta de Crédito:
<INPUT TYPE="TEXT" NAME="cardNum" size="20"><BR>
Repita el Número de Tarjeta de Crédito:
<INPUT TYPE="TEXT" NAME="cardNum" size="20"><BR><BR>
<CENTER>
<INPUT TYPE="submit" VALUE="Ordenar">
</CENTER>
</FORM>
</BODY>
</HTML>

```

## EJERCICIOS

1. Crear una clase "Encabezado" cuyos métodos puedan ser Utilizados por otras clases. El objetivo es que la clase Encabezado contenga un método que reciba como parámetros el Título y el Mensaje de Encabezado y pueda generar el HTML respectivo.
2. Escribir una clase "Mensajes" cuyos métodos puedan ser utilizados por otras clases. La clase debe retornar un Mensaje en HTML con una imagen, es decir, poseerá un método que reciba como parámetros el mensaje, y un tipo de mensaje por medio del cual de desplegara una imagen diferente.





## Clase N° 10 Acceso a Base de Datos con Java Servlets



### OBJETIVOS

Al finalizar la clase, el estudiante será capaz de:

- Definir los diferentes medios de Conectividad con Bases de Datos en JAVA
- Identificar como se establece la Conexión por medio de ODBC

### DESARROLLO

La API JDBC es una interfaz de acceso a **RDBMS** (Relational Database Management System) independiente de la plataforma y del gestor de bases de datos utilizado. Se relaciona muy a menudo con el acrónimo ODBC por lo que se suele expresar como **Java Database Connectivity** pero oficialmente, según Javasoft, JDBC no significa nada ni es acrónimo de nada.

El API consiste en una serie de interfaces Java implementadas por un controlador. Este programa de gestión se encarga de la traducción a las llamadas estándar que requiere la base de datos compatible con el. De esta manera el programador puede abstraerse de la programación específica de la base de datos creando código que funcionará para todas los RDBMS que cuenten con un driver JDBC con solo cambiar tal driver.

En la actualidad se encuentran drivers JDBC para todos los sistemas de gestión de bases de datos mas populares(e incluso podríamos decir existentes) como Informix, Oracle, SQL Server, DB2, InterBase, SyBase... y otros productos de índole no comercial como mSql, mySql y PostGreSql, etc.

Aun así existe un tipo especial de drivers denominados puentes JDBC-ODBC que traducen las llamadas en JDBC a llamadas en el estándar de comunicación con bases de datos desarrollado por Microsoft ODBC por lo que en ultimo termino siempre se podrá utilizar uno de estos drivers ya que la totalidad de los sistemas de gestión de bases de datos cuentan con un driver de este ultimo tipo.

#### El paquete java.sql

Consta de una serie de clases e interfaces de las cuales pasaremos a discutir las más importantes:

#### Driver

Se trata de una clase que implementa el controlador JDBC específico de la base de datos y es suministrado por el proveedor de bases de datos. Junto a la clase *DriverManager* permite cargar y descargar los controladores de forma dinámica. El controlador de sirve de una cadena para localizar y acceder a recursos dentro la base de datos con una sintaxis muy parecida a una URL. En todo caso esta cadena será de la forma:

**jdbc:<controlador>://<servidor>:<puerto>/<base de datos>**

Antes de realizar la conexión con la base de datos se debe haber cargado en memoria el controlador para lo que se usa el método estático de la clase **Class.forName(String)**.

#### Connection

Esta interfaz representa una sesión persistente con la base de datos que es devuelta por el Driver. Nos permite utilizar transacciones (si el DBMS lo admite) así como obtener una interfaz para la ejecución de instrucciones SQL.

#### Statement

Esta interfaz se trata de un vehículo para la ejecución de sentencias SQL a la base de datos y la extracción de resultados. A este respecto hay que señalar que JDBC acepta el estándar SQL-92 como mínimo exigible por lo que implementaciones nuevas y/o dependientes del DBMS pueden no estar admitidas.

### **ResultSet**

Representa un conjunto de resultados de forma abstracta (esto es una "tabla"). Dependiendo de su creación permite acceso secuencial o aleatorio y presenta una serie de métodos para obtener información de los resultados y para movernos por el conjunto.

Una vez vistas las clases e interfaces para la gestión de consultas JDBC veremos los pasos a seguir para realizar una consulta a la base de datos. Inicialmente se debe cargar en memoria el controlador JDBC que vayamos a usar:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Esta sentencia hace que la JVM busque en todas las rutas especificadas por el CLASPATH la clase correspondiente al driver y la cargue en memoria de tal manera que este lista para posteriores usos. Seguidamente se debe realizar la conexión con la base de datos:

```
/*
 * Se usa ahora un driver Oracle para acceder a la maquina local y a la
 * tabla Ejemplo:
 */
String Url = "jdbc:oracle://localhost:8080/Ejemplo";
Connection conn = DriverManager.getConnection(url);
```

NOTA: También existen versiones de este último método que permiten realizar la conexión con la BD especificando un nombre de usuario y una contraseña.

Ahora creamos una sentencia para poder interactuar con la BD mediante el uso de SQL:

```
Statement stm = conn.createStatement();
```

Ahora se deberían usar algunos métodos de la interfaz Statement dependientes del tipo de sentencia SQL que queramos realizar:

```
/*
 * La ejecución de la instrucción SQL devuelve resultados
 */
ResultSet rs = stm.executeQuery("SELECT * FROM Ejemplo");
int numRowsUpdated = stm.executeUpdate("INSERT INTO Ejemplo VALUES
(`Pepe`,`Sánchez`,`45598652`));
```

La interfaz ResultSet presenta métodos para obtener un tipo SQL convertido a un tipo Java a partir del nombre de la columna de la forma **getXXX(String nombreColumna)** y se desplaza a través de las filas usando el método boolean.

### **next()**

que desplaza el indicador de posición del ResultSet a la siguiente columna y devuelve un booleano indicando si hay mas filas (inicialmente se encuentra en la primer fila). Las XXX representan algún tipo Java como int, String, float, double...obteniéndose métodos como getInt(string), getString(String),...

Ahora si disponemos de un objeto ResultSet podemos usar sus métodos para desplazarnos por el de la siguiente manera:

```
while(rs.next())
```



```
{
System.out.print(rs.getString("Nombre")+ "-");
System.out.println(rs.getFloat("Sueldo"));
}
```

El método **getString** es invocado sobre el objeto **ResultSet: rs**, por eso **getString** recuperará (obtendrá) el valor almacenado en la columna **Nombre** de la fila actual de **rs**. El valor recuperado por **getString** se ha convertido desde un **VARCHAR** de SQL a un **String** de Java y se podría ser asignado a un objeto **String s**. Observe que como utilizamos la variable **s** en la expresión **print** mostrada arriba, de esta forma:

```
String s = rs.getString("Nombre");
print( s + "- ");
```

La situación es similar con el método **getFloat** excepto en que recupera el valor almacenado en la columna **Sueldo**, que es un **FLOAT** de SQL, y lo convierte a un **float** de Java antes de asignarlo a la variable **n**.

```
float n = rs.getFloat("Sueldo");
print( n );
```

JDBC ofrece dos formas para identificar la columna de la que un método **getXXX** obtiene un valor. Una forma es dar el nombre de la columna, como se ha hecho arriba. La segunda forma es dar el índice de la columna (el número de columna), con un **1** significando la primera columna, un **2** para la segunda, etc. Si utilizáramos el número de columna en vez del nombre de columna el código anterior se podría parecer a esto:

```
String s = rs.getString(1);
float n = rs.getFloat(2);
```

La primera línea de código obtiene el valor de la primera columna de la fila actual de **rs** (columna **Nombre**), convirtiéndolo a un objeto **String** de Java y asignándolo a **s**. La segunda línea de código obtiene el valor de la segunda columna de la fila actual de **rs**, lo convierte a un **float** de Java y lo asigna a **n**. Recuerda que el número de columna se refiere al número de columna en la hoja de resultados no en la tabla original.

En suma, JDBC permite utilizar tanto el nombre como el número de la columna como argumento a un método **getXXX**. Utilizar el número de columna es un poco más eficiente, y hay algunos casos donde es necesario utilizarlo.

JDBC permite muchas lateralidades para utilizar los métodos **getXXX** para obtener diferentes tipos de datos SQL.

Por ejemplo, el método **getInt** puede ser utilizado para recuperar cualquier tipo numérico de caracteres. Los datos recuperados serán convertidos a un **int**; esto es, si el tipo SQL es **VARCHAR**, JDBC intentará convertirlo en un entero. Se recomienda utilizar el método **getInt** sólo para recuperar **INTEGER** de SQL.

### Ejemplo.

El siguiente ejemplo muestra primero una página en HTML, que pide al usuario la introducción de algunos datos.

Estos son enviados a través de un método http a un servlet llamado **Acceso.java** que conecta a la base de datos e introduce los parámetros enviados por el usuario.

### Ejemplodeclase8.htm

```
<HTML>
<HEAD>
<TITLE>Ejemplo de Programación IV</TITLE>
</HEAD>
<BODY>
<H2>Introduzca los siguientes datos:</H2><hr>
```







## Guía Práctica No 10 Bases de Datos con Java Servlets (Uso de Excepciones)



### OBJETIVOS

Al finalizar la Práctica, el estudiante será capaz de:

- Realizar la conectividad a una base de datos utilizando el API JDBC, o por medio de una fuente de datos ODBC.
- Utilizar consultas y Subconsultas de SQL a bases de Datos relacionales.
- Utilizar las excepciones en las Clases de Java Servlets.
- Lanzar y Capturar excepciones por medio de los métodos que proporciona Java para el Manejo de los "Eventos Excepcionales".

### PROCEDIMIENTO

#### **Interfaz de Conexión con el Gestor de Base de Datos.**

Una de las tareas más importantes y más frecuentemente realizadas por los servlets es la conexión a bases de datos mediante JDBC. Esto es debido a que los servlets son un componente ideal para hacer las funciones de capa media en un sistema con una arquitectura de tres capas como la mostrada en la figura siguiente.

Arquitectura cliente-servidor de 3 capas.

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los drivers JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de driver. En cualquier caso, tanto el Servidor HTTP como el Servidor de Base de Datos pueden estar en la misma máquina, aunque en sistemas empresariales de cierta importancia esto no suele ocurrir con frecuencia.

#### **Manejo de Errores utilizando Excepciones**

Existe una regla de oro en el mundo de la programación: en los programas ocurren errores. Esto es sabido. Pero ¿qué sucede realmente después de que ha ocurrido el error? ¿Cómo se maneja el error? ¿Quién lo maneja?, ¿Puede recuperarlo el programa?

El lenguaje Java utiliza **excepciones** para proporcionar capacidades de manejo de errores. En esta guía aprenderás qué es una excepción, cómo lanzar y capturar excepciones, qué hacer con una excepción una vez capturada, y cómo hacer un mejor uso de las excepciones heredadas de las clases proporcionadas por el entorno de desarrollo de Java.

El término excepción es una forma corta de la frase "suceso excepcional" y puede definirse de la siguiente forma:

**Definición:** Una excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias.

#### **Capturar y Manejar Excepciones**

##### **El Bloque try**

El primer paso en la escritura de un manejador de excepciones es poner la sentencia Java dentro de la cual se puede producir la excepción dentro de un bloque try. Se dice que el bloque try gobierna las sentencias encerradas dentro de él y define el ámbito de cualquier manejador de excepciones (establecido por el bloque catch subsecuente) asociado con él.

**Los bloques catch**

Después se debe asociar un manejador de excepciones con un bloque try proporcionándole uno o más bloques catch directamente después del bloque try.

**El bloque finally**

El bloque finally de Java proporciona un mecanismo que permite a sus métodos limpiarse a si mismos sin importar lo que sucede dentro del bloque try. Se utiliza el bloque finally para cerrar ficheros o liberar otros recursos del sistema.

**Capturar y Manejar Excepciones**

Todos los métodos Java utilizan la sentencia **throw** para lanzar una excepción. Esta sentencia requiere un solo argumento, un objeto Throwable. En el sistema Java, los objetos lanzables son ejemplares de la clase Throwable definida en el paquete java.lang. Aquí tienes un ejemplo de la sentencia throw:

**throw algunObjetoThrowable;**

Si se intenta lanzar un objeto que no es 'lanzable', el compilador rehusa la compilación del programa y muestra un mensaje de error similar a éste:

**testing.java:10: Cannot throw class java.lang.Integer; it must be a subclass of class java.lang.Throwable.  
throw new Integer(4);**

**Introducción a SQL (Structured Query Language)**

SQL (Structured Query Language o Lenguaje Estructurado de Consultas) es un lenguaje empleado para crear, manipular, examinar y manejar bases de datos relacionales. Proporciona una serie de sentencias estándar que permiten realizar las tareas antes descritas. SQL fue estandarizado según las normas ANSI (American National Standards Institute) en 1992, paliando de alguna forma la incompatibilidad de los productos de los distintos fabricantes de bases de datos (Oracle, Sybase, Microsoft, Informix, etc.). Esto quiere decir que una misma sentencia permite a priori manipular los datos recogidos en cualquier base de datos que soporte el estándar ANSI, con independencia del tipo de base de datos.

La mayoría de los programas de base de datos más populares soportan el estándar SQL-92, y adicionalmente proporcionan extensiones al mismo, aunque éstas ya no están estandarizadas y son propias de cada fabricante.

JDBC soporta el estándar ANSI SQL-92 y exige que cualquier driver JDBC sea compatible con dicho estándar.

Para poder enviar sentencias SQL a una base de datos, es preciso que un programa escrito en Java esté previamente conectado a dicha base de datos, y que haya un objeto Statement disponible.

**REGLAS SINTÁCTICAS**

SQL tiene su propia sintaxis que hay que tener en cuenta, pues a veces puede ocurrir que sin producirse ningún problema en la compilación, al tratar de ejecutar una sentencia se produzca algún error debido a una incorrecta sintaxis en la sentencia. Por tanto, será necesario seguir las siguientes normas:

SQL no es sensible a los espacios en blanco<sup>1</sup>. Los retornos de carro, tabuladores y espacios en blanco no tienen ningún significado especial. Las palabras clave y comandos están delimitados por comas (,), y cuando sea necesario, debe emplearse el paréntesis para agruparlos.

Las consultas son insensibles a mayúsculas y minúsculas. Sin embargo, los valores almacenados en las bases de datos sí que son sensibles a las mismas, por lo que habrá que tener cuidado al introducir valores, efectuar comparaciones, etc.

A la hora de introducir un String, éste deberá ir encerrado entre comillas simples, ya que de lo contrario se producirán errores en la ejecución.

### **Ejemplo Práctico Utilizando Conexión a Bases de Datos y Manejo de Excepciones.**

El siguiente archivo HTML, llama a la clase servlet "ListaAlumnos.java", que mostrará un listado de los alumnos pertenecientes a un grupo específico elegido por el usuario en el siguiente formulario.

```
<!-- fichero Formulario.htm -->
<html>
<head>
<title>Grupos de prácticas</title>
</head>
<body>
<h2 align="center"><font face="Verdana">Escoja el grupo de prácticas cuya lista desea
ver</font></h2>
<hr>
<p><font face="Verdana">Grupos de Estudiantes:</font></p>
<form method="POST" action="/servlet/ListaAlumnos"
name="Formulario">
<p align="center">
<font face="Verdana">
<input type="radio" value="SIS11" checked name="GRUPO">SIS11&nbsp;
<input type="radio" name="GRUPO" value="SIS12">SIS12&nbsp;
<input type="radio" name="GRUPO" value="SIS13">SIS13&nbsp;
<input type="radio" name="GRUPO" value="SIS14">SIS14 </font>
</p>
</p></center>
<div align="center"><center><p>
<font face="Verdana">
<input type="submit" value="Enviar" name="BotonEnviar">
<input type="reset" value="Borrar" name="BotonBorrar"> </font>
</p></center></div>
</form>
</body>
</html>
```

1 Utilice los corchetes para referirse a campos o tablas que están separadas por espacios en blanco por Ej., [Detalle de pedidos]

```
// fichero ListaAlumnos.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;
public class ListaAlumnos extends HttpServlet {
    Connection conn = null;
    // Vector que contendrá los objetos Alumno
    Vector vectorAlumnos=null;
    //Método llamada mediante un HTTP POST
    public void doPost (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        // se establece el tipo de contenido MIME de la respuesta
        resp.setContentType("text/html");
        // se obtiene un PrintWriter donde escribir (sólo para mandar texto)
        PrintWriter out=resp.getWriter();
```

```

// Obtención del grupo de prácticas
String grupo = null;
grupo = req.getParameter("GRUPO");
if(grupo==null) {
resp.sendError(500, "Se ha producido un error en la lectura " +
"de la solicitud");
return;
}
out.println("<html>");
out.println("<head>");
out.println("<title>Lista de alumnos del grupo "+grupo+"</title>");
out.println("</head>");
out.println("<body>");
// url de la base de datos
String url=new String("jdbc:odbc:alumnos");
// Carga del Driver
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(ClassNotFoundException ex) {
out.println("Error al cargar el driver");
out.println(ex.getMessage());
}
// Establecimiento de la conexión
try {
conn=DriverManager.getConnection(url,"","");
}
catch (SQLException sqlEx) {
out.println("Se ha producido un error al establecer "+
"la conexión con: "+url);
out.println(sqlEx.getMessage());
}
// Consulta a la base de datos para obtener la lista de alumnos de un grupo
if(obtenerLista(resp,grupo)==0) {
// Mostrar la lista de alumnos mediante una página HTML
mostrarListaAlumnos(resp, grupo);
}
else if(obtenerLista(resp,grupo)==-3) {
resp.sendError(500, "No se ha encontrado el grupo: " +grupo);
}
else
resp.sendError(500, "Se ha producido un error en el acceso " +
"a la base de datos");
} // fin del método doPost()
public int obtenerLista(HttpServletResponse resp,String grupo)
throws ServletException, IOException {
// se obtiene un PrintWriter donde escribir (sólo para mandar texto)
PrintWriter out=resp.getWriter();
Statement stmt = null;
ResultSet rs = null;
String query = "SELECT DISTINCT Carnet, " +
"Nombre, "+
"Apellidos, "+
"GrupoPractica "+
"FROM TablaAlumnos WHERE GrupoPractica='"+grupo+"'";
// Ejecución del query
try {
stmt=conn.createStatement();
rs=stmt.executeQuery(query);

```

```

vectorAlumnos=new Vector();
// Lectura del ResultSet
// En Java2
while (rs.next()) {
    Alumno temp=new Alumno();
    temp.setNombre(rs.getString("Nombre"));
    temp.setApellidos(rs.getString("Apellidos"));
    temp.setCarnet(rs.getLong("Carnet"));
    vectorAlumnos.addElement(temp);
}
if(vectorAlumnos.size()==0)
    return -3;
return 0;
}
catch (SQLException sql) {
    out.println("Se produjo un error al crear el Statement");
    out.println(sql.getMessage());
    return -1;
} finally {
    // se cierra el Statment
    if(stmt!=null) {
        try {
            stmt.close();
        }
        catch(SQLException e) {
            out.println("Error al cerrar el Statement");
            out.println(e.getMessage());
            return -2;
        }
    }
    // se cierra el Connection
    if(conn!=null) {
        try {
            conn.close();
        }
        catch(SQLException e) {
            out.println("Error al cerrar el Statement");
            out.println(e.getMessage());
            return -2;
        }
    }
} // fin del finally
} // fin del método obtenerLista()
public void mostrarListaAlumnos(HttpServletResponse resp, String grupo)
throws ServletException, IOException {
    // se obtiene un PrintWriter donde escribir (sólo para mandar texto)
    PrintWriter out=resp.getWriter();
    // se manda la lista
    out.println("<H2 align=\"center\">Lista de alumnos del grupo "+
    grupo+"</H2><hr><p>");
    out.println("<div align=\"center\"><center>");
    out.println("");
    out.println("<table border=\"1\" width=\"70%\">");
    out.println("<tr>");
    out.println("<th width=\"25%\" bgcolor=\"#808080\">"+
    "<font color=\"#FFFFFF\">Carnet</font></td>");
    out.println("<th width=\"25%\" bgcolor=\"#808080\">"+
    "<font color=\"#FFFFFF\">Nombre</font></td>");
    out.println("<th width=\"25%\" bgcolor=\"#808080\">"+

```



```

"<font color=\"#FFFFFF\">Apellidos</font></td>");
out.println("</tr>");
// Datos del Alumno por filas
Alumno alum=null;
for (int i=0; i<vectorAlumnos.size();i++) {
alum=(Alumno)vectorAlumnos.elementAt(i);
out.println("<tr>");
out.println("<td width=\"25%\">" +alum.getCarnet()+"</td>");
out.println("<td width=\"25%\">" +alum.getNombre()+"</td>");
out.println("<td width=\"25%\">" +alum.getApellidos()+"</td>");
out.println("</tr>");
}
out.println("</table>");
out.println("</center></div>");
out.println("</body>");
out.println("</html>");
// se fuerza la descarga del buffer y se cierra el PrintWriter
out.flush();
out.close();
} // fin del método mostrarListaAlumnos()
} // fin de la clase ListaAlumnos

```

Puede observarse que este servlet efectúa la conexión con la base de datos cuyo DSN es **alumnos**, y comprueba que la conexión se ha realizado con éxito.

La petición del cliente es de tipo HTTP POST, por lo que se ha redefinido el método doPost(). En este se lee el parámetro GRUPO. En caso de que haya algún problema en la lectura de dicho parámetro, lanza un mensaje de error.

Una vez que se sabe cuál es el grupo cuya lista quiere visualizar el cliente, se llama al método **obtenerLista**, que tiene como uno de sus parámetros precisamente el nombre del grupo a mostrar. En este método se realiza la consulta con la base de datos, mediante el método *executeQuery()* de la interface *Statement*.

En este ejemplo, además, al leer los valores de la base de datos, estos son almacenados en un Vector de objetos de la clase **Alumno2**, que ha sido creada para este ejemplo, y cuyo código puede observarse a continuación.

```

public class Alumno {
// Definición de variables miembro
private String nombre;
private String apellidos;
private long carnet;
private String grupoPractica;
// Métodos para establecer los datos
public void setNombre(String nom) { nombre=nom; }
public void setApellidos(String apel) { apellidos=apel; }
public void setCarnet(long carn) { carnet=carn; }
public void setGrupoPractica(String grupo) { grupoPractica=grupo; }
// Métodos de recuperación de datos
public String getNombre() { return nombre; }
public String getApellidos() { return apellidos; }
public long getCarnet() { return carnet; }
public String getGrupoPractica() { return grupoPractica; }
} // fin de la clase Alumno

```

<b>EJERCICIOS PROPUESTOS.</b>
-------------------------------

1. Crear una interfaz de usuario, utilizando un formulario de HTML, que pida el id de empleado, nombre o Apellido, de tal forma que realice una búsqueda por cualquiera de esos parámetros y muestre la información del empleado o los empleados que coincidan con los parámetros de búsqueda.
2. Crear una interfaz de Usuario, para la búsqueda de productos por categoría (ya sea por Id o por nombre) por medio de un formulario, el servlet de java mostrará el nombre de la categoría seleccionada y el listado de productos (id de producto, Nombre del producto, Nombre del Proveedor, Precio Unitario y Existencia).
3. Crear un formulario que pida un Id de Cliente, y que llame a un servlet que muestre la información del cliente (Nombre de la compañía, Nombre del contacto, Cargo del contacto, Dirección, Teléfono, Fax), y además muestre a parte los Id de Pedidos que ha realizado y la fecha en que los realizó, el id de producto deberá ser un link a otro servlet, que mostrará la información del Pedido realizado por el Cliente (id de pedido, fecha de pedido, la Fecha de entrega) y el detalle de los productos que contiene el pedido, realizando el calculo del total a pagar por el cliente (tomando en cuenta los descuentos).



## Clase Teórica N° 11

### Utilidades para programar en JAVA Servlets



#### OBJETIVOS

Al finalizar, el estudiante será capaz de:

- Definir conceptos sobre la Tecnología de Internet

#### DESARROLLO

##### Equivalentes Servlet a la Variables Estándar CGI

Aunque probablemente tiene más sentido pensar en diferentes fuentes de datos (datos de petición, datos de servidor, etc.) como distintas, los programadores experimentados en CGI podrían encontrar muy útil la siguiente tabla. Asumimos que **request** es el **HttpServletRequest** suministrado a los métodos **doGet** y **doPost**.

**Variable CGI Significado Acceso desde doGet o doPost**

Variable CGI	Significado	Acceso desde doGet o doPost
AUTH_TYPE	Si se suministró una cabecera Authorization, este es el esquema especificado (basic o digest)	request.getAuthType()
CONTENT_LENGTH	Sólo para peticiones POST, el número de bytes enviados.	Técnicamente, el equivalente es String.valueOf(request.getContentLength()) un String) pero probablemente querremos sólo llamar a request.getContentLength(), que devuelve un int.
CONTENT_TYPE	El tipo MIME de los datos adjuntos, si se especifica.	request.getContentType()
DOCUMENT_ROOT	Path al directorio que corresponde con http://host/	getServletContext().getRealPath("/") Observa que era request.getRealPath("/") en especificaciones servlet anteriores.
HTTP_XXX_YYY	Acceso a cabeceras arbitrarias HTTP	request.getHeader("Xxx-Yyy")
PATH_INFO	Información de Path adjunto a la URL. Como los servlets, al contrario que los programas estándares CGI, pueden hablar con el servidor, no necesitan tratar esto de forma separada. La información del path podría ser enviada como parte normal de los datos de formulario.	request.getPathInfo()
PATH_TRANSLATED	La información del path mapeado al path real en el servidor. De nuevo, los servlets no necesitan tener un caso especial para esto.	request.getPathTranslated()
QUERY_STRING	Para peticiones GET, son los datos adjuntos como un gran string, con los valores codificados. Raramente querremos una fila de datos en los servlets; en su lugar, usaremos request.getParameter para acceder a parámetros individuales.	request.getQueryString()
REMOTE_ADDR	La dirección IP del cliente que hizo la petición, por ejemplo "192.9.48.9".	request.getRemoteAddr()
REMOTE_HOST	El nombre de dominio totalmente cualificado (por ejemplo "java.sun.com") del cliente que hizo la petición. Se devuelve la dirección IP si no se puede	request.getRemoteHost()

Variable CGI	Significado	Acceso desde doGet o doPost
	determinar.	
REMOTE_USER	Si se suministró una cabecera Authorization, la parte del usuario.	request.getRemoteUser()
REQUEST_METHOD	El tipo de petición, que normalmente es GET o POST, pero ocasionalmente puede ser HEAD, PUT, DELETE, OPTIONS, o TRACE.	request.getMethod()
SCRIPT_NAME	Path del servlet.	request.getServletPath()
SERVER_NAME	Nombre del Servidor Web.	request.getServerName()
SERVER_PORT	Puerto por el que escucha el servidor.	Técnicamente, el equivalente es String.valueOf(request.getServerPort()), que devuelve un String. Normalmente sólo queremos llamar a request.getServerPort(), que devuelve un int.
SERVER_PROTOCOL	Nombre y versión usada en la línea de petición (por ejemplo HTTP/1.0 o HTTP/1.1).	request.getProtocol()
SERVER_SOFTWARE	Información identificativa del servidor Web.	getServletContext().getServerInfo()

### Ejemplo: Leer las Variables CGI

Aquí tenemos un servlet que crea una tabla que muestra los valores de todas las variables CGI distintas a **HTTP\_XXX\_YYY**, que son sólo cabeceras de petición HTTP que se mostraron en la página anterior.

#### MostrarCGIVariables.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
/** Este servlet crea una tabla que muestra los valores de las variable CGI
 */
public class MostrarCGIVariables extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        //Declaración de una matriz para el almacenamiento de las variables CGI
        String[ ][ ] variables = { { "AUTH_TYPE", request.getAuthType() },
        { "CONTENT_LENGTH", String.valueOf(request.getContentLength()) },
        { "CONTENT_TYPE", request.getContentType() },
        { "DOCUMENT_ROOT", getServletContext().getRealPath("/") },
        { "PATH_INFO", request.getPathInfo() },
        { "PATH_TRANSLATED", request.getPathTranslated() },
        { "QUERY_STRING", request.getQueryString() },
        { "REMOTE_ADDR", request.getRemoteAddr() },
        { "REMOTE_HOST", request.getRemoteHost() },
        { "REMOTE_USER", request.getRemoteUser() },
        { "REQUEST_METHOD", request.getMethod() },
        { "SCRIPT_NAME", request.getServletPath() },
        { "SERVER_NAME", request.getServerName() },
        { "SERVER_PORT", String.valueOf(request.getServerPort()) },
        { "SERVER_PROTOCOL", request.getProtocol() },
        { "SERVER_SOFTWARE", getServletContext().getServerInfo() }
        };
        String title = "Servlet de Ejemplo: Mostrar Variables CGI ";
        out.println(Utilidad.headConTitle(title) +
```

```

"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1><hr><p>\n" +
"<TABLE BORDER=1 ALIGN=CENTER>\n" +
"<TR BGCOLOR=\"#FFAD00\">\n" +
"<TH>Nombre de Variable CGI<TH>Valor");
for(int i=0; i<variables.length; i++) {
String varName = variables[i][0];
String varValue = variables[i][1];
if (varValue == null)
varValue = "<I>No Especificado </I>";
out.println("<TR><TD>" + varName + "<TD>" + varValue);
}
out.println("</TABLE></BODY></HTML>");
}
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
doGet(request, response);
}
}

```

### EL MÉTODO `valueOf()`

Como es conveniente, la clase `String` proporciona un método estático **`valueOf()`**. Se puede utilizar este método para convertir variables de diferentes tipos a un `String`. Por ejemplo, para imprimir el número `pi`:

```
System.out.println(String.valueOf(Math.PI));
```

### Convertir Cadenas a Números

La clase `String` no proporciona ningún método para convertir una cadena en un número. Sin embargo, cuatro clases de los "tipos envolventes" (`Integer`, `Double`, `Float`, y `Long`) proporcionan unos métodos de clase llamados **`valueOf()`** que convierten una cadena en un objeto de ese tipo. Aquí tenemos un pequeño ejemplo del método **`valueOf()`** de la clase `Float`:

```
String piStr = "3.14159";
Float pi = Float.valueOf(piStr);
Métodos Accesores
```

### FraseInversa.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
/** Este servlet crea una tabla que muestra una frase a la Inversa
 */
public class FraseInversa extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Servlet de Ejemplo #2: Uso de toString";
out.println(Utilidad.headConTitle(title) +
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1><hr><p>\n" +
"<TABLE BORDER=1 width=75% ALIGN=CENTER>\n" +
"<TR bgcolor= >\n");
String Fuente= "ESTA ES LA FRASE DE PRUEBA";

```

```

out.println("<td>La frase a la Original es: <b>" + Fuente);
int i, len = Fuente.length();
StringBuffer destino = new StringBuffer(len);
for (i = (len - 1); i >= 0; i--)
{
    destino.append(Fuente.charAt(i));
}
out.println("</b><tr><td>La frase a la inversa es: <b>" + destino.toString());
out.println("</b></TABLE></BODY></HTML>");
}
public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}
}

```

Los métodos utilizados para obtener información de un objeto son conocidos como **métodos accesoros**. La clase **FraseInversa** utiliza dos métodos accesoros de String para obtener información sobre el string **Fuente**.

Primero utiliza el método accesor: **length()** para obtener la longitud de la cadena **Fuente**.

```
int len = Fuente.length();
```

Segundo, utiliza el método accesor: **charAt()** que devuelve el carácter que está situado en la posición indicada en su argumento.

```
Fuente.charAt(i)
```

El carácter devuelto por **charAt()** es el que se añade al StringBuffer **destino**. Como la variable del bucle **i** empieza al final de **Fuente** y avanza hasta el principio de la cadena, los caracteres se añaden en orden inverso al StringBuffer. El método **toString()** convierte un objeto de otro tipo, en este caso stringbuffer a un String.

### MÉTODOS DE HTTPSERVETREQUEST Y HTTPSERVLETRESPONSE

los métodos de clase HttpServlet que puede redefinir el programador reciben como argumentos un objeto HttpServletRequest y otro HttpServletResponse. La interface HttpServletRequest proporciona métodos para obtener información acerca de la petición del cliente, por otro lado, el objeto de la interface HttpServletResponse permite enviar desde el servlet al cliente información acerca del estado del servidor así como establecer los valores del header del mensaje saliente, en las siguientes tablas teneis los métodos más útiles de estas dos clases, también se añaden los métodos de la clase ServletConfig del método init.

Clase HttpServletRequest	
GetCookies()	Devuelve un array de cookies encontradas en la petición
GetDateHeader(String)	Devuelve la fecha de la petición
GetHeader(String)	Devuelve el contenido del header HTTP de la petición
GetMethod()	Devuelve el método de la petición Get, Post, Put, etc
GetRemoteUser()	Devuelve el nombre del usuario que está haciendo la petición
GetRequestedSessionId()	Devuelve el id de sesión de la petición
GetHeaderNames()	Devuelve el nombre del header HTTP

Clase HttpServletResponse	
AddCookie(Cookie)	Sirve para añadir una nueva cookie a la respuesta
ContainsHeader(String)	Verifica si el header HTTP del mensaje de respuesta contiene un campo con el nombre especificado
SendRedirect(String)	Redirige al cliente a la URL especificada
SendError(int)	Envia un error de respuesta al cliente usando el code indicado y un mensaje por defecto
SendError(int,String)	Envia un error de respuesta al cliente usando el code indicado y un mensaje por defecto

### GRAFICOS DE BARRAS UTILIZANDO APPLETS.



El grafico anterior se crea a partir de un applet de java (*Barchart2.class*) Cuyos parámetros son:

```
<applet code base = "DirectorioVirtual" code="Barchart2.class" width=273
height=197 align="left">
<!-- ESTOS SON LOS PARAMETROS GLOBALES PARA EL GRAFICO >
<param name=title value="Aquí va el Titulo del Grafico"> <!-- Titulo del grafico>
<param name=columns value=" n "> <!-- numero de Barras en el Grafico>
<param name=orientation value="vertical"> <!-- orientación horizontal o vertical>
<param name=printval value="yes"> <!-- deseas imprimir los valores de cada barra>
<param name=bgcolor value="f0c0a0"> <!-- color en RGB para el contorno del applet>
<param name=insetcolor value="ffffe0"> <!-- color en RGB para el fondo del grafico de
barras>
<!-- ESTOS PARAMETROS DEFINEN CADA UNA DE LAS BARRAS EN EL GRAFICO >
<param name=c1_style value="striped">
```

```
<param name=c1 value="100">
<param name=c1_color value="blue">
<param name=c1_label value="Q1">
...
...
<param name=cn value="30">
<param name=cn_color value="darkGray">
<param name=cn_label value="Qn">
<param name=cn_style value="solid">
</applet>
```

### DESCRIPCIÓN DE LOS PARAMETROS

El grafico puede ser orientado Horizontal o verticalmente usando *name=orientation value=*, y puede escoger "striped"(rayado) o "solid" (sólido) para cada una de las barras usando *cn\_style* = para la enésima barra.

El ancho de las barras es uniforme, de acuerdo a la escala y al tamaño de la etiqueta. El parámetro "printval" (imprimir valor) ya sea "yes" or "no", y los valores de las barras son escritas al lado de ellas.

Utilice las variables del tag applet, height (altura) y width (ancho) para obtener la dimensión correcta para el grafico (esta puede variar de acuerdo a la orientación que le des al mismo).

Los valores para las barras pueden ser números enteros o reales. Si utiliza notación científica u el exponente utiliza el siguiente formato: 1.546e78, no funcionará, pero si lo escribes 1.546e+78 si lo hará.

El color del fondo del marco del grafico se escoge con el parámetro "bgcolor" y se escribe en 6 dígitos con el formato RGB. Para el caso de los gráficos Verticales, (*orientación = "vertical"*) un segundo color es definido para el rectángulo dentro del grafico también en formato RGB con el parámetro "insetcolor" (el formato de grafico horizontal no utiliza esta definición del segundo color)

Puede escoger entre los siguientes colores permitidos para las barras: red, green, darkGreen, beige, blue, pink, magenta, cyan, white, yellow, gray, and darkGray. Si escoge otro color se imprimira el color por defecto que es el Blue (Azul). Cualquier color puede ser definido para el fondo del grafico y el rectángulo interior, pero no con todas las combinaciones el grafico será legible. Note que los colores de las barras son definidos por el nombre del color opuesto a los colores del fondo y el rectángulo interior del grafico que se especifican en formato RGB.



## Guía Práctica N° 11 Gráficos en Aplicaciones de Java



### OBJETIVOS

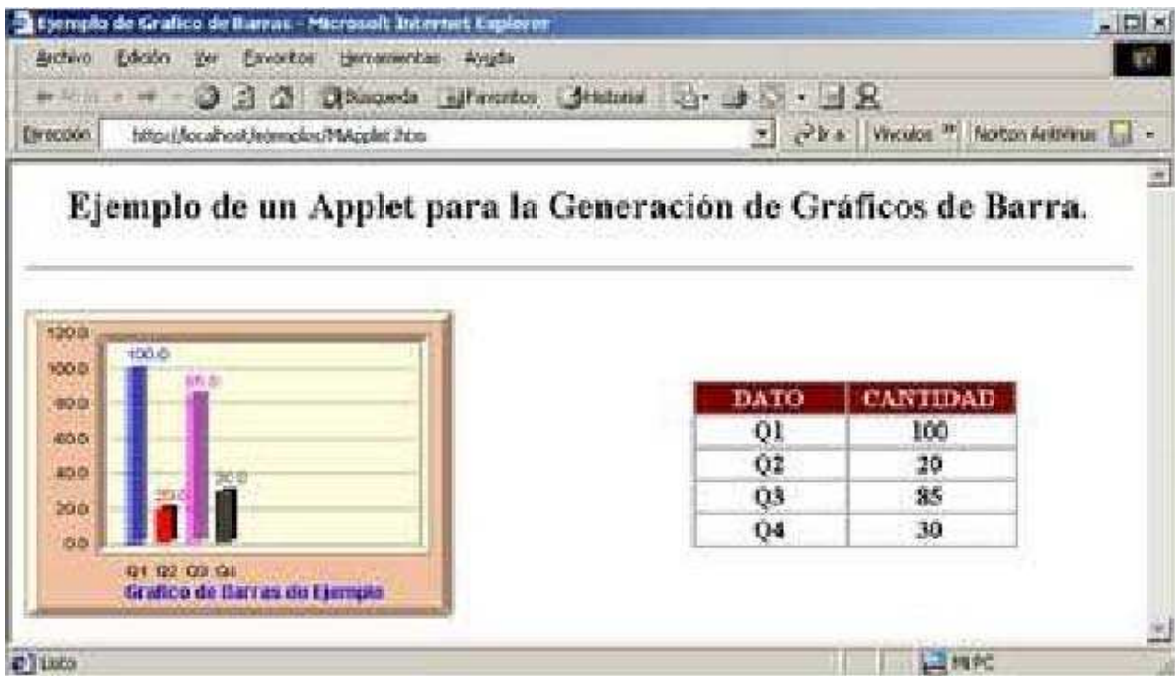
Al finalizar la práctica, el estudiante será capaz de:

- Utilizar clases de java que han sido creadas para la elaboración de graficas de barras.
- Crear Servlets de Java que generen paginas HTML que utilizan el applet para dibujar gráficos de barras.
- Conocer los parámetros que utiliza el applet para la generación de graficas de barra.
- Recuperar información almacenada en bases de datos, mostrarla en tablas de HTML y mostrar el grafico respectivo en pantalla.

### PROCEDIMIENTO

#### Utilizando gráficos de barras en tus páginas.

En tus Páginas HTML puedes utilizar applet3 de Java para la generación de graficas de barras a partir de una tabla de resultados, lo que se visualiza en pantalla puede ser algo parecido a la siguiente figura:



1. Obtener el código binario del archivo **Barchart2.clas4s** y guardarlo en el mismo directorio de sus archivos HTML (si tienes el archivo en un directorio diferente al archivo HTML que lo invoca, deberás agregar la ruta a la especificación del código base del applet tag (como podrás ver más adelante

2. inserte en su archivo HTML un tag de applet con los parámetros apropiados. Por ejemplo, la figura del grafico anterior fue creado por el siguiente código de applet insertado en un archivo HTML normal. Donde los primeros 6 grupos de líneas especifican los parámetros globales del grafico y los siguientes grupos de 4 líneas representan cada parámetro que deberá ser un dato

en el grafico de barras (el parámetro "columns" especifica el numero número de barras que dibuja el grafico):

```
<!-- AQUI SE DECLARA EL TAG DEL APPLET >
<applet code="Barchart2.class" width=273 height=197 align="left">
<!-- ESTOS SON LOS PARAMETROS GLOBALES PARA EL GRAFICO >
<param name=title value="Grafico de Barras de Ejemplo"> <!-- Titulo del grafico>
<param name=columns value="4"> <!-- numero de Barras en el Grafico>
<param name=orientation value="vertical"> <!-- orientación horizontal o vertical>
<param name=printval value="yes"> <!-- deseas imprimir los valores de cada barra>
<param name=bgcolor value="f0c0a0"> <!-- color en RGB para el contorno del applet>
<param name=insetcolor value="ffffe0"> <!-- color en RGB para el fondo del grafico de
barras>
<!-- ESTOS PARAMETROS DEFINEN CADA UNA DE LAS BARRAS EN EL GRAFICO >
<param name=c1_style value="striped">
<param name=c1 value="100">
<param name=c1_color value="blue">
<param name=c1_label value="Q1">
<param name=c2_color value="red">
<param name=c2_label value="Q2">
<param name=c2 value="20">
<param name=c2_style value="solid">
<param name=c3 value="85">
<param name=c3_style value="striped">
<param name=c3_color value="magenta">
<param name=c3_label value="Q3">
<param name=c4 value="30">
<param name=c4_color value="darkGray">
<param name=c4_label value="Q4">
<param name=c4_style value="solid">
</applet>
```

**EL CODIGO COMPLETO DEL EJEMPLO DE LA FIGURA ANTERIOR ES ESTE:**

```
<html>
<head> <title>Ejemplo de Grafico de Barras</title> </head>
<body bgcolor="#ffffff">
<h1 align="center"><font size="5">Ejemplo de un Applet para la Generación de
Gráficos de Barra.</font></h1>
<hr>
<p>
<center>
<table border="0" cellpadding="0" cellspacing="0" style="border-collapse: collapse"
bordercolor="#111111"
width="100%" id="AutoNumber1">
<tr>
<td width="50%">
<applet code="Barchart2.class" width=273 height=197 align="left">
<param name=title value="Grafico de Barras de Ejemplo"> <!-- Titulo del grafico>
<param name=columns value="4"> <!-- numero de Barras en el Grafico>
<param name=orientation value="vertical"> <!-- orientación horizontal o vertical>
<param name=printval value="yes"> <!-- deseas imprimir los valores de cada barra>
<param name=bgcolor value="f0c0a0"> <!-- color en RGB para el contorno del applet>
<param name=insetcolor value="ffffe0"> <!-- color en RGB para el fondo del grafico de barras>
<param name=c1_style value="striped">
<param name=c1 value="100">
<param name=c1_color value="blue">
<param name=c1_label value="Q1">
<param name=c2_color value="red">
<param name=c2_label value="Q2">
```

```

<param name=c2 value="20">
<param name=c2_style value="solid">
<param name=c3 value="85">
<param name=c3_style value="striped">
<param name=c3_color value="magenta">
<param name=c3_label value="Q3">
<param name=c4 value="30">
<param name=c4_color value="darkGray">
<param name=c4_label value="Q4">
<param name=c4_style value="solid">
<center>
</center>
</applet></td>
<td width="50%">
<center>
<table width="208" border="1" cellpadding="0" cellspacing="0">
<tr>
<th width="94" bgcolor="#800000"><font color="#FFFFFF"><b>DATO</b></font></th>
<th width="104" bgcolor="#800000"><font
color="#FFFFFF"><b>CANTIDAD</b></font></th>
</tr>
<tr>
<td width="94" align="center"><b>Q1</b></td>
<td width="104" align="center"><b>100</b></td>
</tr>
<tr>
<td width="94" align="center"><b>Q2</b></td>
<td width="104" align="center"><b>20</b></td>
</tr>
<tr>
<td width="94" align="center"><b>Q3</b></td>
<td width="104" align="center"><b>85</b></td>
</tr>
<tr>
<td width="94" align="center"><b>Q4</b></td>
<td width="104" align="center"><b>30</b></td>
</tr>
</table>
</center>
</div>
</table>
</center>
</body>
</html>

```

### GRAFICAS DE BARRAS UTILIZANDO SERVLETS DE JAVA PARA LA GENERACIÓN DEL CODIGO HTML.

En el siguiente servlet **EjemploGuia12.java** se llenan 2 vectores, cuyos valores son mostrados en una tabla en el navegador, si observas cuando se declara el applet en el servlet debemos decir donde se encuentra la clase **Barchar2**, en este caso se asume que el applet se encuentra en el directorio virtual "**Ejemplo**" (vea la línea en negrita en el código siguiente.)

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class EjemploGuia12 extends HttpServlet {
public void service(HttpServletRequest request,
HttpServletResponse response)

```

```

throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String Sucursales[] = {"SANTA ANA","SANTA TECLA","SAN SALVADOR","SAN MIGUEL"};
int CantidadVenta[] = {250,500,585,175};
out.println("<HTML>\n" +
"<HEAD><TITLE>>Ejercicio de Programación IV GUIA #12</TITLE></HEAD>\n" +
"<BODY>\n" +
"<CENTER><H1>Almacen \"El Baratio\"</H1></CENTER>\n"+
"<CENTER><H3>Cantidad de Bicicletas vendidas en el Año 2001 Por
Sucursales</H3></CENTER><hr>\n"+
"<center><table width=50% border=1 cellpadding=0 cellspacing=0>\n"+
"<tr bgcolor=blue><th><font color=white>SUCURSAL</th><th><font
color=white>CANTIDAD</th>\n");
for (int i=0; i< Sucursales.length; i++)
{
out.println("<tr><td><font face=verdana size=2>"+Sucursales[i]+
"<td><font face=verdana size=2>"+CantidadVenta[i]);
}
out.println("</table></center><p>");
int totalbarras = Sucursales.length;
out.println("<center>");
out.println("<applet codebase=\" /ejemplos\" code=\"Barchart2.class\" width=450
height=310>");
out.println("<param name=title value=\"Ventas realizadas durante el año 2001\">");
out.println("<param name=columns value=\""+totalbarras+"\">");
out.println("<param name=orientation value=\"vertical\">");
out.println("<param name=printval value=\"no\"> ");
out.println("<param name=bgcolor value=\"dddddd\">");
out.println("<param name=insetcolor value=\"ffc0a0\">");
for (int i=0; i< Sucursales.length; i++)
{
int cont=i+1;
out.println("<param name=c"+cont+"_label value=\""+Sucursales[i]+"\">");
out.println("<param name=c"+cont+" value=\""+CantidadVenta[i]+"\">");
out.println("<param name=c"+cont+"_style value=\"striped\">");
out.println("<param name=c"+cont+"_color value=\"red\">");
}
out.println("</applet>");
out.println("</center>");
out.println("</BODY></HTML>");
}
}

```

**EJERCICIOS**

1. Crear una Pagina HTML que muestre la siguiente tabla de resultados, con su grafica respectiva:

SEGUNDO TRIMESTRE DEL AÑO 2002	
PRODUCTO	VENTAS REALIZADAS (EN \$)
IMPRESORA CANNON BJC1000	3,000
DISCOS DUROS (QUANTUM)	5,000
MEMORIA RAM	2,500
UNIDAD DE CD ROM	3,600
MODEM INTERNO	1,200

2. Crear un servlet de java, que muestre el contenido de 2 vectores, en los cuales se encuentran almacenados el número de alumnos inscritos por año desde 1995 hasta el 2002 en el ITCA. Además de mostrar la tabla de resultados hacer un link para mostrar el grafico en la misma pagina.

3. Utilizando la base de datos Neptuno, (cuya conexión ODBC deberá llamarse también Neptuno) Crear un servlet de java, que muestre Cada una de las Categorías de productos que existen en la base de datos con el número total de productos que pertenecen a cada categoría. El servlet deberá mostrar la tabla de resultado y el grafico generado a través de applet



## Bibliografía



### Libros

- Como Programar en JAVA (Deitel y Deitel)Prentice Hall
- Piensa en JAVA-2ª Edicion (Bruce Eckel)Prentice Hall
- Java Servlet Programming Bible (Suresh Rajagopalan, Ramesh Rajamani, Ramesh Krishnaswany and Sridhar Vijendran) Hungry Minds

### Sitos Web

- [www.programacion.com/java/cursos.htm](http://www.programacion.com/java/cursos.htm)
- [www.verextremadura.com/miguel/jsp/JavaServerPages.pdf](http://www.verextremadura.com/miguel/jsp/JavaServerPages.pdf)
- <http://dalila.sip.ucm.es/miembros/olga/javas.html>
- <http://www.aulambra.com/javascript.asp>
- <http://www.programacion.com/html/dinamico/tutorial/indice.htm>



**Primera Edición**  
**Manual de**  
**Aplicaciones Cliente Servidor**

---