JAVA INTERMEDIO

Guía del curso – ICAI

Descripción breve

Al final de este curso el estudiante aprenderá las técnicas básicas de programación Orientada a Objetos, Bases de datos y diseño de aplicaciones Web, mediante el uso de Java como lenguaje de programación.

Autor: Steven R. Brenes Chavarría
Email del autor: steven.brenes.chavarria@una.cr

Introducción al curso

Requisitos

- Conocimiento básico en programación.
- Conocimiento básico de rutinas de programación.
- Conocimiento básico en el uso de alguno de los lenguajes de programación (C, C++, Pascal, C#, VB#, PHP, Cobol, Etc).
- El estudiante debe tener conocimientos generales de matemáticas, haber aprobado el noveno grado.
- Java Básico.

Objetivo General

Al final de este curso el estudiante aprenderá las técnicas básicas de programación Orientada a Objetos, Bases de datos y diseño de aplicaciones Web, mediante el uso de Java como lenguaje de programación.

Objetivos Específicos

- Conocer la terminología básica de ciencias de la computación.
- Analizar, diseñar y desarrollar soluciones algorítmicas a problemas computacionales.
- Plantear, diseñar y probar soluciones computacionales a situaciones reales basadas en el Paradigma de Orientación a Objetos y aplicar los aspectos básicos de este paradigma.
- Conocer y aplicar patrones de diseño de orientación de Objetos.
- Conocer y programar aplicaciones gráficas, orientadas a eventos.
- Conocer los principios de las bases de datos relacionales
- Conocer y programar aplicaciones con conexión a bases de datos
- Conocer y programar aplicaciones Web

Tabla de contenido

Capítulo 1
Temas introductorios al curso

En el presente capítulo se trataran temas, referentes a como diseñar y distribuir una aplicación Java, para tener las mejores prácticas en un diseño de un software de mediana a gran escala. Supongamos un sistema de gran escala, en donde todo el código relacionado con la presentación, validación, reglas de negocios, conexión a bases de datos y demás niveles se encuentre programado en un solo nivel; queda claro que esta práctica dificulta considerablemente la legibilidad, portabilidad, mantenibilidad e incluso la seguridad del código.

Esta es la razón principal por la que los programadores deciden utilizar el Modelo Vista Controlador, con el fin de facilitar el trabajo de programación.

Introducción al MVC

Unos de los modelos de desarrollo de software más utilizados es el denominado Modelo Vista Controlador (MVC). La idea detrás de este modelo es separar el código en diferentes capas o niveles.

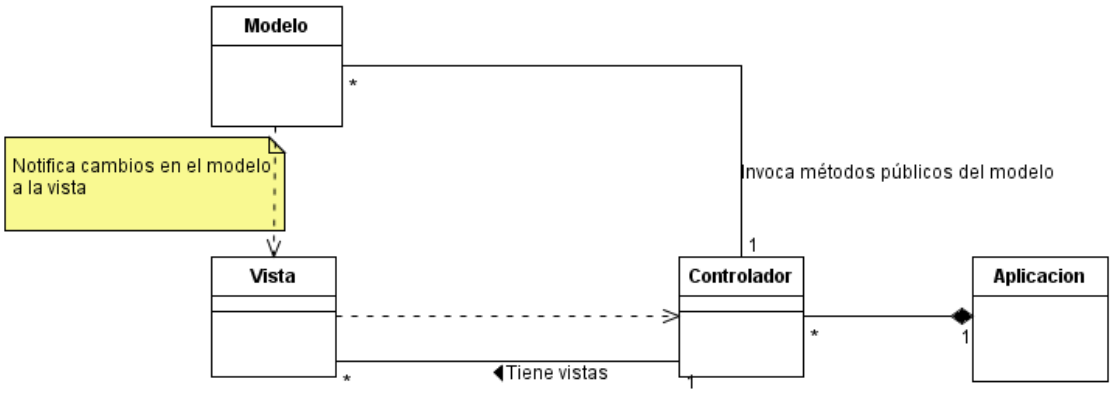


Ilustración MVC 1

El modelo MVC se puede programar en compañía con otros patrones de diseño como el patrón Observador, permitiéndole extender la facilidad de notificarle a toda la aplicación cambios en el estado del modelo. MVC resulta realmente importante para desarrollar aplicaciones que tengan un nivel de complejidad de mediano a gran sistema. El patrón tiene tres componentes principales:

El modelo representa las reglas del negocio, es decir los algoritmos y procesos que tiene que llevarse a cabo para que la aplicación funcione adecuadamente. Pongamos los siguientes ejemplos: se cuenta con un sistema de facturación, el modelo de la aplicación deberá ser capaz de realizar la suma de los productos y almacenarlos persistentemente en la computadora. Observe que el modelo no es el responsable de invocarle la acción ni de mostrar en el monitor la suma, el modelo solamente es responsable de realizar los cálculos necesarios para obtener la suma que debe pagar el cliente.

El controlador es la clase responsable, de ser intermediario entre la vista y el modelo. Cuando una operación de la vista se ejecuta es tarea del controlador invocar a los métodos del modelo. Luego el modelo notifica el cambio al controlador para que este se encargue de actualizarlo en la vista.

La vista es la encargada de leer y mostrar los datos del modelo, continuemos con el ejemplo de la venta de productos de una tienda, la vista deberá ser capaz de mostrarle por monitor las opciones para que el dependiente ingrese la lista de los productos. Una vez que esta operación se da, es tarea de la vista notificarle al controlador los cambios realizados.

Proceso de invocación en el VMC

En los siguientes pasos se detallan como es el accionar del MVC en términos prácticos, para ello seguiremos el ejemplo de una tienda en donde un cliente le da al dependiente una serie de productos para realizar la compra.

1. El dependiente interactúa con la interfaz de usuario (**vista**) de alguna forma, esta vista puede ser una página web, un formulario o un evento como pulsar un botón.

2. La vista una vez se han cargado todos los códigos de barras de los productos que se quieren facturar, debe invocar al **controlador** la solicitud para procesar este evento.

3. El controlador accede al **modelo**, para enviarle el vector de productos. El modelo deberá entonces realizar los procesos básicos de las reglas de negocio. De esta forma el modelo calcula el total que debe pagar, así como los impuestos, generar el identificador de la factura. También podría ser labor del modelo guardar permanentemente la información de los productos, ya sea por medio de un archivo o mediante Bases de Datos.

4. Finalmente el modelo, delega a las vistas la tarea de actualizar la información, esto podría ser por ejemplo mostrar una ventana con los datos de la factura o en un defecto imprimir la factura físicamente. El modelo no debe tener conocimiento directo de la vista, la excepción a esta regla es el uso del patrón **observador** que permite invocar simultáneamente a muchas vistas. El patrón observador no se puede utilizar en aplicaciones Web, puesto que las vistas (paginas HTML) son construidas por demanda, además de no tener referencias a las vistas como si se da en la programación Desktop.

Ventajas de la Arquitectura por Capas

Entre las principales ventajas que provee la arquitectura sobresalen:

1. Separación lógica de los componentes
2. Es posible tener sistemas distribuidos, donde cada host tendrá una capa a su administración (servidores de aplicaciones, servidor de bases de datos, por ejemplo)
3. Mayor seguridad, puesto que el código está disperso
4. Facilidad para mantener una aplicación de mediana a gran escala

Inconvenientes de la arquitectura por Capas

El patrón MVC no tiene desventajas *per se*, puesto que el MVC es un concepto de programación que distribuye el código entre diferentes capas. Lo que si existen son entornos de desarrollo llamados *Frameworks* donde cada uno implementa de forma diferente el MVC.

Es en esta diversidad, donde algunas implementaciones podrían llegar a ser mejores o peores para determinas tareas.

Usos de Hilos en Java

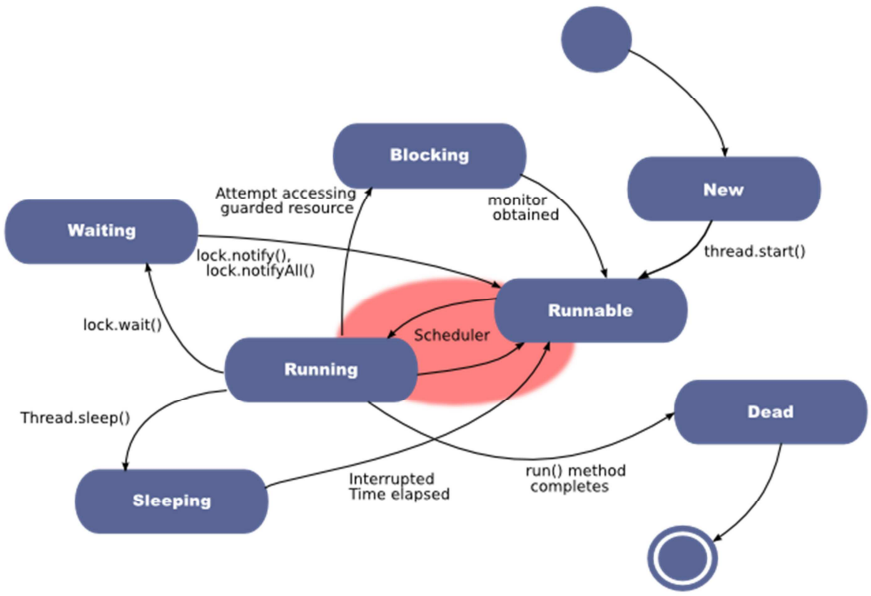
La programación Orientada a Objetos permite modelar el mundo en términos de clases y objetos, con el fin único de administrar la complejidad de los programas. Si comparamos el código fuente de una aplicación escrita en Ensamblador para hacer una sumatoria, y ese mismo algoritmo nos lo llevamos a un lenguaje como Java, es fácil observar que los lenguajes OO permiten administrar la complejidad del código.

Sin embargo, sin importar que se trate de un código escrito en ensamblador, Lisp o Java, en todos los casos se van a ejecutar instrucciones secuenciales. Es decir si tenemos un código como el siguiente:

```
int v1 = sumarPares(10000000000);
int v2 = sumaImpares(10000000000);
```

Se ve ejecutar primero la suma de los pares, una vez que esto termine (que podría ser mucho tiempo si la cantidad de cálculos es muy grande), se inicia el cálculo de la suma de los impares. Esto es un problema si necesitamos que nuestras aplicaciones sean rápidas o que procesen métodos paralelamente.

Es por ello que Java y muchos otros leguajes de programación, ha implementado el concepto de un hilo o thread, en donde cada método podría ejecutarse de forma paralela. Si contamos con una computadora con múltiples procesadores, cada procesador podría encargarse del cálculo de una parte del algoritmo, permitiendo sacarle provecho a la infraestructura de Hardware que tengamos.



Existen muchos estados para un proceso paralelo, en este curso varemos los mas importantes.

El lenguaje de programación Java proporciona soporte para hilos a través de una interfaz y un conjunto de clases. La interfaz de Java y las clases que incluyen funcionalidades sobre hilos son:

- Thread
- Runnable

Todas estas clases son parte del paquete Java.lang. Existen otras clases para controlar la muerte de hilos, cuando estos hilos sufren fallos de sincronización. Este tema queda excluido del curso.

Thread

La clase Thread es la clase responsable de producir hilos para otras clases. Para añadir la funcionalidad de hilo a una clase simplemente se hereda la clase de Thread a la nueva clase. Existe la necesidad de implementar un método run donde el procesamiento de un hilo toma lugar, y a menudo se refieren a él como el cuerpo del hilo. La clase Thread también define los métodos start y stop, los cuales te permiten comenzar y parar la ejecución del hilo.

Runnable

Java no soporta herencia múltiple, es decir, no se puede derivar una clase de varias clases padre, por lo que algunas ocasiones se nos hace imposible heredar de la clase Thread (cuando ya fue heredada por otra clase). En estos casos lo más simple es implementar la interface Runnable, permitiendo la herencia de una clase superior y contar con la posibilidad de tener un hilo en ejecución.

Las clases que implementan la interfaz Runnable proporcionan un método run que es ejecutado por un objeto hilo asociado que es creado aparte. Esta es una herramienta muy útil y a menudo es la única salida que existe incorporar multihilo dentro de las clases.

Creando hilos

Como se explicó anteriormente existen dos formas de implementar los hilos en Java. Vamos a iniciar el ejemplo con una clase llamada Numero, la cual tiene métodos para sumar enteros positivos (pares e impares).

```
class Numero extends Thread {

   public void run() {
     …
   }

}
```

```
public class Numero implements
Runnable {
   private Thread t;

   public void run() {
      …
   }
}
```

Observe las sutilezas entre ambas implementaciones, en el primer caso (columna de la izquierda) se ha heredado completamente la clase Thread a la clase Numero, y se ha sobreescrito el método run propio de la clase Thread. La segunda forma (columna de la

derecha) es mediante la implementación de una clase Interface, en este caso se implementa un método run que eventualmente será ejecutado por la clase Thread.

Estados de un Hilo

El comportamiento de un hilo depende del estado en que se encuentre, este estado define su modo de operación actual, por ejemplo, si está corriendo o no. Los siguientes son estados de un Hilo:

- New
- Runnable
- Not running
- Dead

New

Un hilo está en el estado new la primera vez que se crea y hasta que el método start es llamado. Los hilos en estado new ya han sido inicializados y están listos para empezar a trabajar, pero aún no han sido notificados para que empiecen a realizar su trabajo.

Runnable

Cuando se llama al método start de un hilo nuevo, el método run es invocado y el hilo entra en el estado runnable. Este estado podría llamarse "running" porque la ejecución del método run significa que el hilo está corriendo. Sin embargo, debemos tener en cuenta la prioridad de los hilos. Aunque cada hilo está corriendo desde el punto de vista del usuario, en realidad todos los hilos, excepto el que en estos momentos está utilizando la CPU, están en el estado runnable (ejecutables, listos para correr) en cualquier momento dado.

Not running

El estado not running se aplica a todos los hilos que están parados por alguna razón. Cuando un hilo está en este estado, está listo para ser usado y es capaz de volver al estado runnable en un momento dado. Los hilos pueden pasar al estado not running a través de varias vías.

A continuación se citan diferentes eventos que pueden hacer que un hilo esté parado de modo temporal.

- El método suspend ha sido llamado, se reactiva con le método resume.
- El método sleep ha sido llamado, se debe esperar determinada cantidad de milisegundos.
- El método wait ha sido llamado, debe esperar al método Notify de los objetos.
- El hilo está bloqueado por I/O, se están usado recursos de disco duro, debe esperar que el Sistema Operativo libere este recurso.

Dead

Un hilo entra en estado dead cuando ya no es un objeto necesario. Los hilos en estado dead no pueden ser resucitados y ejecutados de nuevo. Un hilo puede entrar en estado dead a través de dos vías:

- El método run termina su ejecución.
- El método stop es llamado.

La primera opción es el modo natural de que un hilo muera. Uno puede pensar en la muerte de un hilo cuando su método run termina la ejecución como una muerte por causas naturales.

La segunda causa se da con el método stop, en este caso se detiene o aborta el proceso por alguna razón, por ejemplo si se cierra la aplicación, es conveniente detener el Hilo.

Iniciando Hilos

El primer paso es crear una clase que vamos a utilizar como el cuerpo del hilo, en el siguiente código se crea una clase Número que implementa la interface Runnable, cuyo cuerpo del hilo simplemente muestra por consola un contador numérico.

```
package datos;
public class Numero implements Runnable {

    private String id;

    public Numero(String nombre){
        id = nombre;
    }

    @Override
    public void run() {
        for(long i = 0; i < (long)90000000; i++){
            System.out.println(id + " - " + i);
        }
    }
}
```

Lo siguiente es poder crear el hilo, para ello se ejecuta el siguiente código en el main:

```
   public static void main(String[] args) {
        Numero num1 = new Numero("Hilo 1");
        Thread thread = new Thread (num1 , "hilo 1") ;
        thread.start();
   }
```

Este código va crear un hilo contador, lo interesante sería crear muchos hilos y ejecutarlos simultáneamente, para ello implemente el siguiente código en el main:

```
    public static void main(String[] args) throws InterruptedException {
        Numero num1 = new Numero("Hilo 1");
        Numero num2 = new Numero("Hilo 2");
        Numero num3 = new Numero("Hilo 3");
        Numero num4 = new Numero("Hilo 4");
```

```
        Thread thread1 = new Thread (num1 , "hilo 1") ;
        Thread thread2 = new Thread (num2 , "hilo 2") ;
        Thread thread3 = new Thread (num3 , "hilo 3") ;
        Thread thread4 = new Thread (num4 , "hilo 4") ;

        thread1.start();
        Thread.sleep(1000);
        thread2.start();
        Thread.sleep(1000);
        thread3.start();
        Thread.sleep(1000);
        thread4.start();
        Thread.sleep(1000);
    }
```

En este caso en particular, se van a crear cuatro hilos independientes, inicializados un segundos después del otro, es decir se crea el hilo 1 y se espera un segundo para crear el hilo2, así sucesivamente. La ejecución de dicho algoritmo se vería como la siguiente imagen:

```
Hilo 2 - 108289
Hilo 2 - 108290
Hilo 2 - 108291
Hilo 2 - 108292
Hilo 2 - 108293
Hilo 2 - 108294
Hilo 2 - 108295
Hilo 4 - 98342
Hilo 4 - 98343
Hilo 4 - 98344
Hilo 4 - 98345
Hilo 4 - 98346
Hilo 4 - 98347
```

El tiempo que un hilo este activo va depender del tiempo en que el Sistema Operativo le preste los recursos al programa, es decir puede que al hilo 1 le de 2milisegundos de vida, para luego darle 4milisegundos al hilo 2.

Para suspender un Hilo en particular, se realiza mediante la instrucción:

- `thread1.suspend();`
- `thread1.stop();`

Según sea necesario por el programador, otros temas de hilos no serán analizados en este curso como por ejemplo administrar la prioridad de cada hilo, mecanismos para sincronizar hilos y grupos de hilos. El lector puede consultar la bibliografía y profundizar en este tema mediante el link:

http://zarza.usal.es/~fgarcia/docencia/poo/01-02/trabajos/S3T3.pdf

Manejo de Sockets

Muchas de las aplicaciones corporativas no trabajan de forma aislada, sino que en cambio tienen que comunicarse con otras aplicaciones. Los sockets representan un mecanismo para comunicar dos aplicaciones Java por medio de una red Internet.

Clases para comunicación: java.net

En las aplicaciones en red es muy común el paradigma cliente-servidor. El servidor es el que espera las conexiones del cliente (en un lugar claramente definido) y el cliente es el que lanza las peticiones a la maquina donde se está ejecutando el servidor, y al lugar donde está esperando el servidor (el puerto específico que atiende). Una vez establecida la conexión, ésta es tratada como un *stream* (flujo) típico de entrada/salida.

Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación. Típicamente, no se necesita trabajar con las capas TCP y UDP, en su lugar se puede utilizar las clases del paquete `java.net`. Estas clases proporcionan comunicación de red independiente del sistema.

A través de las clases del paquete `java.net`, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases `URL`, `URLConnection`, `Socket`, y `SocketServer` utilizan TCP para comunicarse a través de la Red.

TCP proporciona un canal de comunicación fiable punto a punto, lo que utilizan para comunicarse las aplicaciones cliente-servidor en Internet. Las clases `Socket` y `ServerSocket` del paquete `java.net` proporcionan un canal de comunicación independiente del sistema utilizando TCP, cada una de las cuales implementa el lado del cliente y el servidor respectivamente.

Así el paquete java.net proporciona, entre otras, las siguientes clases:

- **`Socket`**: Implementa un extremo de la conexión TCP.
- **`ServerSocket`**: Se encarga de implementar el extremo Servidor de la conexión en la que se esperarán las conexiones de los clientes.
- **`InetAddress`**: Se encarga de implementar la dirección IP.

La clase `Socket` del paquete `java.net` es una implementación independiente de la plataforma de un cliente para un enlace de comunicación de dos vías entre un cliente y un servidor. Utilizando la clase `java.net.Socket` en lugar de tratar con código nativo, los programas Java pueden comunicarse a través de la red de una forma independiente de la plataforma.

El entorno de desarrollo de Java incluye un paquete, `java.io,` que contiene un juego de canales de entrada y salida que los programas pueden utilizar para leer y escribir datos. Las clases `InputStream` y `OutputStream` del paquete `java.io` son superclases abstractas que definen el comportamiento de los canales de I/O de tipo stream de Java. `java.io` también incluye muchas subclases de `InputStream` y `OtputStream` que implementan tipos especificos de canales de I/O.

Creación de un Socket

Sea el programa cliente, **PruebaEco**, conecta con el Echo del servidor (en el puerto 7) mediante un *socket*. El cliente lee y escribe a través del *socket*. **PruebaEco** envía todo el texto tecleado en su entrada estándar al Echo del servidor, escribiéndole el texto al *socket*. El servidor repite todos los caracteres recibidos en su entrada desde el cliente de vuelta a través del *socket* al cliente. El programa cliente lee y muestra los datos pasados de vuelta desde el servidor.

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

public class PruebaEco {

	public static void main(String[] args) {
		Socket ecoSocket = null;
		DataOutputStream salida = null;
		DataInputStream entrada = null;
		DataInputStream stdIn = new DataInputStream(System.in);

/* Establecimiento de la conexión del Socket entre el cliente y el servidor y
apertura del canal E/S sobre el socket : */

		try {
			ecoSocket = new Socket(args[0], 7);			salida =
new DataOutputStream(ecoSocket.getOutputStream());		entrada = new
DataInputStream(ecoSocket.getInputStream());
		} catch (UnknownHostException e) {
			System.err.println("No conozco al host: " + args[0]);
		} catch (IOException e) {
			System.err.println("Error de E/S para la conexión con: " +
args[0]);
		}

/* Ahora lee desde el stream de entrada estándar una línea cada vez. El programa
escribe inmediatamente la entrada seguida por un carácter de nueva línea en el
stream de salida conectado al socket. */

		if (ecoSocket != null && salida != null && entrada != null) {
			try {
				String userInput;
				while ((userInput = stdIn.readLine()) != null) {
				salida.writeBytes(userInput);
	salida.writeByte('\n');
					System.out.println("eco: " + entrada.readLine());
				}

/* Cuando el usuario teclea un carácter de fin de entrada, el bucle while
termina.
Cierre de los streams de entrada y salida conectados al socket, y cierre de la
conexión del socket con el servidor. */

				salida.close();				entrada.close();
				ecoSocket.close();
			} catch (IOException e) {
    System.err.println("E/S fallo en la conexión a: " + args[0]);
			}
		}
	}
}
```

Capítulo 2
Uso del JDBC para acceso a BD

Los grandes sistemas de cómputo, tiene algo en común. Todos y cada uno de ellos por detrás son soportados por una base de datos que les permite almacenar de forma persistente la información del sistema. Imaginemos un sistema perfectamente bien diseñado, pero que fuera incapaz de recuperar la información cuando la computadora se reinicie, serie en un buen sentido una gran obra ingenieril sin funcionalidad.

En el presente capitulo, se van a explorar los conceptos básicos de bases de datos. Así como los códigos fuentes de Java para conectarse a una base de datos. Para efectos de esta guía se va utilizar MySQL como herramienta de bases de datos.

Principios de bases de datos

Sobre el concepto de los datos

Las personas utilizan la palabra "datos" diariamente, para expresar una unidad atómica de información, por ejemplo el número 4 representa un dato, el cual por sí mismo no significa absolutamente nada. Cuando al número 4 se le endosa una unidad entonces decimos que es información útil. Por ejemplo, para el humano decir 4 años tiene más sentido que decir tiene 4… ¿Qué? Las bases de datos son excelentes formas de almacenar datos (no información, pues la información supone conocimiento del tema).

Sobre las bases de datos

Una base de datos, es en un sentido purista una aplicación que administra un conjunto de datos; sin embargo esa definición no es suficiente para explicar que es una base de datos. Java al ser un lenguaje de programación multipropósito, sería capaz de crear una base de datos desde 0, sin embargo es impráctico para los profesionales invertir grandes cantidades de recursos creando su propia base de datos, en vez de utilizar alguna opción libre. A estos programas que administran las bases de datos se les llaman Sistemas Gestores de Bases de Datos (SGBD). Para considerar que un SGBD es un buen gestor debe de tener las siguientes características:

Atomicidad es la característica que describe las transacciones, en donde se da "todo o nada". Supongamos un programa para transferencias monetarias, ahora supongamos que dicho programa usa una base de datos para quitar dinero de una cuenta y ponerle dinero a otra cuenta. Si durante el proceso de quitarle dinero a la primera cuenta, la base de datos cae por determinada falla, es tarea del SGBD abortar todo proceso

involucrado y deshacer los cambios. De esta forma se mantiene el todo y la parte. Si la transacción fuera exitosa entonces se dice que la base de datos esta ***consistente***[1].

Consistencia como producto de transacciones atómica, una base de datos deber estar siempre (o en algún momento en el tiempo, en el caso de las replicaciones) en un estado consistente. Es decir, todos los datos de la base tienen un estado apropiado. Además de las transacciones atómicas es posible llegar a la consistencia mediante reglas, constraints, cascades y triggers.

Isolation (aislamiento) tiene relaciones a la teoría de bloqueos, una transacciones debe ser aislada en el sentido que su ejecución no afecte a otra transacción. Supongamos que existen la transacción T1 y T2. Ahora T1 afecta los datos de salarios para leerlos, mientras la transacción T2 realiza una actualización aumentándole 5.000 colones a todos los empleados. Para lograr que ambas transacciones se ejecuten de forma adecuada, es necesario aplicar un *bloqueo no exclusivo* a la transacción T1 mientras que la transacción T2 deberá tener un *bloqueo exclusivo*, para que la información de lectura sea la correcta; mientras se **aísla** la transacción T2.

Durabilidad una vez que una transacción se ejecuta, la misma deberá ser permanente. Es decir, no podría suceder que una transacción se deshaga sin motivo. Para ello los SGBD implementan logs de transacciones y escritura a los archivos físicos.

El conjunto de estas cuatro características, típicamente se denotan por ACID.

Sobre las tablas

Las tablas son los elementos básicos de las bases de datos. La mejor forma de representarlas es como si fueran hojas electrónicas, donde existen datos de la misma naturaleza (filas) a las que se les llaman tuplas. Cada tupla está formada por múltiples dimensiones a las que se les conoce como columnas. Por ejemplo la siguiente tabla representa información relacionada con los recursos humanos de una empresa.

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela María	Avalos Chavarría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

Como se puede observar, la tupla número 5 tiene información horizontal (llamadas columnas), la cual se puede representar de la siguiente forma: `(05, “Gabriela María”, “Avalos Chavarría” 02, 500000).`

[1] Consistencia se refiere a que todas las transacciones de una base de datos, son transacciones completas y por lo tanto no hay información incorrecta entre los datos.

Tipos de datos

Como lo ha notado previamente, la tupla anterior está construida por números enteros (05, 02), literales (“Gabriela”, “Avalos Chavarría”) y decimales (`500000.0`). Los SGBD tienen una gran cantidad de tipos de datos. En la siguiente tabla se expresan los tipos de datos de MySQL.

Nombre	Tipo	Tamaño
TINYINT	Numérico entero	1 BYTE
SMALLINT	Numérico entero	2 BYTE
MEDIUMINT	Numérico entero	3 BYTE
INT	Numérico entero	4 BYTE
BIGINT	Numérico entero	8 BYTE
FLOAT	Numérico flotante	8 BYTE
DOUBLE	Numérico flotante	8 BYTE

Tipos de datos numéricos

También existen datos de tipo **fechas,** estos permiten almacenar una fecha específica. Opcionalmente se puede almacenar una hora.

Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-día.

DateTime: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

TimeStamp: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037.

Por otro lado, también existen datos de tipo literal o cadena de texto.

Char(n): almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n): almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Consultas DML

Existen lenguajes especiales para comunicarse con las bases de datos, estos lenguajes reciben el nombre de SQL. Estos son lenguajes que son invocados a través de un conector, permitiendo modificar el estado de una base de datos.

Podemos ver el paso de consultas SQL como una acción hacia una caja negra, la cual nos garantiza el cumplimiento del ACID, por lo que la labor del programador de aplicaciones consiste en la manipulación de la parte visual (vista), el desarrollo del

controlador y parte de la lógica del modelo (solo la parte de la conexión a la base de datos).

Existen muchas formas de mover toda la lógica de la aplicación, a las bases de datos. Esto se logra con los procedimientos almacenados, sin embargo este curso nos centraremos en colocar la lógica del lado de Java.

Se puede clasificar el SQL en dos grandes conjuntos, las instrucciones DML que permite manipular datos, como por ejemplo: agregar una nueva tupla, modificar una tupla, eliminar una tupla o consultar la tupla.

Insertar datos a una tabla

Supongamos que tenemos la siguiente tabla:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

Pero queremos agregarle un nuevo registro, puesto que la aplicación necesita registrar un nuevo empleado en el sistema. El nuevo empleado tiene la siguiente estructura: (07, "Rafa", "Mata Calderón", 01, 46664.6).

En términos generales la sintaxis para agregar un nuevo registro es:

```
INSERT INTO <nombre_tabla> VALUES (<v1>, <v2>,…);
```

Por ejemplo, la expresión quedaría:

```
INSERT INTO personas VALUES (07, 'Rafa', 'Mata Calderon', 01, 46664.6);
```

Como resultado de la ejecución, la tabla quedaría de la siguiente forma:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarrría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0
07	Rafa	Mata Calderon	01	46664.6

Eliminar datos a una tabla

Supongamos que tenemos la siguiente tabla:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarrría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

Ahora se nos ha solicitado eliminar a todos los empleados que tengan el campo ID menor o igual que 2 o que tengan el campo nombre igual a 'Xinia'. Y que el salario **NO** sea mayor a `550000`.

En términos generales la sintaxis para eliminar un registro es:

```
DELETE <nombre_tabla> WHERE <condición1> <expresionLogica> …
```

Donde la condición, debe ser una expresión booleana valida. Las siguientes son expresiones validas:

- `A =` 4, A igual a 4
- `A <` 4, A menor que 4
- `A <=` 4, A menor o igual que 4
- `A <>` 4, A diferente que 4
- `A >` 4, A mayor que 4
- `A >=` 4, A mayor o igual que 4

Mientras la expresión lógica puede ser cualquiera de las siguientes:

- `<…a> AND <…b>`, Se cumple tanto a como b al mismo tiempo
- `<…a> OR <…b>`, Se cumple a o b (pudiera ser ambas al mismo tiempo)

Así la expresión quedaría:

```
DELETE personas WHERE (ID <= 2 OR nombre = 'Xinia') AND salario <= 550000
```

Como resultado de la ejecución, la tabla quedaría de la siguiente forma:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0

Modificar datos a una tabla

Supongamos que tenemos la siguiente tabla:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarrría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

Ahora se nos ha solicitado modificare el salario a todos los empleados que tangan un salario menor que 55000000, para ello súmele 10.000 colones a todos los empleados.

En términos generales la sintaxis para eliminar un registro es:

```
UPDATE FROM <nombre_tabla> SET <columna> = <valor> WHERE [<condición1><expresionLogica> …]
```

La expresión quedaría:

```
UPDATE FROM personas
SET salario = salario + 10000
WHERE salario <= 550000
```

Si queremos aplicarme el cambio a todos los datos de la tabla, es necesario ejecutar la siguiente expresión:

```
UPDATE FROM personas
SET salario = salario + 10000
~~WHERE salario <= 550000~~
```

Como resultado de la primera ejecución, la tabla quedaría de la siguiente forma:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	**501000.0**
05	Gabriela	Avalos Chavarrría	02	**501000.0**
06	Xinia	Chavarría Oviedo	05	**201000.0**

Consultar datos a una tabla

Supongamos que tenemos la siguiente tabla:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarrría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

Ahora se nos ha solicitado crear un informe, el cual debe mostrar todos los subalternos del jefe 01. Es decir, los que tengan como campo jefe = 01. Además, lo que interesa en el informe es el ID, nombre y apellido.

En términos generales la sintaxis para obtener un(os) registro es:

```
SELECT {*|<columnas>} FROM <tabla> WHERE <condicion1>…
```

Así la expresión quedaría:

```
SELECT id, nombre, apellidos FROM personas WHERE jefe = 01
```

Como resultado se crea una tabla en memoria que es una sub-relación (podría ser el mismo conjunto original) a la tabla en la que se le invoca la consulta.

PERSONAS		
ID	**Nombre**	**Apellidos**
02	Ezzio	Auditore
03	Nicolas	Brenes Alvares

De haberse usado el * en la consulta quedaría:

```
SELECT * FROM personas WHERE jefe = 01
```

El resultado debería ser:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0

Consultas DDL

Las consultas DDL permiten modificar la base de datos a nivel de estructura “lógica”, esto es agregar una tabla, eliminar la tabla, agregarle columnas, removerle columnas, crear procedimientos, cambiar dominios (o tipos de datos) a las columnas entre muchas otras funcionalidades. Para efectos de este curso, los DDL no serán estudiados.

Objetos de JDBC

Prácticamente todas las aplicaciones J2EE almacenan, recuperan y manipulan información que reside en las bases de datos. Existe para ello una facilidad en Java para establecer conexión con las bases de datos, y ejecutar cualquier DDL y DML que soporte la base de dato en cuestión.

Existe muchos SGBD comerciales, que van desde Oracle, DB2, Sysbase y SQL Server. Sin mencionar las alternativas libres como MySQL, Postgress. La idea que nació en los años noventa fue desarrollar algún mecanismo para conectarse a las bases de datos más populares; la primera barrera que debieron romper los investigadores de Sun Microsystems fue la enorme cantidad de lenguajes de programación, pues cada SGBD tiene su propio lenguaje y sus propios estándares de conexión; esto significa que se debía reescribir todo el código de bajo nivel que se tenía para conectarse a una base de datos Oracle, si queríamos cambiar de bases por Sysbase.

La solución de Sus Microsystem fue el desarrollo en 1996, del paquete para desarrolladores Java Developer Kit (JDK) y de la API Java Data Base Conector (JDBC); estos controladores fueron incorporados por necesidad, puesto que antes de la introducción de estos controladores a Java no se le consideraba un lenguaje empresarial (puesto que no existían forma de conectarse a la base de datos).

El JBDC de hecho no era un controlador, era una serie de normas y estándares de como deberá funcionar el conector en Java, la idea fue incentivar a desarrolladores de bases de dato y a terceras personas a construir su conector propio; con el fin claramente de explotar todo el surgimiento de Java.

Como resultado del desarrollo de muchos conectores JDBC, se podría utilizar objetos de alto nivel de Java en consultas de bajo nivel en las bases de datos. Esto permitía a los desarrollares simplificar el desarrollo de aplicaciones.

El controlador JDBC hace que los componentes J2EE sean independientes de la base de datos, lo cual se ajusta a la filosofía Java de independencia entre plataformas. Las consultas se envían sin ninguna validación, esto significa que es labor de la base de datos validar y comprobar la correcta ejecución de las consultas y procedimientos.

Proceso para establecer conexión

En términos generales cada SGBD desarrolla su propio conector, y por lo tanto el protocolo de conexión. Sin embargo la gran mayoría sigue los siguientes pasos básicos:

1. Cargar el controlador JDBC
2. Conectar con el SGBD
3. Crear y ejecutar una instrucción SQL
4. Procesar los datos que devuelven el SGBD
5. Terminar la conexión

Cargar el controlador JDBC

Es necesario cargar el controlador JDBC para poder interactuar con la base de datos. Java particularmente puede crear instancias de objetos mediante una URL; es justamente esta URL la que debe ser cambiada para poder crear la instancia del objeto JDBC.

Es de esta forma, que si un programador quiere realizar una conexión a una base de datos Microsoft Access, el programador debe escribir la rutina que cargue el controlador Puente JDBC/ODBC, el cual se llama sun.jdbc.JdbcOdbcDriver.

Para crear el controlador se invoca el método Class.forName( ) y se le pasa el URL de la conexión que se necesita. Por ejemplo el siguiente código crea una conexión para Microsoft Access:

```
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
```

Conectar con el SGBD

Una vez que se ha escogido la cadena de conexión, es necesario conectarse mediante el método DriverManager.getConnection( ).Esta clase es la más alta en la jerarquía de JDBC y es la responsable de controlar la información de los controladores.

Una vez que se invoca el método getConnection( ), este va ser el encargado de implementar la interface Connection, la cual vamos a utilizar para realizar conexiones con el SGBD.

En el siguiente ejemplo, el URL “jdbc:odbc:InformacionEmpleados” nos permite acceder al esquema InformacionEmpleados de la base de datos de Microsoft Access.

```
package ejemplo.pkg1;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexionBD {

    private String url = "jdbc:odbc:InformacionEmpleados";
    private String usuario = "sbrenesms";
    private String clave = "12345";
    private Connection bd = null;

    public ConexionBD(){
        try {
            Class.forName("jdbc.odbc.JdbcOdbcDriver");
            bd = DriverManager.getConnection(url, usuario, clave);
        } catch (ClassNotFoundException | SQLException ex) {
        }
    }
}
```

Crear y ejecutar instrucciones SQL

El siguiente paso después de crear la conexión, mediante el administrador de conexiones, consiste en invocar la sentencia SQL (véase inicio del capítulo) que le indican al SGBD que hacer ya, sea insertar, eliminar, actualizar o consultar tablas.

El método Connection.createStatement( ) se utiliza para crear un objeto de tipo Statement (instrucción). El objeto Statement se utiliza para ejecutar una consulta y devolver un objeto ResultSet (conjunto de resultados) la cual contiene la información que el SGBD regresa; típicamente una o más tuplas.

```
public ResultSet recuperarEmpleados() throws SQLException{
     String query = "SELECT * FROM empleados";
     Statement consulta  = bd.createStatement();
     ResultSet resultados = consulta.executeQuery(url);
     consulta.close();
     return resultados;
}
```

El código anterior es capaz de recuperar la consulta de todos los empleados, de una base de datos previamente conectada con el objeto BD. Es realmente importante cerrar la conexión una vez que se ha terminado de ejecutar, de esta forma evitamos consumir memoria del *pool de conexiones* del SGBD.

Recuperar conjuntos a alto nivel

Supongamos que necesitamos recuperar la siguiente tabla, de forma que un programador sea capaz de "mapear" la tabla a una clase Persona. Para nuestro efecto vamos a recuperar la siguiente tabla:

PERSONAS				
ID	**Nombre**	**Apellidos**	**Jefe**	**Salario**
01	Steven	Brenes Chavarría	null	900000.0
02	Ezzio	Auditore	01	600000.0
03	Nicolas	Brenes Alvares	01	600000.0
04	Sofia	Peralta García	02	500000.0
05	Gabriela	Avalos Chavarrría	02	500000.0
06	Xinia	Chavarría Oviedo	05	200000.0

El siguiente paso es crear la clase Persona en Java, el código resultante sería:

```
public class Persona {
    private int id;
    private String nombre;
    private String apellidos;
    private int jefe;
    private double salario;

    public Persona(int id, String nombre, String apellidos, int
jefe, double salario) {
        this.id = id;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.jefe = jefe;
        this.salario = salario;
    }

    public int getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public String getApellidos() {
        return apellidos;
    }

    public int getJefe() {
        return jefe;
    }

    public double getSalario() {
        return salario;
    }
}
```

El siguiente paso es crear un método, que pueda recuperar la tablas personas y convertirla en una lista de personas, para mapearlas directamente a Java. Para ello es necesario ir leyendo tupla por tupla y cargando los datos en variables locales, para posteriormente crear con estas variables locales, y con ello crear la clase completa. El siguiente código ejemplifica el procedimiento:

```
public List<Persona> recuperarEmpleados() throws Exception {
      List<Persona> salida = new ArrayList<Persona>();
      String query = "SELECT * FROM personas";
      try (Statement consulta = bd.createStatement()) {
         ResultSet resultados = consulta.executeQuery(url);
         while(resultados.next()){
           String nombre = resultados.getString("nombre");
           String apellidos = resultados.getString("apellidos");
           int id = resultados.getInt("id");
           int jefe = resultados.getInt("jefe");
           double salario = resultados.getDouble("salario");
           Persona nuevaPersona = new Persona(id,
                    nombre,
                    apellidos,
                    jefe,
                    salario);
           salida.add(nuevaPersona);
        }
     }
     return salida;
}
```

Una de las formas que se pueden cargar los datos del DataSet es mediante un número entero que va desde 0…n, así por ejemplo la consulta:

```
SELECT nombre, apellidos, salario FROM personas
```

Podrían ser cargada con la siguiente igualdades: nombre (0), apellidos (1), salario (2). Sin embargo el problema de este modelo, se da cuando se cambia el esquema de la consulta o se cambia la estructura de la tabla; ambas situaciones podrían darse sin que el desarrollador se dé cuenta del cambio. Es por ello que la recomendación oficial consiste en cargar los elementos indicándoles explícitamente el nombre del campo, como se detalla mediante las siguientes líneas de programación:

```
String nombre = resultados.getString("nombre");
String apellidos = resultados.getString("apellidos");
```

Mapeo de datos Java contra SQL

En el código anterior se ha utilizado la invocación del método getString, getInt, getDouble. Estos métodos son utilizados para mapear los datos del SGBD con datos de Java. Además de los métodos de getYYY(), existen los métodos de setYYY() y updateYYY().

En la siguiente tabla se estarán mapeando los datos desde el SGBD contra los datos de Java.

TIPO SQL	**TIPO JAVA**
CHAR	String
VARCHAR	String
LONGCHAR	String
Numeric	Java.math.BigDecimal
Decimal	Java.math.BigDecimal
BIT	Boolean
TINYINT	Byte
Smallint	Short
Integer	Integer
Bigint	Long
Real	Float
Float	Float
Double	Double
Binary	Byte[]
Varbinary	Byte[]
longvarbinary	Byte[]
BLOB	Java.sql.Blob
CLOB	Java.sql.Clob

Objetos de tipo Statement

En los ejemplos de códigos anteriores, se han utilizado para invocar a la base de datos el objeto Statement, a este objeto se le introduce una sentencia SQL. Sin embargo Java provee otras versiones de Statement que se adaptan a las necesidades especiales de los programadores. Existen tres versiones diferentes de objetos Statement.

El objeto **Statement** ejecuta la consulta SQL de forma directa, sin esperar nada. El objeto **PreparedStatement** ejecuta una consulta SQL ejecutada previamente, esto mejora enormemente el rendimiento en consultas que son frecuentemente utilizadas. Finalmente el objeto **CallableStatement** es utilizado para invocar procedimientos almacenados[2].

El objeto Statement

El objeto Statement se utiliza cada vez que un componente J2EE necesita ejecutar una consulta de forma inmediata sin tener que compilarla primero. El objeto Statement tiene un método executeQuery("consulta"); que recibe una consulta la cual es

[2] Un procedimiento almacenados en los SGBD, son equivalentes a las funciones en JAVA. Son bloques de código escrito en una extensión de SQL.

trasmitida directamente al SGBD para su ejecución. El método executeQuery() retorna un objeto de tipo ResultSet que representa los datos que se han invocado en la consulta. El objeto ResultSet tiene métodos para ir iterando los resultados obtenidos de la base de datos mediante el patrón iterador[3].

El objeto Statement también contiene el método executeUpdate("instrucción SQL"), este método es utilizado para invocar consultas INSERT, UPDATE y DELETE. Esto debido a que las consultas anteriores no retornan datos de tipo tabla, por lo que no existe necesidad de regresar el objeto ResultSet; a diferencia del executeQuery(…) que retorna un objeto de tipo ResultSet, el método executeUpdate(…) retorna un entero, que representa la cantidad de columnas modificadas, de forma tal que si nuestro update afecta a solamente 3 tuplas, el resultado de este método va ser claramente 3.

Ejemplo del uso del Statement

El siguiente código permite ingresar mediante el objeto Statement, una persona a la tabla Personas. Observe como se construye el SQL mediante concatenación, en otros objetos Statement es posible sustituir dicho SQL por mecanismos mucho más eficientes.

El procedimiento consisten en abrir la conexión, crear un objeto Statement, a este objeto se le invoca el método executeUpdate con el SQL que se ha creado manualmente, al final se debe cerrar todas las conexionas abiertas.

```
public boolean ingresarPersona(int id, String nombre){
  int modificado = 0;
  try {
    Class.forName("jdbc.odbc.JdbcOdbcDriver");
    bd = DriverManager.getConnection(url, usuario, clave);
     String sql = "INSERT INTO personas VALUES(" + id + "'" + nombre + "')";
    Statement comando = bd.createStatement();
    modificado = comando.executeUpdate(sql);
    comando.close();
    bd.close();
  } catch (Exception ex) {
  }
  finally {
    return modificado == 1;
  }
}
```

El objeto PreparedStatement

Antes de ejecutar una consulta el SGDB tiene que compilar la consulta, con el objetivo de obtener la mejor ruta para acceder a los datos. A este proceso se le llama simplemente *path* y tiene por objeto identificar la forma más eficiente de responder a la consulta.

[3] El patrón iterador permite manipular colecciones de datos de forma más simple para el programador, existe típicamente el método next() el cual retorna el dato actual.

El uso del objeto PreparedStament, se justica cuando la consulta que se invoca es reutilizable en varias partes de la aplicación, por ejemplo si tenemos una consulta donde se cargan los primeros 10 clientes. Podría justificarse el uso de este objeto para mejorar el rendimiento del sistema al evitar recompilar una consulta varias veces.

La diferencia a nivel de programación es el uso del operador de sustitución, en el caso de Java la sustitución se hace mediante el carácter signo de pregunta "?". Este carácter es sustituido por la objeto con los valores que necesitosl; en otras palabras ya no tenemos que crear el SQL manualmente, solo es necesario especificar el valor de Java que va ser utilizado.

Por ejemplo, supongamos la siguiente consulta:

```
SELECT * FROM personas WHERE id = ?
```

Mediante el objeto PreparedStatement es posible cambiar el signo de pregunta por el ID del empleado que queremos recuperar. El siguiente código ejemplifica el uso de este objeto al ingresar a la base de datos una persona nueva.

```
public boolean ingresarPersona(int id, String nombre, String apellido) {
        int modificado = 0;
        try {
            Class.forName("jdbc.odbc.JdbcOdbcDriver");
            bd = DriverManager.getConnection(url, usuario, clave);
            String sql = "INSERT INTO personas VALUES(?, ?, ?)";
            PreparedStatement comando = bd.prepareStatement(sql);
            comando.setInt(1, id);
            comando.setString(2, nombre);
            comando.setString(3, apellido);
            modificado = comando.executeUpdate();
            comando.close();
            bd.close();
        } catch (Exception ex) {
        } finally {
            return modificado == 1;
        }
    }
```

Observe el código anterior, existen algunas definiciones de métodos del objeto PreparedStatement como setString el cual sustituye el ? por el valor adecuado. Luego se ha invocado el método executeUpdate sin parámetros simplemente porque el SQL ya fue procesado cuando se crea el objeto.

Este objeto es útil cuando se necesita invocar consultas SQL que nunca cambiar, es decir que no tenga parámetros, es por ello que para el código anterior no se recomienda usar el objeto Statement.

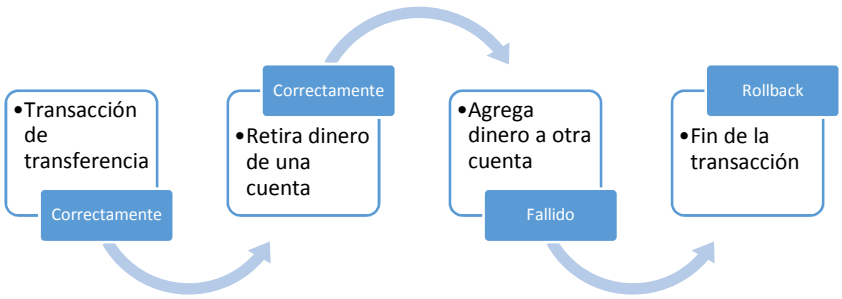
Observe que modificar el método para eliminar o actualizar una persona de una tabla, consiste simplemente en cambiar la consulta SQL.

Manejo de transacciones

Una de las características del ACID, es la consistencia. En torno a la consistencia, las bases de datos deben estar en un estado donde todos los datos son correctos. Para asegurar el proceso de transacción en Java es necesario explicar nuevos métodos que nos permiten definir las reglas del todo o del nada.

Cuando se plantean transacciones en Java, es importante tener claro la idea central. Se define una transacción atómica como aquella transacción en donde al fallar al menos uno de los procesos, toda la transacción se debe deshacer. Al proceso de deshacer en bases de datos, se le denota como "*rollback*" mientras que al proceso de hacer permanente los cambios se le llama "commit".

Observe el siguiente diagrama de procesos, el cual está modelando la transacción de hacer una transferencia de dinero entre dos cuentas del mismo banco. Para el caso del ejemplo, se suponen que una transacción correcta es aquella transacción en la que al ser ejecutada no produce ninguna excepción.



Particularmente la transacción anterior falló en la etapa de agregar dinero a la otra cuenta, esto podría ser cauda por ejemplo de que no exista el número de cuenta por lo que al sumarle el dinero lanza una excepción. Como la base no puede quedar en un estado inconsistente (cuentas debitadas pero nunca acreditadas) es necesario invocar el Rollback; esto hace que el retiro de dinero de la primer etapa se deshaga. Las causas que lanzan un Rollback podrían ser:

1. La cuenta origen no exista
2. La cuenta origen este congelada
3. La cuenta origen no tengan fondos suficientes
4. La cuenta origen tenga bloqueado todos los débitos
5. La cuenta origen tenga bloqueado los débitos mayores a X monto
6. La cuenta destino no exista
7. La cuenta destino está congelada
8. La cuenta destino no acepte créditos
9. La cuenta destino no acepte créditos mayores a X monto
10. Otros fallos en el SGBD, como tablespace llenos o HDD corruptos

El siguiente código escrito en Java tiene la intensión de modelar el proceso de realizar la transferencia de dinero entre dos cuentas.

```
public boolean transferirDinero(String orig, String dest, double
monto){
        int modificado1 = 0;
        int modificado2 = 0;
        boolean terminoBien = true;
        try {
            Class.forName("jdbc.odbc.JdbcOdbcDriver");
            bd = DriverManager.getConnection(url, usuario, clave);
            bd.setAutoCommit(false);
            String sql1 = "UPDATE cuentas SET dinero = dinero - ?
WHERE dueno = ?";
            String sql2 = "UPDATE cuentas SET dinero = dinero + ?
WHERE dueno = ?";
            PreparedStatement              comando1             =
bd.prepareStatement(sql1);
            PreparedStatement              comando2             =
bd.prepareStatement(sql2);
            //
            comando1.setDouble(1, monto);
            comando1.setString(2, orig);
            //
            comando2.setDouble(1, monto);
            comando2.setString(2, dest);

            modificado1 = comando1.executeUpdate();
            modificado2 = comando2.executeUpdate();
            if(modificado1 == 0 || modificado2 == 0){
                // rollback pues alguna de las 2 cuentas no existe
                bd.rollback();
                terminoBien = false;
            }
            else{
                bd.commit();
            }
            comando1.close();
            comando2.close();
            bd.close();
        } catch (Exception ex) {
            bd.rollback();
            bd.close();
            terminoBien = false;
        } finally {
            return terminoBien;
        }
    }
```

Ponga mucha atención al código anterior, esta con negrita resaltado lo nuevos aprendizajes en torno al manejo de transacciones. Observe que hay que definirle al

conector que el parámetro autoCommit va ser false, de otra forma cada operación execute ejecutaría un Commit implícito. También es importante que observe cuando se debe invocar el commit (cuando las 2 operaciones fueran completadas correctamente). Desde luego, observe también el momento que se ejecuta el Rollback, el mismo es ejecutado en dos situaciones: la primera cuando alguna (o ambas) de las cuentas no existan, o cuando la ejecución falle (por ejemplo una violación a una regla de integridad).

De esta forma, se ejecutan todas y cada una de las operaciones creando una transacción atómica, que garantiza que no existan débitos sin créditos a las diversas cuentas.

Ejemplos de URL para la conexión

En las páginas anteriores se han presentado ejemplos para conectarse, a bases de datos mediante ODBC, específicamente al SGBD Microsoft Access. Ahora es momento de presentar otras alternativas para bases de datos propietarias y libres.

Las siguientes líneas, ejemplifican los posibles URLs para abrir la conexión, de las principales bases de datos. Cada URL tiene una serie de elementos importantes que se deben de tomar en cuenta.

Toda conexión necesita de un "**nombre de host**", este valor puede ser una dirección IP (por ejemplo: 192.46.43.67), un sinónimo (como localhost) o una dirección URL como por ejemplo: http://servidorbasesdatos.com

Además de la dirección del servidor, muchas veces es necesario especificarle un **puerto** de entrada; este puedo no necesariamente es el mismo siempre, va cambiar según varios criterios, como por ejemplo las políticas de seguridad, o el direccionamiento de puerto del servidor.

Supongamos que no sabemos el puerto, entonces se podría usar los siguientes valores por "default". El valor predeterminado para bases de datos DB2 es 50000; para bases de datos Oracle es 1521; para bases de datos Informix es 9088; y para MySQL es 3306.

Un mismo SGBD tiene diferentes "instancias" que son en sí mismas bases de datos, es normal tener por ejemplo la base de datos de contabilidad separada lógicamente de la base de datos de ventas, por motivos de seguridad; en el caso de Oracle se llaman instancias, mientras que en otros SGBD se les llaman esquemas. Para conectar el JDBC es necesario especificar el **nombre de la base de datos**.

Esquema de las URLs

Informix

```
jdbc:informix-
sqli://nombre_host:puerto/nombre_base_datos:INFORMIXSERVER=servidor;
DELIMIDENT=YIFX_LOCK_MODE_WAIT=-1
```

Oracle

```
jdbc:oracle:thin:@nombre_host:puerto:nombre_base_datos
```

MySQL

`jdbc:mysql://nombre_host:puerto/nombre_base_datos`

DB2

`"jdbc:db2://nombre_host:puerto/nombre_base_datos:currentSchema=NCIM;"`

SQL Server

`jdbc:sqlserver://nombre_host\\nombre_base_datos;puerto`

PostgreSQL

`jdbc:postgresql://nombre_host:puerto/nombre_base_datos`

Ejemplos de URLs conformadas

Informix

```
jdbc:informix-sqli://192.168.1.2:9088/itnm:INFORMIXSERVER=demo_on;
DELIMIDENT=Y; IFX_LOCK_MODE_WAIT=-1
```

Esta URL de ejemplo conecta a una base de datos de Informix con las siguientes propiedades:

1. La dirección IP del host de servidor de base de datos es 192.168.1.2.
2. La base de datos se ejecuta en el puerto 9088. Este es el puerto predeterminado para Informix.
3. El nombre de la base de datos de Informix es itnm.
4. El nombre de la instancia del servidor de Informix es demo_on.

Oracle

```
jdbc:oracle:thin:192.168.1.2:1521:bdVentas
```

Esta URL de ejemplo conecta a una base de datos de Oracle con las siguientes propiedades:

1. La dirección IP del host de servidor de base de datos es 192.168.1.2.
2. La base de datos se ejecuta en el puerto 1521. Este es el puerto predeterminado para Oracle.
3. El nombre de la base de datos de Oracle es `bdVentas`

MySQL

```
jdbc:mysql://192.168.1.2:3306/bdVentas
```

Esta URL de ejemplo conecta a una base de datos de MySQL con las siguientes propiedades:

1. La dirección IP del host de servidor de base de datos es 192.168.1.2.
2. La base de datos se ejecuta en el puerto 3306. Este es el puerto predeterminado para MySQL.
3. El nombre del esquema de la base de datos de topología es `bdVentas`.

DB2

```
jdbc:db2://192.168.1.2:50000/itnm:NCIM
```

Esta URL de ejemplo conecta a una base de datos de DB2 con las propiedades siguientes:

1. La dirección IP del host de servidor de base de datos es 192.168.1.2.

2. La base de datos se ejecuta en el puerto 50000. Este es el puerto predeterminado para DB2.
3. El nombre de la base de datos de DB2 es itnm.
4. El nombre del esquema de la base de datos de la topología, en mayúsculas, es NCIM.

SQL Server

```
jdbc:sqlserver://192.168.1.2\\sqlexpress;50000;user=sa;password=se
cret
```

Esta URL de ejemplo conecta a una base de datos de SQL Server con las propiedades siguientes:

1. La dirección IP del host de servidor de base de datos es 192.168.1.2.
2. La base de datos se ejecuta en el puerto 50000.
3. El nombre de la base de datos de SQL Server es sqlexpress.

PostgreSQL

```
jdbc:postgresql://localhost:5432/testdb
```

Esta URL de ejemplo conecta a una base de datos de PostgreSQL con las propiedades siguientes:

1. La dirección IP del host de servidor de base de datos es `localhost`
2. La base de datos se ejecuta en el puerto 5432.
3. El nombre de la base de datos es testdb

Ejemplos de controladores JDBC

Los códigos de arriba son ejemplos de URLs para conectarse a diferentes bases de datos. Sin embargo falta especificar los códigos de los conectores JDBC. Estos códigos típicamente deben estar acompañados de un API el cual se debe descargar del sitio web del SGBD.

Los siguientes códigos se deben poner en la clase: Class.forName("nombreDelConector");

Informix: `com.informix.jdbc.IfxDriver`

Oracle: `oracle.jdbc.driver.OracleDriver`

MySQL: `com.mysql.jdbc.Driver`

DB2: `com.ibm.db2.jcc.DB2Driver`

SQL Server: `com.microsoft.sqlserver.jdbc.SQLServerDriver`

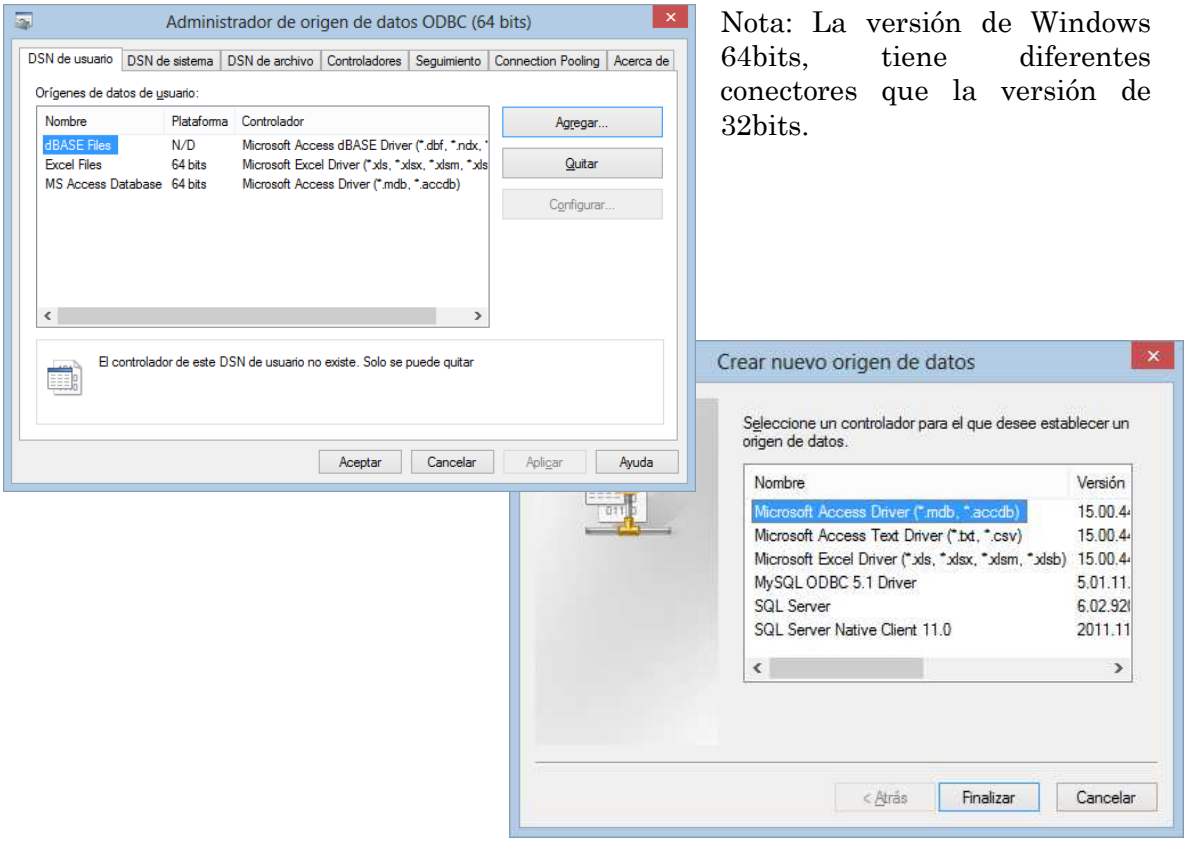
PostgreSQL: `org.postgresql.Driver`

Asociar el puente JDBC/ODBC con la base de datos

Algunas bases de datos no cuentan con su propio conector de base de datos, en estos casos lo normal es utilizar una conexión ODBC para establecer conexión entre el código Java y la base de datos.

Para establecer una conexión, siga los siguientes pasos en Windows:

1. Seleccionar Inicio | Configuración | Panel de Control.
2. Seleccionar Fuente de datos ODBC, con lo que se mostrará el Administrador de Orígenes de datos ODBC.
3. Incluir un nuevo usuario mediante el botón Agregar.
4. Se desplegaran todas aquellas bases de datos que estén registradas en la computadora.
5. Seleccionar el controlador y dar click a finalizar.
6. Según sea el controlador se desplegaran más datos que típicamente se refieren al nombre del esquema, nombre de usuario, contraseña para conectarse.
7. Dar click en finalizar.



Nota: La versión de Windows 64bits, tiene diferentes conectores que la versión de 32bits.

Capítulo 3
Hyper Text Markup Language

La construcción de sitios web ha traído a muchas empresas y sectores del gobierno poder crear portales para realizar trámites en línea. De esta posibilidad técnica es como surgen la idea en Costa Rica de crear un GobiernoDigital el cual tiene como metada poder realizar todos los tramites en línea; para ello los programadores han recurido a muchos lenguajes de programación como ASP.net, en el caso particular de Java; existe todo una sub-especificación del lenguaje llamado JSP que permite crear portales web con funcionalidades (sitios transaccionales). En este capítulo aprenderemos los fundamentos básicos del HTML, el cual se toma como base a la hora de crear paginas JSP.

Introducción a HTML

HTML es la abreviatura de HyperText Markup Language, y es el lenguaje que todos los navegadores usan para crear las páginas que visitamos.

Es un lenguaje de programación muy simple, que utiliza etiquetas anidadas para expresar la semántica de los objetos a mostrar, el texto que se ingresa entre los paréntesis angulares (<… >) tiene un descriptor que le permite al programador identificar el tipo de objeto que se está declarando. Por ejemplo el siguiente código representa un texto: "*hola* ***mundo***".

```
<i>hola <b>mundo</b></i>
```

La letra i del primer paréntesis significa italic, y la b de <b> significa bold. El resultado de esta página web es el texto "*hola* ***mundo***". Observe que estos elementos se pueden anidar tanto como el programador necesite.

Las etiquetas podrán incluir una serie de atributos o parámetros, en su mayoría opcionales, que nos permitirán definir diferentes posibilidades o características de la misma. Estos atributos quedarán definidos por su nombre (que será explicativo de su utilidad) y el valor que toman separados por un signo de igual. En el caso de que el valor que tome el atributo tenga más de una palabra deberá expresarse entre comillas, en caso contrario no será necesario. Así por ejemplo la etiqueta `<table border=2>` nos permitirá definir una tabla con borde de tamaño 2.

Reglas de formato

Todos los navegadores utilizan reglas para interpretar las páginas web, sin embargo no todos los navegadores utilizan las mismas reglas, a esto se le llama Estándar, y la meta de los desarrolladles de navegadores es ir cumpliendo cada vez más estándares. En general podemos citar las siguientes reglas genéricas.

- El espacio en blanco es ignorado. Ya que un documento HTML puede estar en cualquier tipo de fuente y además la ventana del navegador puede ser de cualquier tamaño. **El tamaño es relativo no absoluto.**

  - Existe normalmente una etiqueta de inicio y otra de fin. La etiqueta de fin contendrá el mismo texto que la de inicio añadiéndole al principio una barra inclinada /. La etiqueta afectará por tanto a todo lo que esté incluido entre las etiquetas de inicio y fin. **Existen etiquetas que no cumplen esta regla.**Estructura de un documento HTML

Todo documento HTML tiene como estructura básica las siguientes líneas, estas confirman un esqueleto básico que permite separar dos grande grupos. Los metadatos (head) y el cuerpo (body). Los metadatos ayudan a al navegador a interpretar correctamente el código que se encuentra en el body, mientras que el body se usa para definir propiamente la página.

```
<html>
      <head>
                  …
      </head>
      <body>
                  …
      </body>
 </html>
```

Ninguno de estos elementos es obligatorio, pudiendo crear documentos HTML sin incluir estas etiquetas de identificación. No obstante es altamente recomendable la construcción de páginas HTML siguiendo esta estructura.

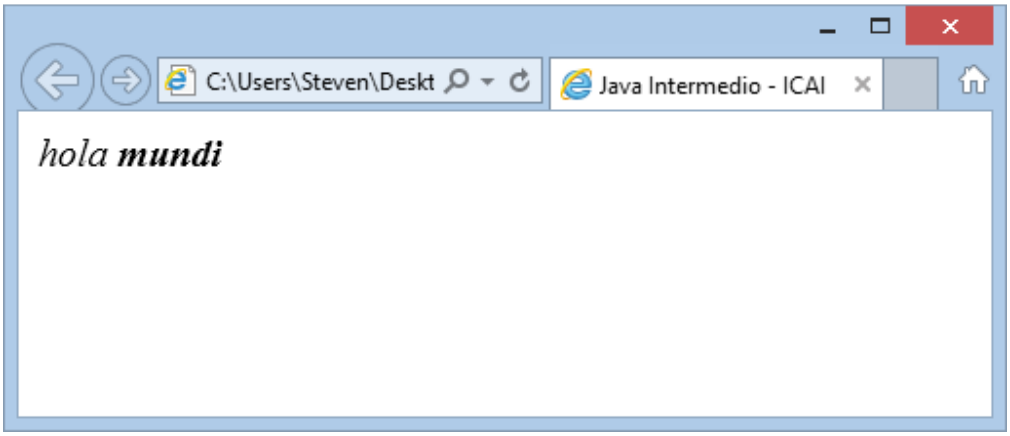
Cabecera del documento HTML

En las cabeceras de cualquier documento HTML incluiremos información general de la página, es decir, información que es global a la página.

Título del documento

```
<title>Mensaje del titulo</title>
```

El título de nuestra página web viene especificado por las etiquetas: <TITLE> y </TITLE>. Todo elemento es opcional, pero se recomienda utilizar siempre la etiqueta `<title>`, esta etiqueta permite definir el nombre de la página, en la siguiente imagen se ha ejecutado como title: “Java Intermedio – ICAI”.




```
<html>
	<head>
		<title>Java Intermedio - ICAI</title>
	</head>
	<body>
		<i>hola <b>mundi</b></i>
	</body>
</html>
```

Indicador de refresco del documento

```
<META http-equiv="refresh" content="número_segundos;url=URL_destino">
```

Esta etiqueta permite redireccionar desde una página a otra página (o la misma). Supongamos que nuestro sitio se actualiza cada 5 minutos con información nueva, entonces queremos que de forma automática el navegador refresque consigo misma. El siguiente código hace que después de 1 minutos (60 segundos) el sitio nos lleve automáticamente a la página oficial del ICAI: http//www.icai.ac.cr

```
<html>
	<head>
	    <title>Java Intermedio - ICAI</title>
	    <META http-equiv="refresh"
content="4;url=http://www.icai.ac.cr">
	</head>
	<body>
		<i>hola <b>mundi</b></i>
	</body>
</html>
```

Definición de colores

Antes de seguir, se debe explicar cómo se conforman los colores en los documentos HTML, estos colores se pueden identificar mediante dos maneras. La primera es utilizando el nombre (name) el cual se puede verificar en la siguiente tabla, primera columna. La segunda forma es mediante un código RGB (Red, Green, Blue) escrito en hexadecimal, mediante el siguiente formando: (#rrggbb) así por ejemplo el rojo #FF0000 en RGB quedaría (255, 0, 0).

Nombre	Código de color	Color mostrado
Black	#000000	Negro
Blue	#0000FF	Azul
Navy	#000080	Azul marino
Lime	#00FF00	Lima
White	#FFFFFF	Blanco
Purple	#800080	Púrpura
Yellow	#FFFF00	Amarillo
Olive	#808000	Oliva
Red	#FF0000	Rojo
maroon	#800000	Marrón
gray	#808080	Gris
fuchsia	#FF00FF	Fucsia
green	#008000	Verde
silver	#C0C0C0	Plata
aqua	#00FFFF	Agua

Cuerpo del documento HTML

En el cuerpo de un documento HTML es donde incluiremos los elementos que deseamos mostrar, estos elementos pueden ser texto plano, video, imágenes, links, hasta elementos más complejos como código JavaScript.

Las etiquetas `<BODY>` y `</BODY>` son las que van a delimitar el cuerpo de nuestro documento. Estas etiquetas contienen argumentos los cuales podemos personalizar para tener un sitio ajustado a nuestras necesidades.

La etiqueta `<BODY>` presenta una serie de atributos que van a afectar a todo el documento en su conjunto. Estos atributos nos van a permitir definir los colores del texto, del fondo, y de los hiperenlaces del documento. Incluso nos permitirán insertar una imagen de fondo en nuestra página.

```
<BODY background="URL" bgcolor="#rrggbb"
text="name" link="name" vlink="name" >
```

- `background="URL"`. Nos va a permitir mostrar una imagen como fondo de nuestro documento HTML. El camino a esta imagen vendrá especificado por la URL que definamos. Si la imagen no rellena todo el fondo del documento, ésta será reproducida tantas veces como sea necesario hasta completar todo el fondo.

- `bgcolor=#rrggbb"`. Nos va a permitir definir un color para el fondo de nuestro documento. Este atributo será ignorado si previamente hemos utilizado el atributo *background*.

- `text=#rrggbb`. Nos permitirá definir un color para el texto de nuestro documento. Por defecto es negro.

- `link=#rrggbb ó name.` Indica el color que tendrán los hiperenlaces que no han sido accedidos. Por defecto es azul. Como todavía no sabemos insertar hiperenlaces en nuestro documento vamos a dejar el ejemplo correspondiente para más adelante.
- `vlink=#rrggbb`. Indica el color de los hiperenlaces que ya han sido accedidos. Por defecto es púrpura.

Caracteres especiales

Se estarán preguntando cómo se podría escribir los caracteres especiales como el menor que o mayor que, puesto que estos son elementos propios de las etiquetas. También se estarán como se podría poner tildes si el HTML solamente soporta código ASCII. Para todos estos caracteres especiales existen códigos; así por ejemplo cuando se necesite insertar un menor que se debe escribir el código "<".

Símbolo	Código
< (menor que)	<
> (mayor que)	>
& (ampersand)	&
" (comillas)	"

Lo mismo sucede con las tildes y las eñes. Es necesario utilizar símbolos especiales para poder escribir el código respectivo, de esta forma si queremos escribir el apellido "Chavarría" es necesario escribir el código "Chavarrií". La siguiente tabla se detallan los caracteres especiales para tildes, eñes y otros símbolos del español.

LETRA	CÓDIGO	LETRA	CÓDIGO	LETRA	CÓDIGO
á	á	Á	Á	ñ	ñ
é	é	É	É	Ñ	Ñ
í	í	Í	Í	ü	ü
ó	ó	Ó	Ó	Ü	Ü
ú	ú	Ú	Ú	¿	¿
				¡	¡

Espaciado y saltos de línea

En HTML sólo se admite un único espacio en blanco separando cualquier elemento o texto, el resto de espacios serán ignorados por el navegador. Esto significa que los

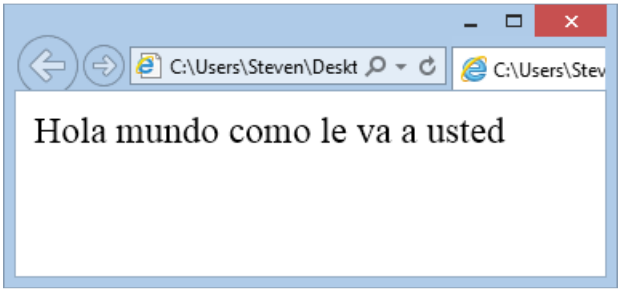
navegadores solamente van interpretar los espacios, saltos de línea y demás elementos de espacios una única vez:

```
<html>
      <head></head>
      <body>
            <p>
                  Hola    mundo

                  como

                  le va a            usted
            </p
      </body>
</html>
```

La ejecución del código anterior, genera la siguiente pagina web.



Para resolver estas características del HTML se han introducido, símbolos que permiten agregar espacios en blanco o saltos de línea.

Inclusión de espacios en blanco:

Nos permitirá la inclusión de más de un espacio en blanco entre dos textos consecutivos, de forma que estos espacios se muestren de forma efectiva en el navegador. Tendremos que incluir tantas expresiones   como espacios en blanco se deseen incluir.

`Hola     Mundo` equivale a “Hola mundo”

Salto de línea:

Nos permite dar un salto a la línea, es decir un cambio de renglón.

`Hola<br/>Mundo` equivale a

```
“Hola
mundo”
```

Cambio de párrafo: <p>

Permite definir un párrafo, introduciendo normalmente un espacio de separación de dos líneas con el texto siguiente al punto donde hayamos insertado la etiqueta <P>. La etiqueta de fin párrafo </P> es opcional. Aunque siempre es recomendable delimitar claramente el inicio y final de un párrafo. Además, cuando usemos esta etiqueta como cerrada <P>..... </P>, tenemos la posibilidad de incluirle el atributo align el cual indica al navegador la forma de justificar el texto incluido en el párrafo. El formato sería el siguiente:

```
<p align= [left | right | center | justify]>Texto de ejemplo</p>
```

EJEMPLO	RESULTADO
`<P ALIGN=RIGHT>TEXTO DE EJEMPLO</P>`	Texto de ejemplo
`<P ALIGN=CENTER>TEXTO DE EJEMPLO</P>`	Texto de ejemplo

Línea Horizontal: <hr>

Nos permite insertar una línea horizontal, cuyo tamaño podremos determinar a través de sus atributos. Si no especificamos ningún atributo dibujará una línea que ocupe el ancho de la pantalla del navegador. Su utilidad es la de permitirnos dividir nuestra página en distintas secciones.

El formato de la etiqueta con sus posibles atributos es:

`<HR align``=[left | right | center ] noshade size=n width=n>`

- **`align`**`= left / right / center.` Permite establecer la alineación de la línea a la izquierda, a la derecha o centrarla.
- `noshade`. No muestra la sombra de la línea, evitando el efecto de tres dimensiones.
- `size=n.` Indica el grosor de la línea en pixels.
- `width=n.` Especificará el ancho de la línea.

Ejemplo	Resultado
`<hr>`	

`<hr size=3 noshade>`	
`<hr size=5 width=50% align=right>`	
`<hr size=10 width=50% align=center>`	

Cabeceras: <h1>

En un documento HTML podemos incluir seis tipos distintos de cabeceras, que van a constituir normalmente el título y los distintos temas que forman el documento, aunque podremos utilizarlas en el punto que queramos del documento para resaltar cualquier texto. Estas cabeceras son definidas por las etiquetas <H1><H2><H3><H4><H5> y <H6>. La cabecera <H1>será la que muestre el texto de mayor tamaño, este tamaño irá disminuyendo hasta llegar a la cabecera </H6>. Como podemos ver en la siguiente.

Ejemplo	Resultado
`<H1>Cabecera 1</H1>`	Cabecera 1
`<H2>Cabecera 2</H2>`	Cabecera 2
`<H3>Cabecera 3</H3>`	Cabecera 3
`<H4>Cabecera 4</H4>`	Cabecera 4
`<H5>Cabecera 5</H5>`	Cabecera 5
`<H6>Cabecera 6</H6>`	Cabecera 6

Atributos del texto

Al texto de nuestro documento HTML le podemos aplicar distintos atributos (**negrita**, *cursiva*, subrayado), al igual que hacemos cuando trabajamos con el procesador de textos en la edición de nuestros documentos. Para aplicar estos atributos disponemos de distintas etiquetas.

Atributo	Etiqueta	Ejemplo	Resultado
Negrita	<B></B>	<B>Texto en negrita</B>	**Texto en negrita**
Cursiva	<I></I>	<I>Texto en cursiva</I>	*Texto en cursiva*
Teletype	<TT></TT>	<TT>Texto en modo teletype</TT>	Texto en teletype
Subrayado	<U></U>	<U>Texto subrayado</U>	***Texto subrayado***
Tachado	<S></S>	<S>Texto tachado</S>	Texto tachado
Superíndice		^{Texto en modo superíndice}	Texto en modo superíndice
Subíndice		_{Texto en modo subíndice}	Texto en modo subíndice
Centrado	<CENTER></CENTER>	<CENTER>Texto centrado</CENTER>	Texto centrado

Existen otras etiquetas que nos van a servir para especificar, de manera diferenciada, unidades lógicas de nuestro documento HTML tales como citas, direcciones de correo, etc. En algunos casos el formato obtenido con estas etiquetas de estilo lógico va a ser el mismo que con las anteriores, a las que se les denomina también etiquetas de estilo físico.

Etiqueta	Ejemplo	Resultado
<STRONG></STRONG>	<STRONG>Especifica texto resaltado (igual < B>)</STRONG >	**Especifica texto resaltado (igual <B>)**
<CITE></CITE>	<CITE>Indica una cita o título (igual < I>)</CITE >	*Indica una cita o título (igual <I>)*
<STRIKE></STRIKE>	<STRIKE>Texto tachado (igual < S>)</STRIKE >	Texto tachado (igual <S>)

Si queremos aplicar efectos más espectaculares a nuestro documento HTML, debemos variar el tamaño, el color y el tipo de letra del texto. La etiqueta que nos permite todo esto es <FONT>...</FONT>, por medio de sus atributos size, color y face:

```
<FONT size="n" olor="#rrggbb" face="nombre de font" >
```

- `size="n".` El atributo size nos permite especificar un tamaño determinado para la fuente del texto incluido entre las etiquetas de inicio y fin, el cual puede estar entre 1 y 7. El texto de tamaño normal equivale a la fuente de tamaño 3 (fuente base). Por tanto, si especificamos `size=+2`, el tamaño de la fuente será 5. Y si especificamos `size= -1`, el tamaño será 2.

Ejemplo	**Resultado**
`<FONT size=2>Tamaño 2< /FONT >`	Tamaño 2
`<FONT size=+2>Tamaño 5 (3+2)</FONT>`	Tamaño 5 (3+2)
`<FONT size=-1>Tamaño 2 (3-1)< /FONT >`	Tamaño 2 (3-1)

- `face="nombre de font".` Nos va a permitir escribir texto con el tipo de letra que le especifiquemos. En el caso de que el tipo de letra que le hayamos especificado no esté cargada en la computadora que lee la página, se usará el font por defecto del navegador.

Ejemplo	**Resultado**
`<FONT face=Tahoma>Tipo de letra Tahoma</FONT>`	Tipo de letra Tahoma
`<FONT size=4 color=blue face=Tahoma>Texto azul, de tamaño 4 y Tahoma</FONT>`	Texto azul, de tamaño 4 y Tahoma

Listas

Podemos representar elementos en forma de lista dentro de nuestros documentos de una forma muy sencilla y con una gran versatilidad. Estas listas podrán incluir cualquiera de los elementos HTML e incluso podemos definir listas anidadas, es decir, listas dentro de listas. HTML nos permite crear tres tipos distintos de listas:

- Listas no numeradas
- Listas numeradas
- Listas de definiciones

Listas no numeradas: <UL>

Con este tipo de listas podemos especificar una serie de elementos sin un orden predeterminado, precedidos de una marca o viñeta que nosotros mismos podemos

definir. Para la definición de los límites de la lista utilizaremos la etiqueta <UL>.... </UL>, y para determinar cada uno de los elementos que la componen usaremos la etiqueta <LI>. El formato es el siguiente:

```
<UL type=["disk"|"circle"|"square"]>
  <LH>Título de la lista</LH>
     <LI>Elemento 1
     <LI>Elemento 2
     ...
  <LI>Elemento n
</UL >
```

Listas numeradas: <OL>

Con este tipo de listas podemos especificar una serie de elementos numerados según el lugar que ocupan en la lista. Para la definición de los límites de la lista utilizaremos la etiqueta <OL>.... </OL>, y para determinar cada uno de los elementos que la componen usaremos la etiqueta <LI>. El formato es el siguiente:

```
<OL start="n" type="Tipo de lista">
  <LH>Título de la lista</LH>
   <LI>Elemento 1
   <LI>Elemento 2
   ...
   <LI>Elemento n
</OL >
```

Listas de definiciones: <DL>

Estas listas nos van a servir para especificar una serie de términos y sus definiciones correspondientes. Para la definición de la lista usaremos la etiqueta <DL>.... </DL>, para especificar los términos usaremos la etiqueta <DT> y para especificar la definición correspondiente a cada término usaremos la etiqueta <DD>. El formato es el siguiente:

```
<DL>
       <LH>Título de la lista</LH>
       <DT>Término 1
       <DD>Definición   1
       <DT>Término      2
       <DD>Definición   2
       ....
       <DT>Término n
       <DD>Definición n
</DL>
```

Hiperenlaces

Los hiperenlaces son *enlaces de hipertexto* que nos van a permitir acceder de manera directa a otros documentos HTML independientemente de su ubicación, o a otras zonas dentro de nuestro propio documento.

Enlaces a otras páginas: <a href=...>

Con este tipo de hiperenlaces vamos a poder acceder tanto a otras páginas que estén ubicadas dentro de nuestro propio sistema como a páginas ubicadas en puntos muy distantes del globo. El formato de este tipo de hiperenlaces es:

```
<a href="url a la que se accede">texto del hiperenlace</a>
```

Con el atributo `href` vamos a especificar la URL del documento al que se pretende acceder. El texto contenido entre las etiquetas de comienzo y fin nos va a servir para definir el hiperenlace, por lo que debería ser clarificador del contenido del documento con el que vamos a enlazar. Esta definición aparecerá resaltada normalmente en azul y subrayada. En la mayoría de los navegadores esta definición del hiperenlace es sensible, por lo que cuando el cursor pasa por encima del mismo este cambia de aspecto indicándolo.

Enlaces dentro de la misma página: <A name=...>

Este tipo de hiperenlaces nos va a permitir marcar distintas zonas del documento activo con el objeto de tener un acceso directo a las mismas. Una buena utilidad de este tipo de enlaces radica en la creación de índices para documentos largos, de forma que si pinchamos en el hiperenlace correspondiente al título de un capítulo determinado, el navegador saltará automáticamente hasta el comienzo de dicho capítulo.

Para la creación de estos hiperenlaces, debemos seguir dos pasos:

1. Marcar las distintas zonas o secciones del documento. Esto lo haremos con el parámetro name:

   ```
   <A name="Identificador de sección">Texto de la sección</A>
   ```

   A cada sección le asignaremos un identificador distinto, para poder referenciarlas posteriormente de manera inequívoca.

2. Especificar un enlace a cada una de las secciones que hayamos definido.

   ```
   <A href="#Identificador de sección">Texto del enlace a la sección</A>
   ```

Si nos creamos una página HTML con distintos capítulos, podríamos crear una sección para cada uno de ellos, de forma que si pinchamos en el hiperenlace correspondiente al Capítulo 1, el navegador saltaría directamente a la sección correspondiente al **Capítulo 1**.

Tablas

HTML nos va a permitir la inclusión de cualquiera de los elementos de nuestra página (texto, imágenes, hiperenlaces, listas, etc.), dentro de una tabla. Gracias a lo cual conseguiremos dar una mayor y mejor estructuración a los contenidos de nuestros documentos. Además, la definición de las tablas en HTML es muy abierta, pudiendo en

cualquier momento redimensionar la tabla, es decir, cambiar su número de filas o de columnas, cambiar el tamaño de alguna de sus celdas, etc.

La etiqueta que nos va a permitir la definición de tablas es <TABLE> </TABLE>.

El formato general de la etiqueta sin ningún argumento, es el siguiente:

```
<TABLE>
  <TR >
    <TH>Contenido de la celda </TH>
    <TD>Contenido de la celda < /TD >
   </TR>
</TABLE>
```

Vamos a analizar cada una de estas etiquetas de forma separada:

1. `<TABLE> </TABLE>`: Definición general de la tabla. Dentro de ella definiremos las filas y columnas que la constituyen, pudiendo incluso definir tablas dentro de tablas, es decir, tablas anidadas.
2. `<TR></TR>`: Definición de las filas de la tabla. Por cada etiqueta <TR> que incluyamos se creará una fila en la tabla. No será necesario indicar la etiqueta de cierre.
3. `<TD></TD>`: Definición de cada una de las celdas de la tabla. Vemos que estas etiquetas están contenidas dentro de otra etiqueta de definición de fila, de forma que por cada etiqueta `<TH>` que incluyamos se creará una celda dentro de la fila correspondiente.

Campos personalizables de las tablas

Las tablas tienen las siguientes propiedades, cada una de ellas modifica un elemento visual de las tablas. Su sintaxis es <table propiedad="valor"…>…</table>

- `cellspacing="n".` Indica el espacio en puntos que separa a las celdas contenidas dentro de la tabla, siendo 2 por defecto.
- `width="n".` Indica la anchura de la tabla en puntos o en % en función del ancho de la ventana del visualizador. Si no indicamos este argumento, el ancho de la tabla se ajustará al tamaño del contenido de las celdas. El siguiente código genera una tabla de una sola línea (1 columna y 1 fila) con border recto al 50% del tamaño de la pantalla.

```
<TABLE border=4 width=50%><TR><TD>Mi primera tabla</TABLE>
```

Mi primera tabla

- `height="n"`. Nos permite definir la altura de la tabla en puntos o en % de la altura de la ventana del visualizador. Si indicamos este argumento, lo

recomendable es darlo en puntos ya que es más complicado adecuarnos a la altura de la ventana del visualizador. Al igual que en el caso anterior, si no especificamos este argumento la altura se adecuará al contenido de las celdas.

- `bgcolor= "#rrggbb" o "nombre del color"`. Nos permite definir un color de fondo para todas las celdas de la tabla.

  - `align="left/right/center"`. Con este argumento vamos a indicar la alineación horizontal (*left*:izquierda, *right*:derecha, *center*:centrado) del contenido de todas las celdas de la fila correspondiente.

- `valign="top/middle/bottom".` Con este argumento vamos a indicar la alineación vertical (*top*:arriba, *middle*:centrado, *bottom*:abajo) del contenido de todas las celdas de la fila correspondiente. Posteriormente veremos cómo podremos especificar la alineación del contenido de cada celda de la fila por separado.

- `rowspan="n".` Con este argumento podemos lograr que una celda concreta abarque más de una fila, ya sabemos que por defecto una celda ocupa una sola fila.

Ejemplo	**Resultado**	

`<TABLE border=4>`
`<TR ><TH>Cabecera 1<TH>Cabecera 2`
`<TR ><TD rowspan=2>Celda 1.1<TD>Celda 1.2`
`<TR><TD>Celda 2.2`
`< /TABLE >`
	Cabecera 1	Cabecera 2

	Celda 1.1	Celda 1.2
		Celda 2.2

- `colspan="n".` Con este argumento podemos lograr que una celda se expanda a más de una columna.

Ejemplo	**Resultado**	

`<TABLE border=4>`
`<TR ><TH>Cabecera 1<TH>Cabecera 2`
`<TR ><TD>Celda 1.1<TD>Celda 1.2`
`<TR><TD colspan=2 align=center>Celda 2.2`
`< /TABLE >`
	Cabecera 1	Cabecera 2

	Celda 1.1	Celda 1.2
	Celda 2.2	

Formularios

El formulario es el elemento de HTML que nos va a permitir interactuar con los visitantes de nuestras páginas web, de forma que podemos solicitarle información al usuario y procesarla. De esta forma, podremos crear en nuestras páginas: *encuestas* para solicitar la opinión del visitante sobre algún tema e incluso sobre el contenido de nuestra propia página web, *cuestionarios* para evaluar la asimilación de contenidos sobre un tema concreto que se trate en la página, etc.

El contenido de la información introducida por medio del formulario será enviado a la dirección URL donde resida el programa que se encargará de procesar los datos. A este tipo de programas externos se les denomina programas CGI (Common Gateway Interface).

La etiqueta HTML que nos va a permitir la creación de formularios es <FORM>. Su formato es el siguiente:

```
<FORM action="URL_CGI" method="post/get">
..... Elementos del formulario .....
</FORM>
```

`action="URL_CGI"`. A través de este argumento indicaremos, como ya hemos mencionado, la dirección del programa que va a tratar la información suministrada por el formulario.

`method="post/get".` A través de este argumento indicaremos el método empleado para transferir la información generada por el formulario.

Si indicamos *post*, se producirá la modificación del documento destino, como es el caso de enviar la información a una dirección de correo electrónico. Mediante el método *get* no se producirán cambios en el documento destino.

Entrada básica de datos

Para definir los distintos tipos de campos básicos de entrada de datos usaremos la etiqueta <INPUT>.

El formato es el siguiente:

```
<INPUT type= text / password / checkbox / radio / hidden / submit / image
/ reset name="Variable" value="Valor inicial">
```

- El argumento `type` determinará el tipo de campo de entrada que estamos insertando. A continuación describiremos cada uno de ellos.
- El argumento `name` específica el nombre de la variable que tomará el valor introducido en el campo.
- El argumento `value` específica el valor por defecto que tendrá el campo.

Seguidamente se explicaran los tipos de campos Input, cada uno de ellos tiene una función específica.

Texto corto: type=text

Con este argumento vamos a indicar que el campo a introducir será un texto. El formato sería:

```
<INPUT type= text name="Variable" value="Valor inicialización"
size="Tamaño" maxlength="Longitud máxima">
```

El significado de los nuevos atributos es:

- `size="Tamaño"`. Indicaremos el tamaño en caracteres de la ventana de introducción de texto.
- `maxlength="Longitud máxima".` Indicaremos el número máximo de caracteres a introducir en el campo.

Claves: type=password

Con este argumento indicamos que el campo a introducir será una palabra Contraseña, por lo que los caracteres que se introduzcan serán sustituidos por asteriscos en la visualización por pantalla. El formato es:

```
<INPUT type= password name="Variable" value="Valor inicialización"
size="Tamaño" maxlength="Longitud máxima">
```

Botones de selección: type=checkbox

El checkbox es un botón que presenta dos estados: marcado (1) y desmarcado (0). Podremos variar su estado simplemente pinchando con el ratón. El formato es:

```
<INPUT type= checkbox name="Variable" value="Valor" checked>
```

Si especificamos el argumento checked, el botón aparecerá marcado por defecto. Si incluimos el argumento value, cuando el botón esté marcado su variable asociada adquirirá el valor dado por value.

Selección entre varias opciones: type=radio

Este argumento se usa cuando hay que hacer una selección entre varias alternativas excluyentes, pudiéndose seleccionar únicamente una de las alternativas. Debemos incluir una etiqueta radio por cada una de las posibles alternativas. El formato es:

```
<INPUT type= radio name="Variable" value="Valor" checked>
```

Si especificamos el argumento checked, el botón aparecerá marcado por defecto. En este caso únicamente uno de los botones de radio podrá aparecer marcado, ya que se usa para el caso de opciones excluyentes. Cuando un botón esté seleccionado la variable asociada a la lista de botones adquirirá el valor dado por value.

Campos ocultos: type=hidden

Este tipo de campos no son visibles para el usuario. Su uso tiene sentido en el caso de enviar algún tipo de información que no deba ser visualizada o variada por el lector de nuestra página Web. El formato es:

```
<INPUT type= hidden name="Variable" value="Valor" >
```

Con esta etiqueta asignaríamos a la "Variable" referenciada por name el "Valor" de value y se mandaría junto con el formulario sin que el usuario de la página se entere de nada.

Botón de envío de datos: type=submit

Con este argumento especificamos un botón en el que al pulsar, los datos serán enviados al programa o dirección de correo encargada de procesar la información recogida por el formulario. El formato es:

```
<INPUT type= submit value="Mensaje a mostrar">
```

Botón de borrado de datos: type=reset

Con este argumento especificamos un botón que al ser pulsado borrará el contenido actual de todos los campos, dejándolos con sus valores por defecto. Su formato es:

```
<INPUT type= reset value="Texto del botón">
```

Con el parámetro value especificamos el texto que etiquetará al botón.

Entrada datos en múltiples líneas:

En un formulario también podremos introducir un campo de texto que abarque varias líneas. Para ello usaremos la etiqueta <TEXTAREA> </TEXTAREA>. Su formato es el siguiente:

```
<TEXTAREA name="Variable" rows=Filas cols=Columnas> Contenido por
defecto. < /TEXTAREA >
```

Los argumentos rows y cols nos van a permitir especificar, respectivamente, el número de filas de texto visibles y el número de columnas de texto visibles en el campo.

Introducción a JavaScript

JavaScript es un lenguaje de programación el cual fue desarrollado para programar páginas web dinámicas. Una página web dinámica es una página que se construye de forma dinámica. Pensemos en una página estática como Google, es al final y al cabo un sitio donde la apariencia no cambia. Si analizamos Facebook, nos damos cuenta que la misma se construye de forma personalizada por cada usuario, además las notificaciones son actualizadas periódicamente sin intervención del usuario (a esto se le llama Ajax y es una aplicación del código JavaScript).

JavaScript es un lenguaje interpretado, eso significa que dicho lenguaje se ejecuta por primera vez cuando se consume el código del lado del cliente, esto permite que no

exista tipificación de datos, lo que le da un gran poder para desarrollar aplicaciones muy flexibles.

JavaScript es tan complejo como todo un curso de lenguajes, es por ello que se recomienda que el estudiante realice la lectura por cuenta propia. Para ello se deja el link de un excelente libro gratuito.

Es sumamente recomendado, que el estudiante realice una lectura del siguiente ebook: Jahttp://www.jesusda.com/docs/ebooks/introduccion_javascript.pdf

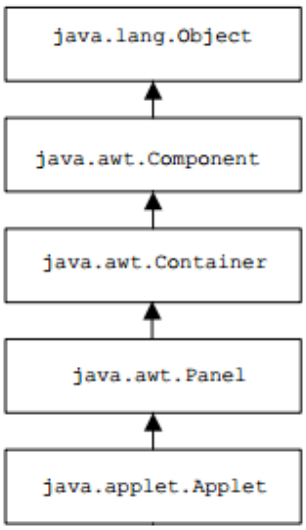
Capítulo 4
Applets

Los applets son aplicaciones escritas en Java, las cuales se pueden empotrar dentro de una ventana. La ilusión final de este proceso es tener una aplicación funcional, en una ventana web. Podemos encontrar Applets en diversos sitios, un ejemplo muy común es el Banco Nacional de Costa Rica, el cual utiliza un Applet para verificar la clave de ingreso al sitio transaccional.

Estructura básica de un Applet

Se puede definir un applet como un programa Java que no puede ser ejecutado por sí mismo, en vez de ello debe ser ejecutado por un navegador u otra aplicación Java. Los applets son seguros, lo que les permite correr dentro de un navegador, a partir del archivo con extensión .class generado por la compilación del applet. Un applet cualquiera es una subclase de la clase java.applet.Applet; es decir sus comportamientos son heredados de la clase Applet.

Todos los applets son subclases de Applet, asi que se debe importar el paquete `java.applet`. La siguiente figura ilustra la jerarquia de herencias de la clase Applet, lo cual da una idea de lo que puede hacer, o el comportamiento de un applet. La clase Applet extiende a la clase Panel de AWT, a su vez extiende a la clase Container, que extiende a la clase Component. Todas estas clases suministran las bases para el desarrollo de interfaces gráficas basadas en ventanas.



Lo que significa que un Applet, hereda los métodos clásicos de los objetos, pero además de un panel, con esto se logra que dicho Applet sea un panel grafico donde podamos modificar y agregar controles como se vio en el curso de Java Introductorio del ICAI.

La clase Applet

La siguiente tabla muestra métodos definidos por la clase Applet en Java2, y proporcionan todo lo necesario para ejecutar un applet desde su inicio hasta su finalización, así como los métodos para cargar y exhibir imágenes y reproducir sonidos.

void **destroy( )**	Este método es llamado por el navegador justamente antes de que el applet termine su ejecución. Su función es liberar la memoria asignada a la aplicación.
AppletContext **getAppletContext( )**	Devuelve el contexto asociado con el applet, lo que le permite consultar y afectar el medio ambiente en el cual se ejecuta.
String **getAppletInfo( )**	Devuelve unacadena que describe el applet.
AudioClip **getAudioClip(URL *url*)**	Devuelve un objeto AudioClip especificado en la direccion *url*.
AudioClip **getAudioClip(URL *url*, String *name*)**	Devuelve un objeto AudioClip especificado en la direccion *url*, con el nombre *name*.
URL **getCodeBase( )**	Devuelve el URL del applet que lo llama.
URL **getDocumentBase( )**	Devuelve el URL del documento HTML que invoca al applet.
Image **getImage**(**URL *url*)**	Devuelve un objeto **Image** en la direccion indicada por *url*.
Image **getImage(URL *url*, String *name*)**	Devuelve un objeto **Image** en la direccion indicada por ***url***, con el nombre ***name***.
Locale **getLocale( )**	Suministra **Locale** del applet si se ha asignado.
String **getParameter(String *name*)**	Devuelve los parametros de la applet asociados con ***name*** desde una página HTML.
String[ ][ ] **getParameterInfo( )**	Devuelve una tabla **String** que describe los parametros reconocidos por la applet.Cada entrada de la tabla se forma de tres cadenas:

	nombre del parametro, tipo y explicación.
void **init( )**	Llamado por el navegador o el appletviewer, cuando el applet es cargada en el sistema y comienza su ejecución.
boolean **isActive( )**	Devuelve **true** si la applet esta activa y **false** si esta parada.
static AudioClip **newAudioClip(URL url)**	Toma un archivo de sonido en la direccion *url.*
void **play(URL *url*)**	Reproduce el archivo de sonido especificado en la direccion absoluta *url*.
void **play(URL *url*, String *name*)**	Reproduce el archivo de sonido con el nombre *name* especificado en la direccion *url*.
void **resize(Dimension *dim*)**	Redimensiona el tamaño del applet de acuerdo a ***dim***. **Dimension** es una clase que contiene dos campos enteros: **width** y **height**.
void **resize(int *ancho*, int *alto*)**	Redimensiona el tamaño del applet de acuerdo a las dimensiones **ancho** y **alto**.
void **setStub(AppletStub *stub*)**	Un stub es una pequeña parte de código que proporciona el enlace entre el applet y el navegador. El método hace que el objeto stub sea el resguardo del applet. Es usado por el interprete de Java y normalmente no lo utiliza el applet.
void **showStatus(String *msg*)**	Para hacer que el argumento `msg` de tipo `String` sea visualizado en en la "ventana de estado" del applet.
void **start( )**	Llamado por el navegador o appletviewer para informar que el applet debe iniciar su ejecución.
void **stop( )**	Llamado por el navegador o appletviewer para informar que el applet debe suspender su ejecución.

Tabla de métodos de la clase Applet (Universidad del Valle, 2013)

El ciclo de vida de un Applet

El ciclo de vida de un Applet, tiene relación con los pasos que siguen un Applet, para su construcción, instanciación, procesamiento y cierre del mismo.

La clase Applet provee el marco para la ejecución de applets, definiendo los métodos que el sistema llama cuando ocurren los eventos más importantes del ciclo de vida. La mayoría de las applets sobreponen algunos o todos los métodos a fin de responder adecuadamente a los eventos que se presenten.

La ejecución de un applet requiere, en primer lugar, que se haya editado su cuerpo, estructura y comportamiento, y generado el respectivo archivo **.class** a partir de su compilación. Se debe escribir un archivo con extension **.html**, el cual tiene dentro de su cuerpo una etiqueta `<applet>` dentro de la cual se llama al archivo **.class** del applet. Este programa es llamado por un navegador o el visualizador *appletviewer* que viene con el JDK.

Veamos ahora los diferentes eventos que se presentan durante el ciclo de vida de un applet.

El ciclo de vida se inicia con el llamado a un navegador, o el *appletviewer* (visor de applets del Java) en la ventana de comandos. Veamos los pasos que se llevan a cabo:

1. El navegador o el *appletviewer*, leen el programa HTML y buscan la etiqueta `<APPLET>` que contiene el llamado al archivo **.class** del applet. El siguiente es el código HTML que se utilizará para llamar al applet

```
<HTML>
   <APPLET CODE = "SimpleApplet.class" WIDTH = 300 HEIGHT = 150>
   </APPLET >
</HTML >
```

2. El navegador descompone la etiqueta `<APPLET>` para localizar el `code` posiblemente el atributo `CODEBASE`. Ambos atributos se verán en detalle más adelante. En nuestro caso se ha utilizado el atributo `CODE`.

3. El navegador convierte los bytes descargados, del directorio o de la dirección `URL`, en la forma de una clase Java.

4. El navegador crea una instancia de la clase cargada para formar un objeto applet. Esto requiere que el applet tenga un constructor sin argumentos.

5. El navegador llama el método `init()` del applet. Para el caso del applet `SimpleApplet.class` se tiene el siguiente:

```
public void init() {
buffer = new StringBuffer(); agregarItem("Inicializando applet...
");
}
```

6. El navegador llama luego el método `start()` del applet.

```
public void start() {
    agregarItem("Arrancando applet... ");
}
```

7. Mientras el applet está ejecutándose, el navegador pasa cualquier evento que tenga que ver con el applet en su interacción con el usuario, por ejemplo movimiento y presión del ratón, presión de teclas, al método `handleEvent()`, que es el manejador de eventos del applet. Se lleva a cabo la actualización de eventos para indicar al applet que necesita repintarse o sea llamar al método `repaint()`. En el caso del applet `SimpleApplet.class` se dibuja un rectángulo alrededor del área de la ventana del applet, y dentro de este dibuja el estado de la cadena `buffer`, por medio del método `paint()`. Se ilustra además el método de usuario `agregarItem()`, el cual genera una salida en la pantalla de comandos por cada interrupción del applet, el cual llama al método repaint(), que se encarga de volver a dibujar el contenido de la ventana, cuando requiera ser restaurada.

```
void agregarItem(String palabra) {
  System.out.println(palabra);
  buffer.append(palabra); repaint();
}

public void paint(Graphics g) {
  g.drawRect(0, 0, getSize().width - 1,getSize().height - 1)
;
  g.drawString(buffer.toString(), 5, 15) ;
}
```

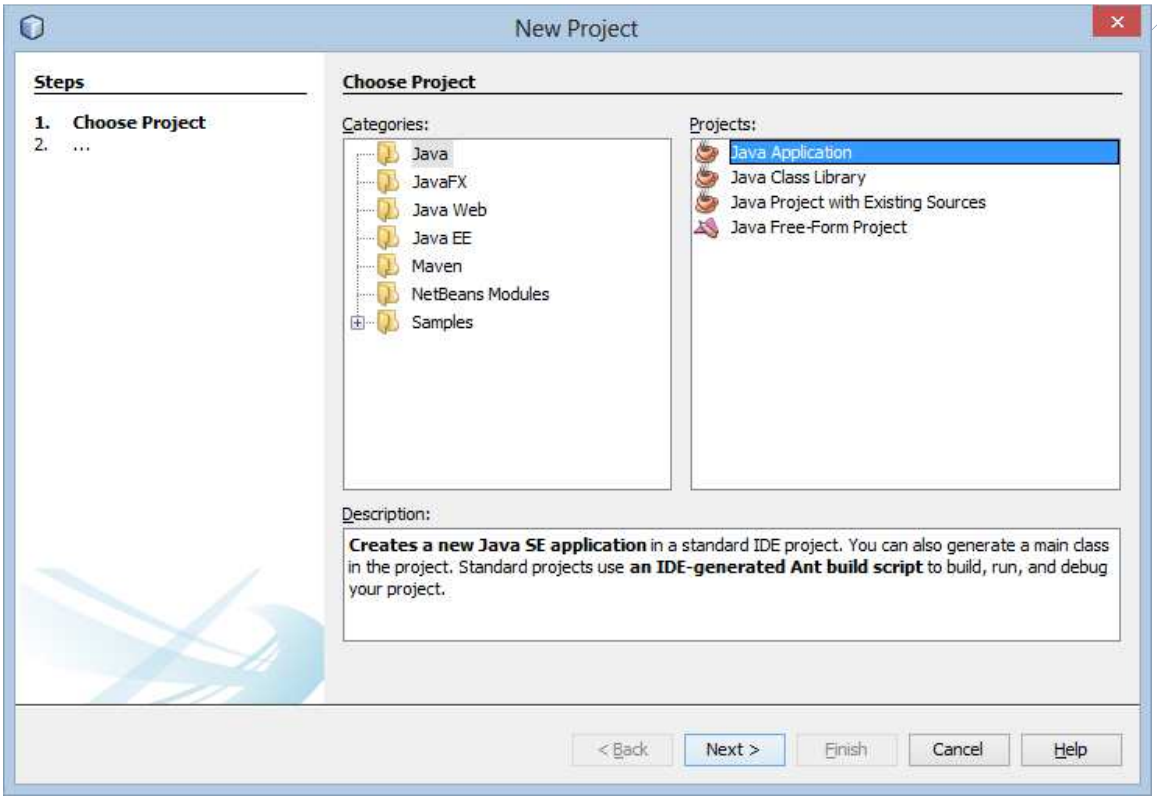
8. El navegador llama al método `stop()`. Para el caso del applet `SimpleApplet.class` se ejecuta cuando se detiene el applet.

```
public void stop() {
agregarItem("Deteniendo applet... ");
}
```

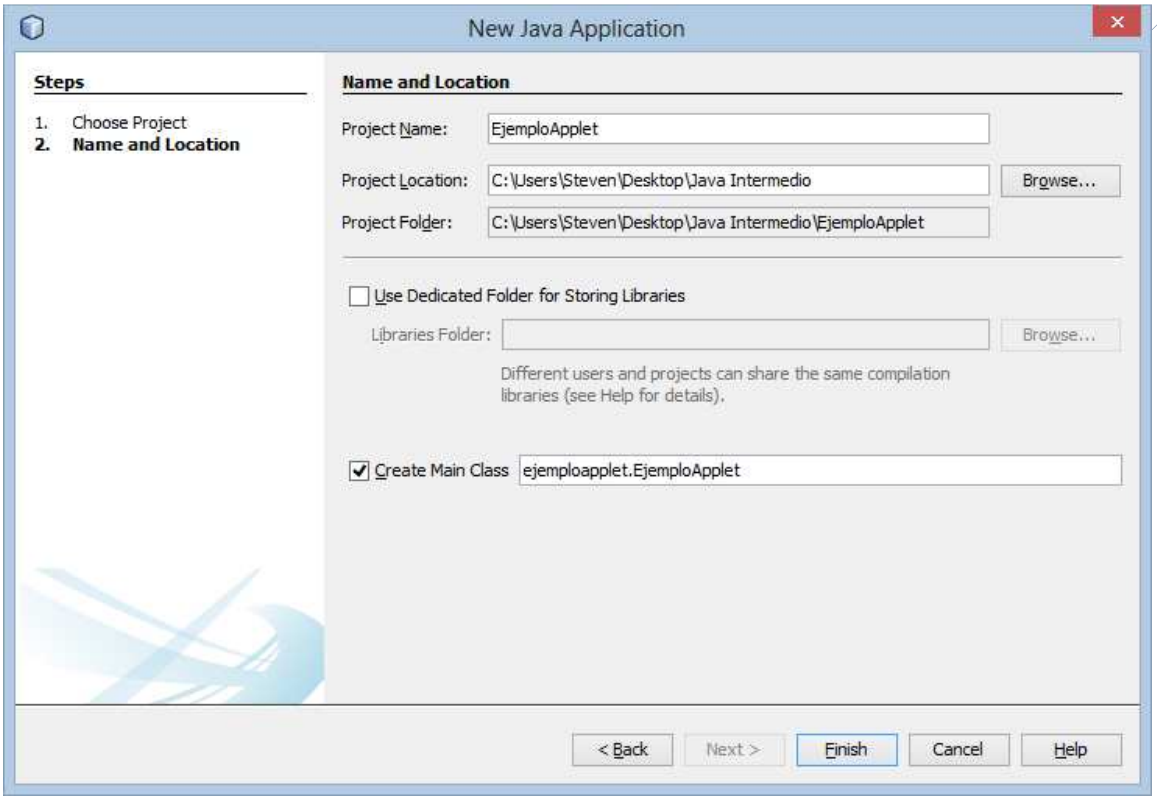
9. El navegador llama al método destroy() Se ejecuta cuando se cierra el applet.

Ejemplo de cómo crear un Applet en NetBeans

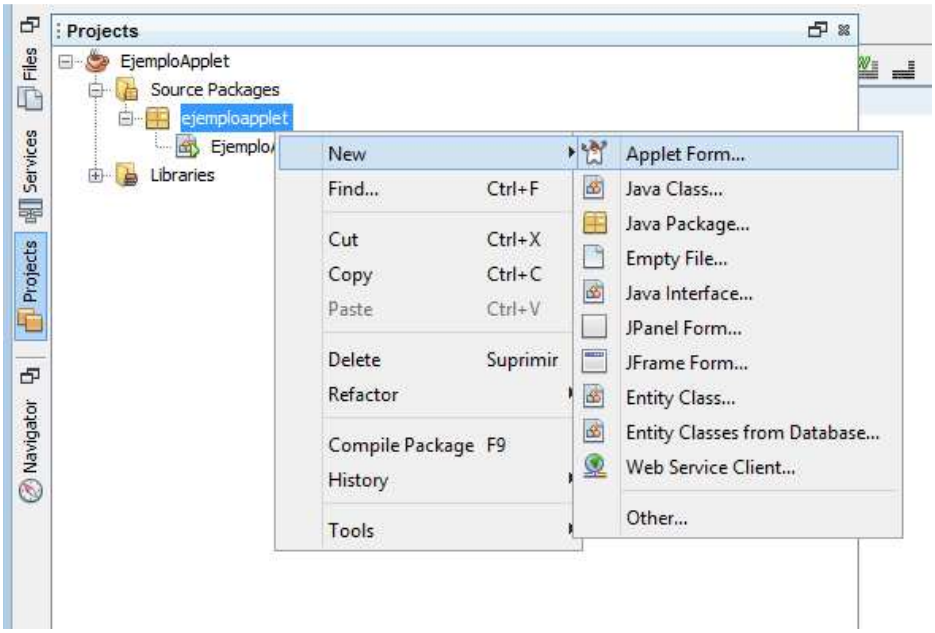
El primer paso para crear un Applet, consiste en crear un nuevo proyecto en NetBeans, a la hora de escoger el tipo de proyecto deberá seleccionar un “Java Application”. En el nombre del proyecto le ponemos “EjemploApplet” y le damos finalizar.

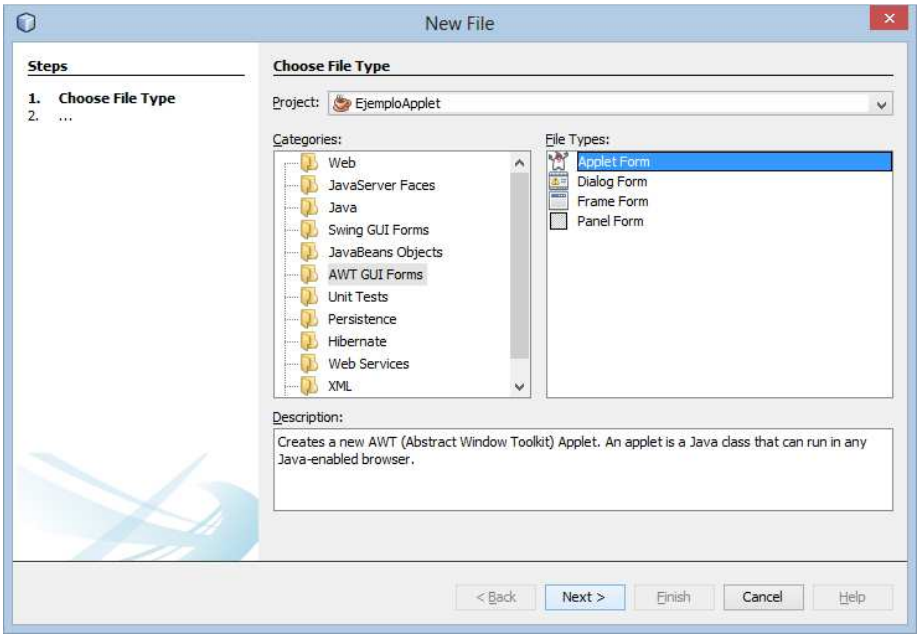


Finalizamos el proceso.

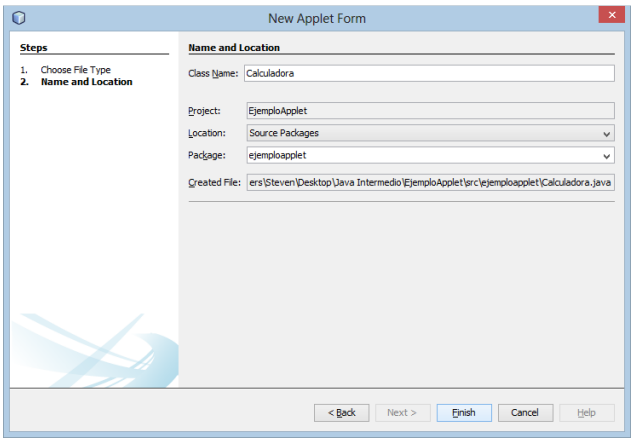


El siguiente paso consiste en crear un elemento de tipo Applet Form, eso lo puedes encontrar dándole click derecho al proyecto, nuevo, Applet Form. Si no aparece aquí usted deberá buscarlos en otros, AWT GUI Forms, Applet Form.

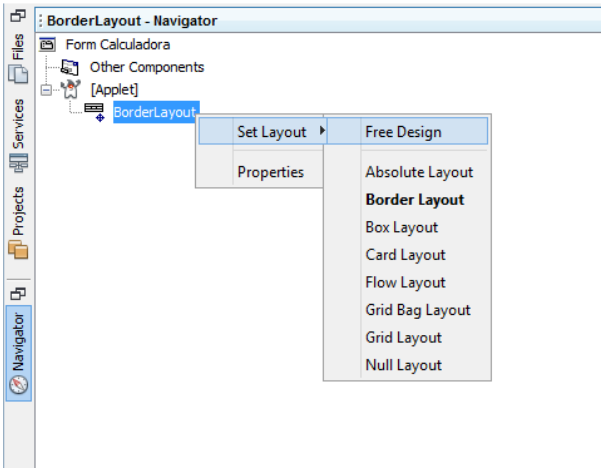




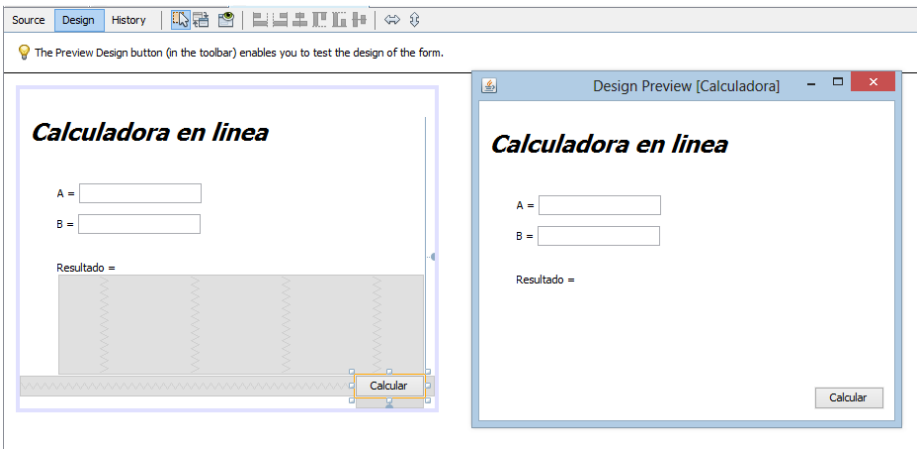
Creamos la ventana Applet Form, con el nombre de "Calculadora". Dele Finalizar para crear la clase Applet.



Ahora NetBeans creará una ventana donde usted podrá arrastrar botones, cajas de texto y demás elementos. Para tener un mejor control del espacio se recomienda habilitar es estilo libre al BorderLayout. Esto lo puede habilitar desde la pestaña Navegator ubicada a la izquierda; click derecho Free Design.



Procesamos a crear la siguiente inteface de usuario, para ello usted tendrá una paleta de objeto ubicado a la derecha de su ventana del IDE. Arrastre un título, dos cajas de texto (jField), un botón y el objeto para mostrar el resultado (jText).



Tendrá una paleta de objeto ubicado a la derecha de su ventana del IDE. Arrastre un título, dos cajas de texto (jField), un botón y el objeto para mostrar el resultado (jText).

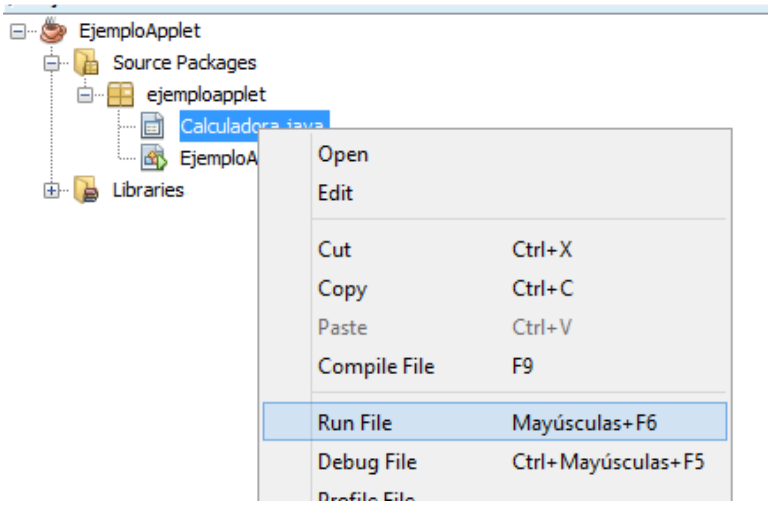
Implemente la lógica del botón, para ello lo que se quiere es poder multiplicar dos simples campos de texto, y dejar el valor en el campo de salida respectivo. Utilice como ejemplo base el siguiente código:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
     // TODO add your handling code here:
     try{
         jLabel4.setText(Integer.toString(
                 Integer.parseInt(jTextField1.getText())
                 *
                 Integer.parseInt(jTextField2.getText())));
     }
     catch (Exception ex) {
         jLabel4.setText("Escriba valores numericos validos.");
     }
 }
```

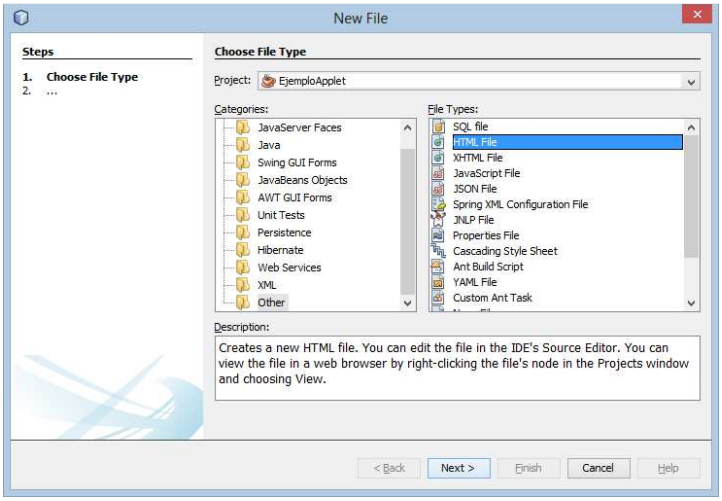
Observe que el código de la clase, tiene la siguiente herencia:

```
public class Calculadora extends java.applet.Applet
```

Es justamente esta línea lo que convierte la clase, en un applet que se pueda empotrar en un navegador. El siguiente paso es compilar el Applet, para ello usted deberá en NetBeans ejecutar el código, dándole click derecho a Calculadora y Run File.



Una vez que tengamos el Applet compilado, vamos a empotrarlo en una página web. Para ello usted deberá agregar un HTML File en NetBeans, el procedimiento es simple; deberá agregar elemento, otros, y bajas hasta donde dice otros como se ve en la siguiente imagen.



Póngale de nombre a la página el que guste. Termine el proceso dándole Finalizar. Esto debió crear un HTML con las siguientes líneas:

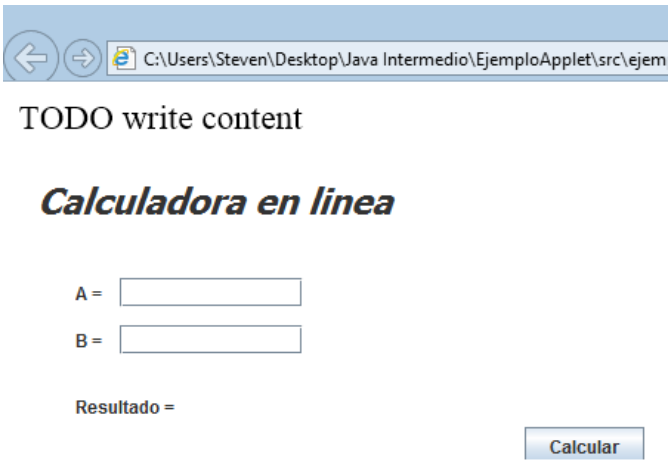
```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Agregamos la siguiente etiqueta:

```
<applet CODE = "ejemploapplet.Calculadora.class"
archive="EjemploApplet.jar" WIDTH = 300 HEIGHT = 150></APPLET >
```

Con ella ya podremos ejecutar nuestra aplicación Java en cualquier navegador web, donde se tenga instalado la máquina virtual de Java.



Capítulo 5
Uso e instalación de Servidores Web

Definición de servidor web

Un servidor web es un software que procesa una aplicación del lado del servidor, en el caso de las aplicaciones Web, como las que utiliza Java es necesario instalar un servidor de aplicaciones.

Las funciones de dicho servidor consiste en la administración de las conexiones bidireccionales unidireccionales- Sincrónicas y asincrónicas, con el cliente (navegador) la idea es que las paginas se construyan del lado del servidor, una vez que se crea la página la misma será enviada al cliente en formato HTML.

Supongamos que entramos al sitio transaccional de un banco, está claro que la información no la tiene el cliente, en vez de ello la tiene el servidor en una base de datos. Una vez que el servidor recibe la solicitud de la persona, la página se construye

en el servidor, el servidor creará la página web utilizando algún lenguaje (PHP, JSP, .net, etc) la cual posteriormente será enviada al cliente en formato HTML.

Para desarrollar en JSP se pueden instalar cualquiera de los siguientes servidores:

1. Apache Tomcat
2. Glassfish

Para ambos casos, el instalador de NetBeans nos preguntara cuál de los dos (pueden ser ambos simultáneamente) queremos instalar en nuestra máquina. La instalación de ellos es fundamental para poder ejecutar la aplicación escrita en JSP.

Instalación de servidor web

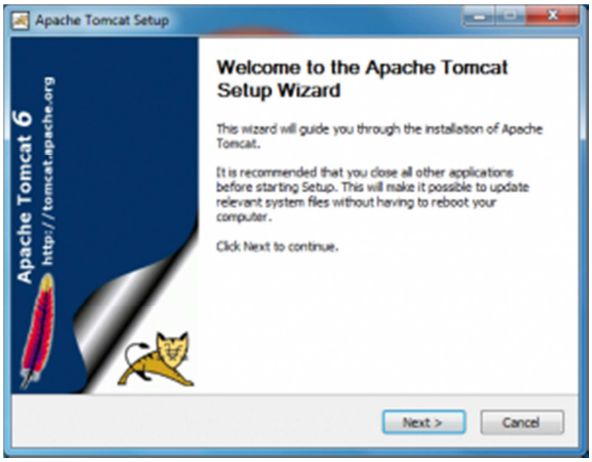
Supongamos que no tenemos instalado Tomcat en nuestras maquinas, esto podría suceder cuando al instalar NetBeans nos saltamos el paso de escoger el servidor de páginas web u JSP. También podría suceder este caso, cuando necesitemos instalar nuestra aplicación en modo de producción.

Requisitos previos para instalar Tomcat:

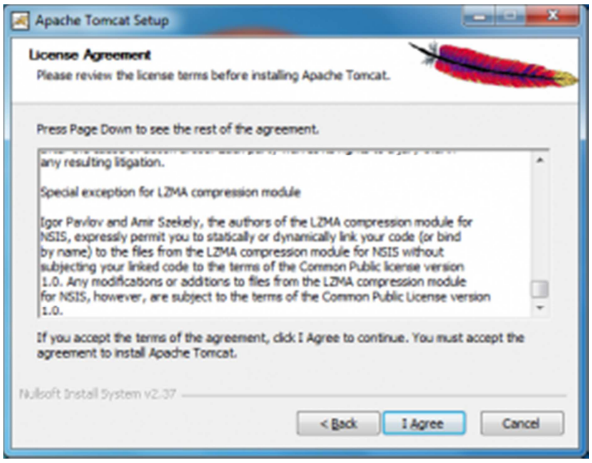
1. Descargar el JDK desde http://java.sun.com/javase/downloads/index.jsp
2. Descargamos el Tomcat desde http://tomcat.apache.org/download-70.cgi
3. Escogemos la distribución 32-bit/64-bit Windows Service Installer

Instalación del Tomcat

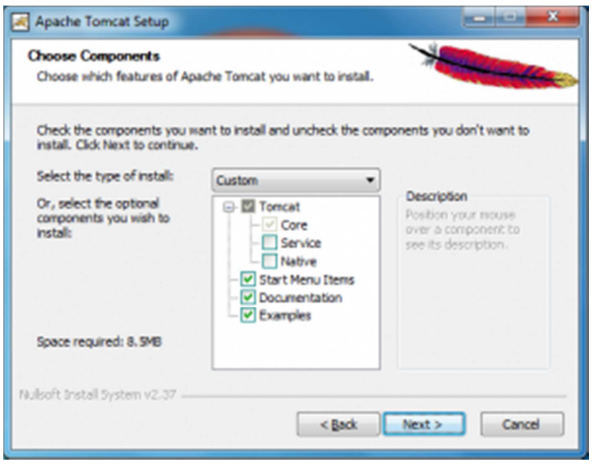
Una vez descargado el Tomcat en nuestra computadora, es hora de proceder con la estación abriendo el archivo descargado.



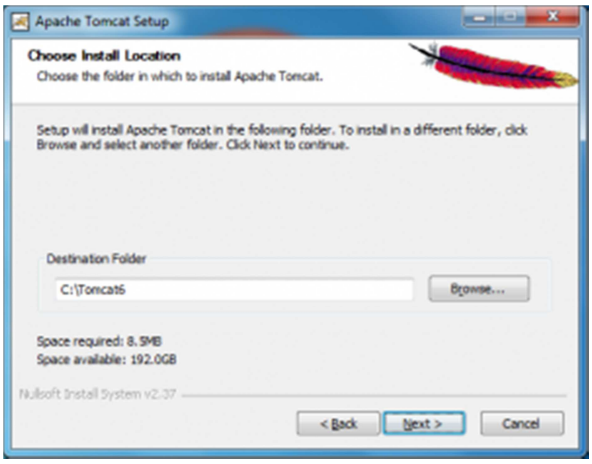
Nos saldrá los términos y condiciones, a las que después de leer podremos continuar con la instalación.



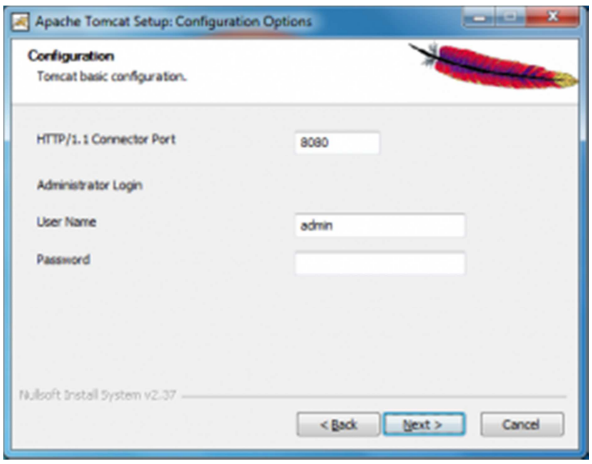
Nos pregunta, que se instalar todo, por default NO se incluye la opción de SERVICE (que solo debemos de activar si queremos que arranque el Tomcat junto con Windows), tampoco incluye la opción de NATIVE (que instala el TOMCAT por medio de DLLs, que supuestamente brindan al TOMCAT un mejor desempeño, si estamos en desarrollo, pues esto no interesa tanto, pero si estamos en un ambiente de producción, podría ser una opción que queramos activar.

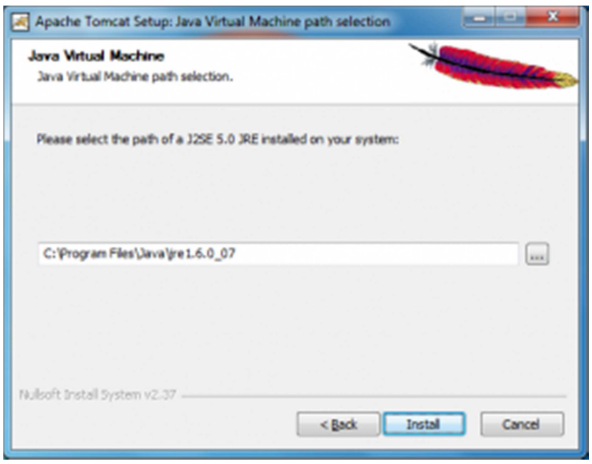


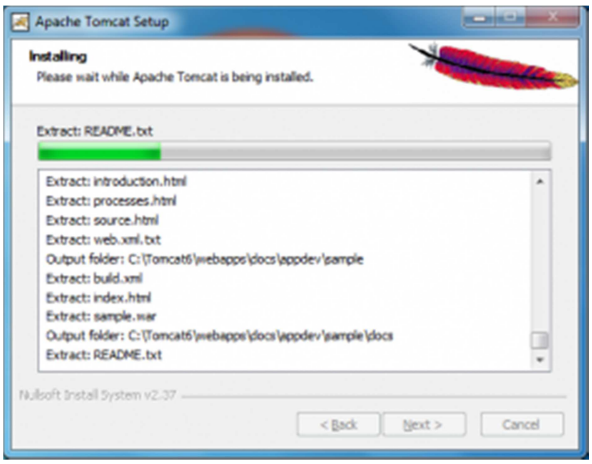
Creamos una carpeta en C:\ para seleccionarle con el instalador. La razón por la que crear una carpeta en vez de utilizar la ruta por defecto, es porque la ruta por defecto nos solicita permisos de administrador cada vez que la maquina se reinicie, sería mucho mejor disponer de una ruta donde iniciar automáticamente.

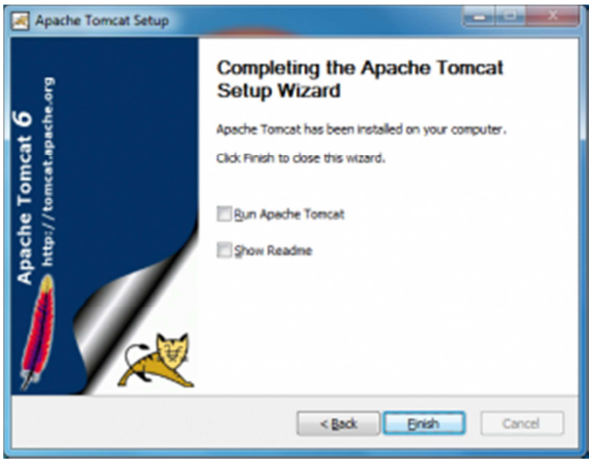


Escogemos el puerto, típicamente 80 o 8080, podemos dejar el username y password vacios.

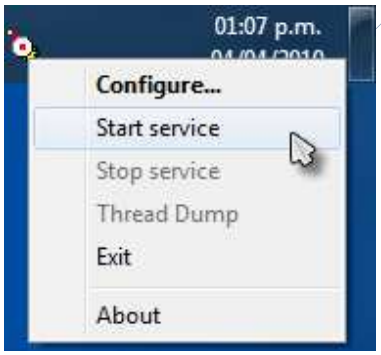



Apache Tomcat Setup: Java Virtual Machine path selection
Java Virtual Machine
Java Virtual Machine path selection.
Please select the path of a J2SE 5.0 JRE installed on your system:
C:\Program Files\Java\jre1.6.0_07
Nullsoft Install System v2.37
< Back
Install
Cancel

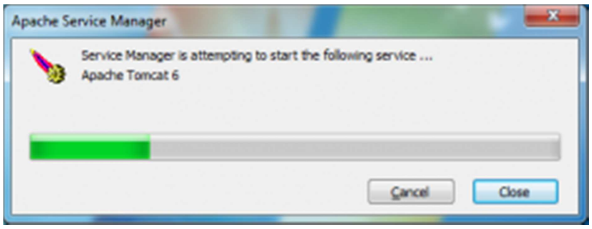

Apache Tomcat Setup
Installing
Please wait while Apache Tomcat is being installed.
Extract: README.txt
Extract: introduction.html
Extract: processes.html
Extract: source.html
Extract: web.xml.txt
Output folder: C:\Tomcat6\webapps\docs\appdev\sample
Extract: build.xml
Extract: index.html
Extract: sample.war
Output folder: C:\Tomcat6\webapps\docs\appdev\sample\docs
Extract: README.txt
Nullsoft Install System v2.37
< Back
Next >
Cancel

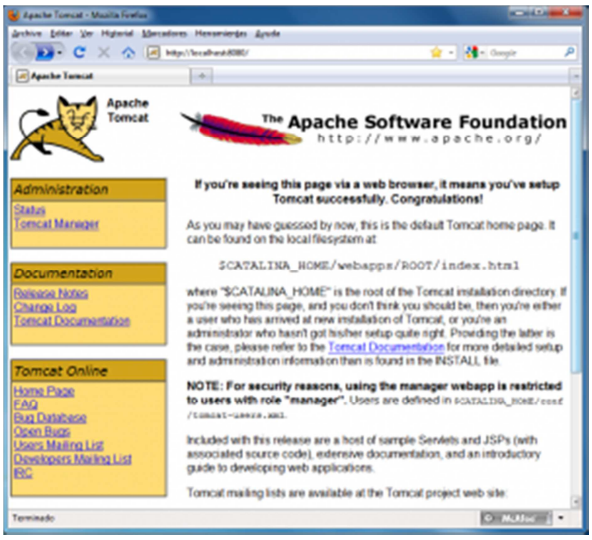

Apache Tomcat Setup
Apache Tomcat 6
http://tomcat.apache.org
Completing the Apache Tomcat Setup Wizard
Apache Tomcat has been installed on your computer.
Click Finish to close this wizard.
Run Apache Tomcat
Show Readme
< Back
Finish
Cancel

Una vez terminada la instalación del Tomcat, es necesario levantar el servicio. Para ello usted deberá darle click derecho sobre el icono del Tomcat como se ve en la siguiente imagen.



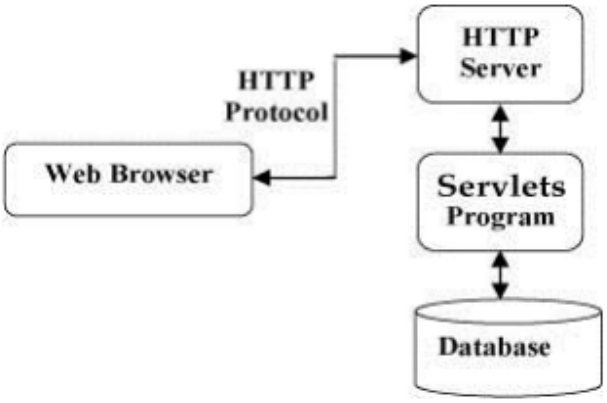
Va durar algunos pocos segundos arrancando, una vez que se ha iniciado y el icono pase a color verdad, podemos ingresar al servidor local. Para ello abra su navegador favorito y escriba: localhost:8080, o localhost:80 dependiendo de cuál puerto abrió para la instalación.





Capítulo 6
Introducción a los Servlets

Cuando se vio el patrón MVC, se ha resaltado que en la programación web, no es posible crear una clase controladora tan formal como si es el caso de la programación de aplicaciones locales. Los servlets serán análogamente equivalentes al controlador de las aplicaciones, la vista serán las páginas JSP; dicho esto procedemos a ampliar el tema de Servlets.



Definición de Servlets

Un servlet es una clase de Java que tiene las siguientes características.

1. No importa el servidor que utiliza para ejecutar el Servlet, siempre se va tener el mismo comportamiento sin importar si se usa Tomcat, Glassfish, o cualquier otro servidor. También es independiente del sistema operativo.
2. Los servlets pueden llamar a otros servlets, esto permite invocar por ejemplo servlets especializados para la comunicación de la base de datos.
3. Los servlets pueden obtener información en concreto de la maquina cliente, por ejemplo dirección IP, versión del navegador, entre muchos demás detalles mediante el protocolo HTTP.
4. Los servlets permiten el uso de Cookies y Sessiones, para tener paginas personalizadas del cliente. Un ejemplo de ello puede ser la lista de un carrito de compras, el cual puede estar en memoria hasta que el usuario del sistema decida comprarlos o eliminar la compra.
5. Los servlets pueden servir de comunicación con la base de datos, permitiéndole funcionar como un controlador.
6. Los servlets permiten generar código HTML. Es así como se puede usar un servlets para crear una página de forma dinámica. Sin embargo, esta forma de trabajar resulta en código excesivo por lo que la mayoría de programadores prefieren no utilizar esta funcionalidad.

Métodos principales de un Servlet

Existen varios métodos que se van hablar más adelante, por el momento citaremos a dos métodos impostantes:

processRequest() permite ejecutar el cuerpo del Servlet, los métodos get y post invocan este método.

getServletInfo() retorna un string con el detalle que hace el servlet.

Uso del método GET

Existen dos formas de comunicarse con un servlet, la forma más simple es mediante una solicitud por URL, en este caso cualquiera podría solicitarle información al servlet; esto es por ejemplo cuando tenemos un sitio web y queremos que cambie de apariencia o comportamiento según el URL.

El método GET se suele asociar a aquellos eventos donde el usuario solicita información, sin embargo si tomamos esto como regla, podríamos estar creando huecos de seguridad al hacer servlets GETs públicos, suponga que hacemos un servlet para obtener el salario de una persona, si hacemos eso lo más probable es que nuestro sitio este dando vacíos de seguridad al dejar que cualquier persona que conozca la cedula de un cliente pueda ver sus fondos económicos.

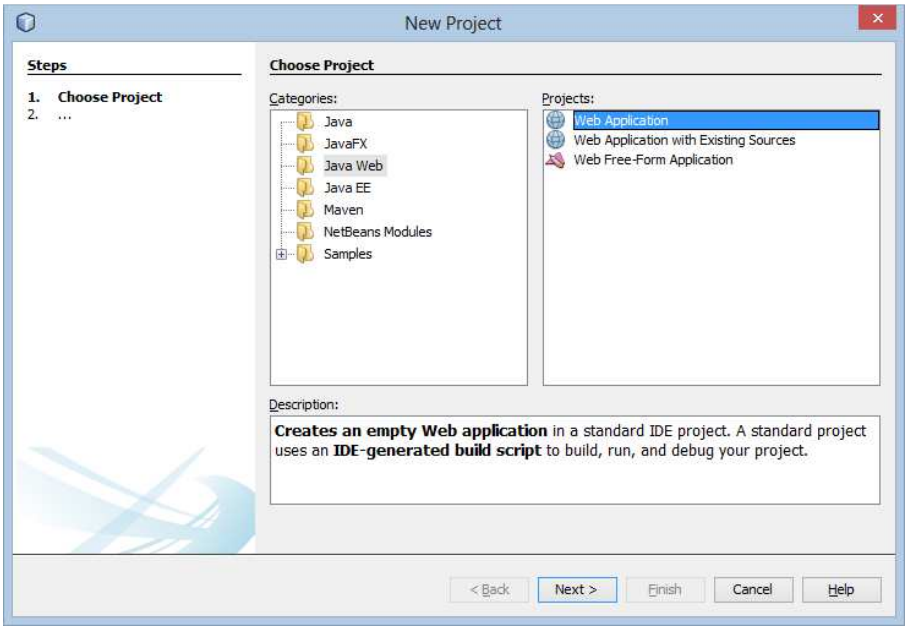
Ejemplo de un método GET

La idea es programar una página donde construya de forma dinámica las tablas de multiplicar. Para ello se va desarrollar un Servlet que cree una etiqueta HTML de tipo Table con la tabla de multiplicar que es recibida por parámetros.

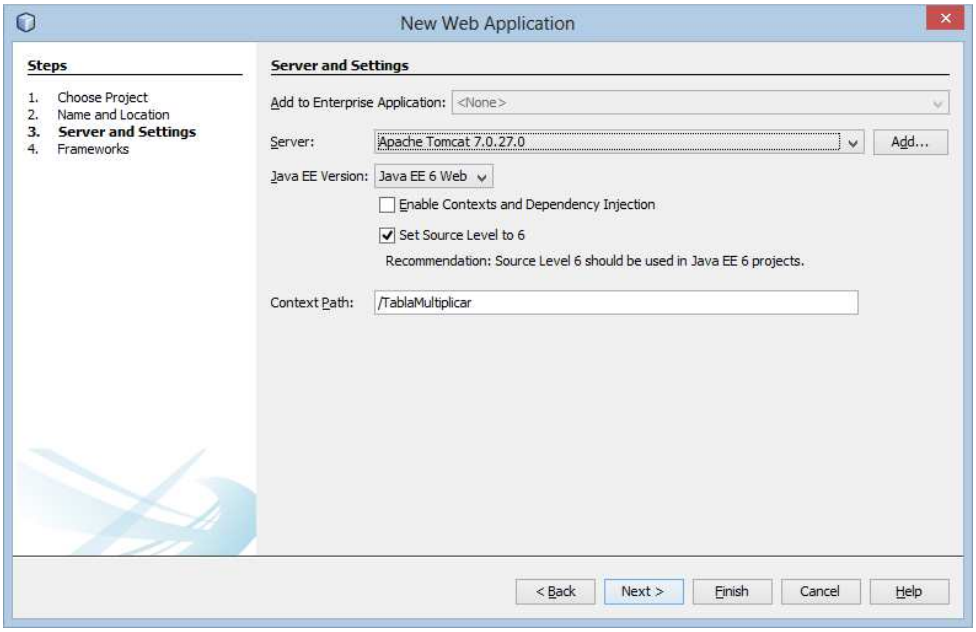
La idea es tener diferentes HTML creados a partir de un único Servlet.

```
http//matemática/Multuplicar?n=2
http//matemática/Multuplicar?n=4
http//matemática/Multuplicar?n=7
```

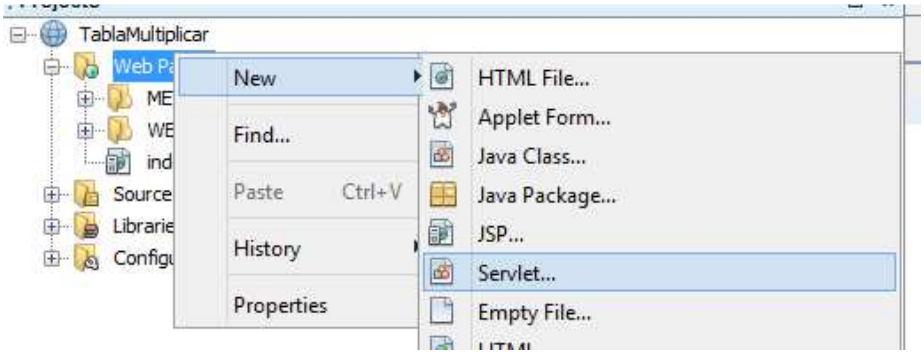
Ingresamos a NetBeans y creamos un nuevo proyecto de tipo Web Application, le ponemos como nombre al proyecto “TablaMultiplicar” y finalizamos la creación.



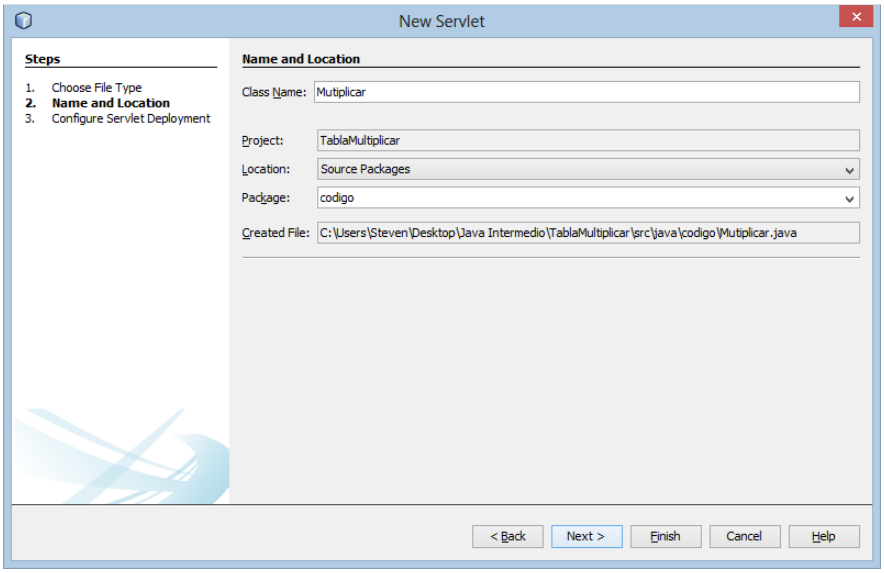
En la siguiente ventana, nos solicitará escoger el Servidor. Se desplegaran aquellos servidores que estén instalados en la maquina local. Terminamos de crear el proyecto con Finish.



Una vez creado el proyecto, nos vamos a ir a la izquierda donde se encuentran los proyectos, con un click derecho le damos agregar y luego nuevo Servlet.



En la ventana creada le damos a Class Name el nombre del Servlet, escojamos para esta práctica el nombre de "Multiplicar". Es altamente recomendado separar la ubicación de los Servlets en packages, para este ejemplo le hemos puesto "código".



Esto ha de crear el siguiente código:

```
package codigo;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet(name = "Mutiplicar", urlPatterns = {"/Mutiplicar"})
public class Mutiplicar extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here. You may use following sample code. */
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Mutiplicar</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet Mutiplicar at " + request.getContextPath() + "</h1>");
```

```
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Modificamos el cuerpo del método ProcessRequest con la siguiente implementación.

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
         response.setContentType("text/html;charset=UTF-8");
         PrintWriter out = response.getWriter();
         try {
             int numero = Integer.parseInt(request.getParameter("n"));
             out.println("<html>");
             out.println("<head>");
             out.println("<title>Ejemplo de multiplicar</title>");
             out.println("</head>");
             out.println("<body>");
             out.println("<table border =1>");
             for(int i = 1; i <= 10; i++){
                 out.println("<tr>");
                 out.println("<td>");
                 out.println(i + " x " + numero + " =");
                 out.println("</td>");
                 out.println("<td>");
                 out.println(Integer.toString(i * numero));
                 out.println("</td>");
                 out.println("</tr>");
             }
             out.println("</table>");
             out.println("</body>");
             out.println("</html>");
         } finally {
             out.close();
         }
     }
```

Lo primero que debemos rescatar es la siguiente línea:

```
int numero = Integer.parseInt(request.getParameter("n"));
```

Lo que hace es leer parámetros de entrada, como puede ver la URL que se ha introducido, se ve que el parámetro n=4.

```
http//matemática/Multuplicar?n=4
```

Para poder pasarle multiples parámetros, es necesario aplicar una concatenación como se observa en la siguiente URL:

```
http//matemática/Multuplicar?n=4&idioma=ingles&precio=78.5
```

Las siguientes líneas permiten crear la tabla creada por la construcción del HTML.

```
out.println("<table border =1>");
for(int i = 1; i <= 10; i++){
    out.println("<tr>");
    out.println("<td>");
    out.println(i + " x " + numero + " =");
    out.println("</td>");
    out.println("<td>");
    out.println(Integer.toString(i * numero));
    out.println("</td>");
    out.println("</tr>");
}
out.println("</table>");
```

La ejecución del URL con diferentes parámetros permite que se crean cualquier tabla de multiplicar, para el ejemplo siguiente se ha utilizado n = 4, n = 6 y n = 13.

1 x 4 =	4
2 x 4 =	8
3 x 4 =	12
4 x 4 =	16
5 x 4 =	20
6 x 4 =	24
7 x 4 =	28
8 x 4 =	32
9 x 4 =	36
10 x 4 =	40

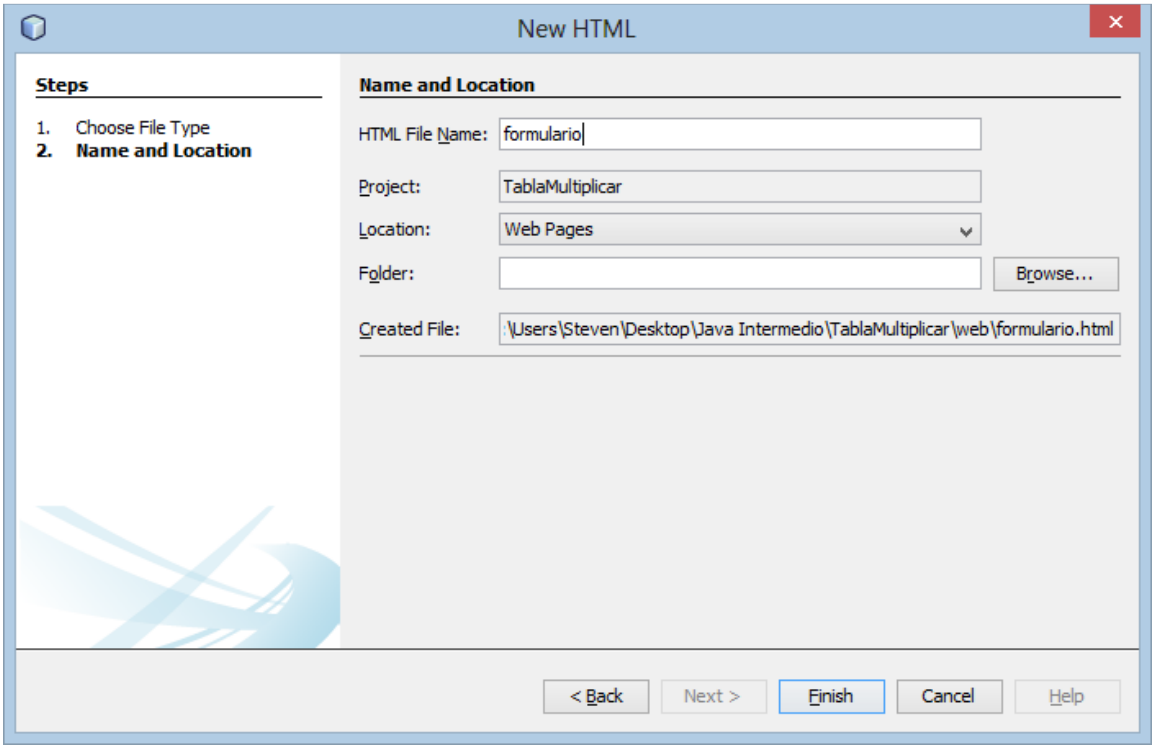
1 x 6 =	6
2 x 6 =	12
3 x 6 =	18
4 x 6 =	24
5 x 6 =	30
6 x 6 =	36
7 x 6 =	42
8 x 6 =	48
9 x 6 =	54
10 x 6 =	60

1 x 13 =	13
2 x 13 =	26
3 x 13 =	39
4 x 13 =	52
5 x 13 =	65
6 x 13 =	78
7 x 13 =	91
8 x 13 =	104
9 x 13 =	117
10 x 13 =	130

Uso del método POST

Ahora que sabemos la existencia de métodos para crear paginas dinámicamente, vamos a desarrollar una aplicación que permita almacenar un formulario de registro. Supongamos que se nos ha contratado para crear una aplicación en Facebook para guardar los siguientes datos: nombre, cedula, teléfono.

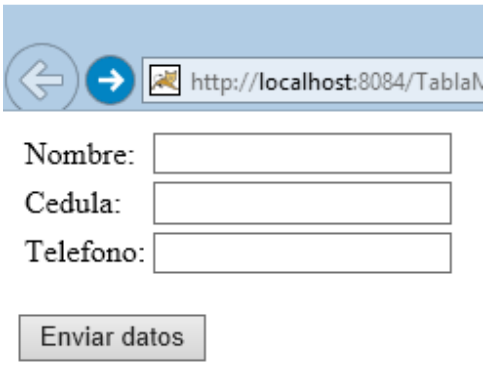
Está claro que el método GET no nos ayuda en esta aplicación, esto al no existir forma de enviarle o pasarle los parámetros en una página HTML. Creamos entonces una página HTML donde iremos a crear nuestros controles.



Complete el HTML con el siguiente código:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <form method="post" action="RegistrarFormulario">
            <table>
                <tr>
                    <td>Nombre:</td>
                    <td><input type="text" name="CampNombre" value="" /></td>
                </tr>
                <tr>
                    <td>Cedula: </td>
                    <td><input type="text" name="CampCedula" value="" /></td>
                </tr>
                <tr>
                    <td>Telefono: </td>
                    <td><input type="text" name="CampTelefono" value="" /></td>
                </tr>
            </table>
            <br/>
            <input type="submit" value="Enviar datos" />
        </form>
    </body>
</html>
```

La aplicación deberá verse de la siguiente forma:



Observe el parámetro action de la página escrita en el código anterior:

```
<form method="post" action="RegistrarFormulario">
```

El nombre que se ponga es muy importante, ahora debemos crear un Servlet con el mismo nombre “RegistrarFormulario”.

```
@WebServlet(name = "RegistrarFormulario", urlPatterns =
{"/RegistrarFormulario"})
public class RegistrarFormulario extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
            String nombre   = request.getParameter("CampNombre");
            String cedula   = request.getParameter("CampCedula");
            String telefono = request.getParameter("CampTelefono");
            guardarRegistro(nombre, cedula, telefono);
            response.sendRedirect("formulario.html");
      } finally {
            out.close();
      }
}    // ..
}
```

De estas líneas, lo nuevo es la forma en como nos desplazamos entre las páginas. La instrucción `response.sendRedirect("formulario.html");` permite una vez ejecutado el servlet, realizar una redirección a la URL que este dentro del método anterior.

Para hacer persistente el cambio se invoca el método guardarRegistro, en el siguiente tema se implementará dicho método.

Servlets y JDBC

Todo Servlet puede ejecutar instrucciones Java, y por ende también puede ejecutar instrucciones de bases de datos. El método que faltaba explicar anteriormente se implementa en las siguientes líneas:

```
boolean guardarRegistro (String r1, String r2, String r3){
  int modificado = 0;
  try {
    Class.forName("jdbc.odbc.JdbcOdbcDriver");
    bd = DriverManager.getConnection(url, usuario, clave);
     String sql = "INSERT INTO personas VALUES(“ + r1 + “,”+r2 + “,”+ r3 + “)";
    Statement comando = bd.createStatement();
    modificado = comando.executeUpdate(sql);
    comando.close();
    bd.close();
  } catch (Exception ex) {
  }
  finally {
    return modificado == 1;
  }
}
```

Este método de insertar en la Base de datos deberá quedar de la siguiente forma:

```
 */
@WebServlet(name = "RegistrarFormulario", urlPatterns = {"/RegistrarFormulario"})
public class RegistrarFormulario extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {...}

    boolean guardarRegistro(String r1, String r2, String r3) {...}

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Manejo del Objeto Sesión

En muchos casos es necesario tener información almacenada en los servidores, pero no en las bases de datos. Podemos plantear muchos ejemplos pero los principales usos son:

1. Realizar validaciones de seguridad, la idea es que en una sesión se almacene la información de la persona que se ha logeado.
2. Datos temporales, como un carrito de comprar. El usuario finalmente puede decidir no aplicar los cargos.

Una sesión es básicamente un registro (sea un dato primitivo o compuesto) que se almacena en el servidor y que lleva registro de la conexión con el cliente. Esto significa que un cliente web puede tener sus datos en un servidor, garantizando cierto nivel de aislamiento.

Métodos de la Interface HttpSession

Es importante explicar los métodos principales que tiene implementado la clase HttpSession, esta es la clase encargada de administrar las sesiones de los clientes en el servidor.

getAttribute(), getAttributeNames(), setAttribute(), removeAttribute()

Estos métodos son usados para establecer, recuperar, remover objetos desde la sesión del usuario. Veremos en ejemplos posteriores como utilizarlos.

getId()

Cada sesión que se crea en el servidor, tiene un único identificador asociado al cliente. Por lo que es imposible que una persona ingrese a los datos de otra persona. Este método retorna el id de la sesión creada.

getCreationTime()

Este método retorna un long con el valor que índica la fecha y hora en la que fue creada la sesión, este método es útil para saber por ejemplo cuando fue la primera vez que se conectó, podríamos limitar el tiempo de estadía en la aplicación como por ejemplo en las instituciones bancarias.

getLastAccessedTime()

Retorna un long con la fecha de la última vez que el usuario accedió a algún recurso del servidor, podría ser utilizada para expulsar a usuarios que estén inactivos en cierto tiempo.

getMaxInactiveInterval(), setMaxInactiveInterval()

Recupera y establece el valor en segundos del tiempo máximo en el que se invalida una sesión. Cuando una sesión se invalida, los datos contenidos en el servidor se destruyen, es así como modificando el valor del setMaxInactiveInterval por un valor, digamos 10 minutos (10*60 segundos) lograremos que si un usuario no accede al servidor, será expulsado automáticamente.

isNew()

Este método retorna true cuando el usuario visita por primera vez algún contenido del servidor. Si se cuenta con 3 paginas, y accedemos a la primera entonces este método retornara false para las restantes 2 paginas.

invalidate()

Este método invalida la sesión, típicamente se utiliza este método para hacer logout, con lo que se termina la sesión y se borran todos los elementos almacenados. Si el mismo usuario ingresa al sistema, se le creará una nueva sesión para reemplazar a la anterior.

Código de ejemplo de Sesiones

Utilizaremos el ejemplo del carrito de compras, para ello es necesario especificar las clases Productos y Carrito. La clase Producto nos va interesar almacenar el nombre, precio y dirección URL del mismo.

Mientras la clase Carrito lo que tiene, es una lista de productos. Además, el carrito nos interesa saber el total a pagar de todos los productos.

```
public class Producto {
    private String nombre;
    private double precio;
    private String direccion;

    public Producto(String nombre, double precio, String direccion) {
        this.nombre = nombre;
        this.precio = precio;
        this.direccion = direccion;
    }

    public String getNombre() {
        return nombre;
    }

    public double getPrecio() {
        return precio;
    }

    public String getDireccion() {
```

```
        return direccion;
    }
}

public  class Carrito {

    private List<Producto> productos;

    public Carrito() {
        productos = new ArrayList<Producto>();
    }

    public void agregarProducto(Producto prod) {
        productos.add(prod);
    }

    public double precio() {
        double suma = 0;
        for(Producto item : productos){
            suma+= item.getPrecio();
        }
        return suma;
    }
}
```

Una vez desarrollado estas dos clases, se procede a crear los Servlets que administraran en memoria la creación de productos del usuario. Se debe crear un sevlet que se llame AgregarProducto que va tener la siguiente implementación:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8");
  PrintWriter out = response.getWriter();
  try {
     String nombre = request.getParameter("campoNombre");
     double precio = Double.parseDouble(request.getParameter("campoPrecio"));
     String direccion = request.getParameter("campoDireccion");
     Producto nuevoProducto = new Producto(nombre, precio, direccion);
     Carrito carritoCliente = (Carrito)request.getSession().getAttribute("carrito_session");
     if(carritoCliente == null){
       carritoCliente = new Carrito();
     }
     carritoCliente.agregarProducto(nuevoProducto);
     request.getSession().setAttribute("nombre_atributo", carritoCliente);
  } finally {
     out.close();
  }
}
```

La clase HttpRequest tiene un método llamado “getSession()” el cual recupera la sesión del navegador cliente, esto permite que los datos entre clientes no se mezclen por accidente. Este método retorna la interface HttpSession, la cual a su vez tiene los métodos que se han mencionado anteriormente.

La clase HttpSession tiene particularmente un método importante llamado getAtribute, el cual permite recuperar un objeto cualquiera, cada objeto tiene asociado una etiqueta que lo identifica; de esta forma la instrucción

request.getSession().getAttribute("carrito_session");

Recupera el objeto carrito, ubicado con la etiqueta "carrito_session", nótese que una sesión puede almacenar cualquier elemento (objeto) de java. Como se almacena como un Object (super clase de Java) entonces su recuperación se debe realizar mediante el siguiente casting:

Carrito carritoCliente = (Carrito)request.getSession().getAttribute("carrito_session");

Por otro lado, si se quiere establecer un objeto nuevo a una sesión, es necesario primero que todo tener el objeto instanciado, para invocar al siguiente método:

request.getSession().setAttribute("nombre_atributo", carritoCliente);

Esta función permite manejar la sesión como si fuera una tabla de dispersión, cuya llave es el campo "nombre_atributo".

También se podrían ejecutar cualquiera de los siguientes métodos:

```
request.getSession().getId();
request.getSession().getCreationTime();
request.getSession().getLastAccessedTime();
request.getSession().setMaxInactiveInterval(600);
request.getSession().isNew();
request.getSession().invalidate();
```

Capítulo 7
Java Server Pages (JSP)

El último capítulo de este libro, termina con Java Server Pages, la idea de estas páginas es crear contenido dinámico mediante código Java.

Definición de JSP

Java Server Pages (JSP) es una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente, la generación dinámica se logra mediante código Java.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Titulo estatico</TITLE></HEAD>
<BODY>
<SMALL>Welcome,
<%
  for(int i= 0; i < 10; i++) {
   out.println(i);
  }
%>
</BODY></HTML>
```

¿Cuáles son las Ventajas de JSP?

Como todo en Java, la portabilidad es importante, a tal punto que JSP es igualmente independiente del Sistema Operativo y Servidor de producción.

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

En el caso de JavaScript, las ventajas son más claras; JavaScript (JS) es interpretado completamente del lado del cliente, esto significa que nunca va poder obtener el precio de un producto para crear una página con los datos recientes.

Arquitectura de una aplicación

El código fuente de una página JSP incluye:

1. Directivas: Dan información global de la página, por ejemplo, importación de clases.
2. Declaraciones: Sirven para declarar métodos y variables.
3. Scripts de JSP: Es el código Java embebido en la página.

4. Expresiones de JSP: Formatea las expresiones como cadenas para incluirlas en la página de salida.

Directivas

Una directiva de JSP es una sentencia que proporciona la información que debe ser cargada. Existen muchas posibles directivas:

Page: Información para la página.
Include: Incluye archivos completos palabra por palabra.
Taglib: La dirección de la librería de tags que se usará en la página.

Declaraciones

Una declaración de JSP, puede definirse como una definición de variables y métodos a nivel de clase que son usadas en la página. Un bloque de declaraciones típico sería <%! declaración %> Un ejemplo de declaración de script sería el siguiente:

```
<HTML>
<HEAD>
<TITLE>Página simple JSP</TITLE>
</HEAD>
<BODY>
      <%!
          String strCadena = "x";

          int intContador = 0;
      %>
</BODY>
</HTML>
```

El siguiente código permite redirigir a la ventana error.jsp en el caso de que suceda algún fallo durante la carga de la aplicación, o errores no controlados.

```
<%@ page language='java' contentType='text/html'
isErrorPage='false' errorPage='error.jsp' %>
```

Esta línea siguiente incorpora elementos de clases Java, como cuando se realizan los imports en código Java, en JSP es posible incluirlos en el documento.

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

Scripts de JSP

Los Scripts son bloques de código Java residentes entre los tags <% y %>. Estos bloques de código estarán dentro del servlets generado. También hay algunos objetos implícitos disponibles para los Scripts desde entorno del Servlet. Vamos a verlos a continuación.

Objetos implícitos	**Descripción**

request	Es la petición del cliente. Es una subclase de la case HttpServletRequest.
response	Es la página JSP de respuesta y es una subclase de HttpServletResponse.
session	El objeto de sesión HTTP asociado a la petición.
application	Lo que devuelve el servlet cuando se llama a getServletConfig().getContext()
out	El objeto que representa la salida de texto por pantalla.
config page	El objeto ServletConfig de la página. Es la forma que tiene la página para referirse a si misma. Se usa como alternativa al objeto this
exception	Es una subclase libre de Throwable que es pasada a la página que majena los errores.

El siguiente fragmento de código muestra cómo obtener el valor de una parámetro mediante el objeto request, y como pasarlo a una cadena para mostrarlo en pantalla.

```
<%
  String strNombre = request.getParameter("nombre");
  out.println(strNombre);
%>
```

Los siguientes códigos son equivalentes:

```
<%= user.getName() %>
<% out.println(user.getName()); %>
```

Expresiones de JSP

Las expresiones son una magnifica herramienta para insertar código embebido dentro de la página HTML. Cualquier cosa que este entre los tags <%= y %> será evaluado, convertido a cadena y posteriormente mostrado en pantalla. La conversión desde el tipo inicial a String es manejada automáticamente.

```
<% for (int i=0;i<5;i++) { %>
     <BR>El valor del contador es <%=i%>
<% } %>
```

Componentes Java Beans

Cuando analizamos el desarrollo de la arquitectura envuelve JavaSerPages, es una buena idea intentar poner toda la lógica de negocio en componentes reutilizables. Estos componentes pueden ser insertados dentro de una página JSP cuando sean requeridos. Con esto se permite reutilizar código en clases independientes del JSP, lo que al final también colabora con el orden de la aplicación.

El lenguaje Java implementa la idea de componentes con los llamados JavaBeans. Un JavaBean es una clase de Java que se adapta a los siguientes criterios:

1. Tiene un constructor publico
2. El constructor no debe tener argumentos, para realizar una instanciación automática.
3. Cada atributo de la clase tiene sus respectivos métodos Set y Gets implementados, para poder variar y obtener datos del componente.
4. Implementa su comportamiento, de la interface Serializables.

Las propiedades son siempre colocadas y recuperadas utilizando una convención denominada común. Para cada propiedad, deben existir dos métodos, uno getXXX() y otro setXXX() dónde xxx es el nombre de la propiedad.

Si queremos almacenar el nombre de una persona sus métodos respectivos van ser: getNombre, y setNombre.

Como se puede deducir, una clase JavaBeans es una clase normal que cumple las definiciones anteriores, la idea es desarrollar este componente fuera de las paginas JSP.
Casi en todos los casos los Beans son usados para encapsular elementos de la GUI visuales y no visuales. Los JavaBeans no tienen sitio en JSP, al menos no si los tenemos en cuenta con el objetivo para lo que fueron diseñados. Si pensamos en ellos como componentes, como simple encapsulación de código Java, entonces su propósito está más claro. Los Beans hacen que nuestras páginas no estén aisladas.

Incluir un Java Bean

La siguiente clase representa un JavaBean

```
public class UserBean implements java.io.Serializable
{
  private String nombre;
  private boolean registrado;

  public String getNombre() {
    return nombre;
  }
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  public boolean getRegistrado() {
    return registrado;
  }
  public void setRegistrado(boolean registrado) {
    this. registrado = registrado;
  }
}
```

JSP proporciona instrucciones especiales para trabajar más cómodamente con Java Beans. Para ello se utiliza la etiqueta `jsp:useBean`. Si el bean aún no existe en el contexto se declara, crea e inicializa el bean. Crea una referencia al bean con el nombre dado por `id`.

Si el bean ya existe en el contexto: Obtiene una referencia al mismo con el nombre dado por `id`.

```
<jsp:useBean id='usuarioID' class='sistema.UserBean’  scope='session'>
  <jsp:setProperty name='usuarioID' property='nombre' value='Gabriela' />
</jsp:useBean>
```

Establecer propiedades al Bean

Permite establecer el valor de una propiedad de un bean. Para ello convierte, si es necesario, el valor de la propiedad desde una cadena de texto al tipo de datos correspondiente (parse implícito). Proporciona un atajo para establecer valores de propiedades a partir de los parámetros de la petición, si ambos tienen el mismo nombre.

```
<jsp:setProperty name='usuarioID' property='nombre'
value='<%= request.getParameter("nombreSesion") %>' />

<jsp:setProperty name='usuarioID' property='nombre' />
```

Recuperar propiedades del Bean

Se recupera el valor de la propiedad del Bean.

```
<jsp:getProperty name='usuarioID' property='nombre' />
```

Referencias

Code Java. (29 de 12 de 2012). Obtenido de http://www.codejava.net/: http://www.codejava.net/java-se/jdbc/jdbc-database-connection-url-for-common-databases

Code Java. (29 de 12 de 2012). Obtenido de Code Java: http://www.codejava.net/java-se/jdbc/jdbc-database-connection-url-for-common-databases

Desarrolloweb. (26 de 12 de 2012). *Desarrollo Web*. Obtenido de http://www.desarrolloweb.com/articulos/1054.php

Gamma, E., Helm, R., Johnson, R., & Vlisides, J. (1994). *Design Patterns - Elements of Reusable Object-Oriented Software*. Holland: Cordon Art.

Grupo EIDOS. (2000). Lenguaje HTML Versión 1.0.0. *www.LaLibreriaDigital.com* .

it.uc3m.es. (8 de 1 de 2013). Obtenido de http://www.it.uc3m.es/labttlat/2007-08/material/jsp/

Keogh, J. (2003). *Manual de Referencia - J2EE*. España: Mc Graw Hill.

Universidad del Valle. (5 de 1 de 2013). Obtenido de http://eisc.univalle.edu.co/materias/Programacion_Interactiva/lecturas/Lectura6.pdf

Wikipedia. (26 de 12 de 2012). Obtenido de Wikipedia: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

www.tecnun.es. (6 de 1 de 2013). Obtenido de http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/javaservlets/servlets.pdf