





JAVA

Hola!

José Julián Ariza



Fundador de **Impacto Tecnológico** desde 2006.

Líder estratégico de Equipos de Desarrollo Ágil en modalidad Online y Presencial

Arquitecto de Software por naturaleza y DevOps por hobby

@JJArizaV – josejulian@impactotecnologico.net

@impactotecno



Temario

BLOQUE I

- Java 8 Platform Overview
- Java Syntax and Class Review
- Encapsulation and Subclassing
- Overriding methods,
Polymorphism and Static
Classes
- Abstract and Nested Classes
- Interfaces and Lambda
Expressions
- Collections and Generics
- Collections streams and filters

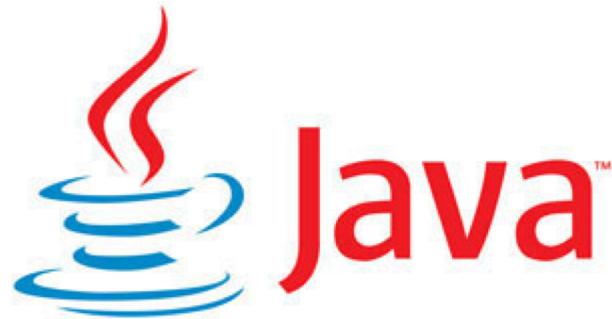
BLOQUE II

- Lambda Built-in Functional Interfaces
- Lambda Operations
- Exceptions and Assertions
- I/O Fundamentals
- Concurrency
- Database Application with JDBC

1. Visión General de JAVA 8

Hagamos una descripción general





Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principio de los años 90's.

Surgió como proyecto de la empresa como un lenguaje de programación sencillo y universal destinado a *electrodomésticos liderado por ...*



Visión General de Java 8

DATOS GENERALES

- Es la última versión del lenguaje de desarrollo
- Es el incremento de versión que añade funcionalidades nuevas al lenguaje de desarrollo
- Lanzado el 18 de Marzo de 2014

NUEVAS FUNCIONALIDADES

- Expresiones LAMBDA
- Referencias a Métodos
- Método por defecto
- Nuevas herramientas
- Stream API
- Date Time API
- Énfasis en las mejores prácticas para el manejo de valores null
- Nashorn

Características de Java

- Orientado a objetos:
 - Salvo los tipos de datos primitivos, todo es objeto en Java. Y además, Java se ha provisto de clases incorporadas que encapsulan los tipos primitivos.
- Sencillo
 - La sintaxis de Java es similar a la del lenguaje C y C++, pero evita características semánticas que les hacen complejos, confusos y no securizados
- Distribuido
 - Java implementa los protocolos de red estándares, lo que permite desarrollar aplicaciones cliente/servidor en arquitectura distribuida, con el fin de invocar tratamientos y/o recuperar datos de máquinas remotas.
- Interpretado
 - Un programa Java no lo ejecuta sino que lo interpreta la máquina virtual o JVM (Java Virtual Machine). Esto hace que sea más lento ...

```
1 // HelloWorld.java  
2 // The traditional first program for novice programmers!  
3 class HelloWorld {  
4     public static void main(String[] args) {  
5         System.out.println("Hello World!");  
6     }  
7 }
```

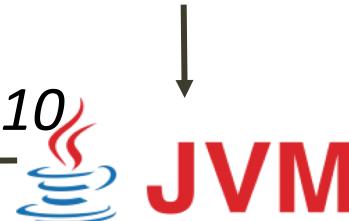


*javac
compilación*

bytecodes



010101010



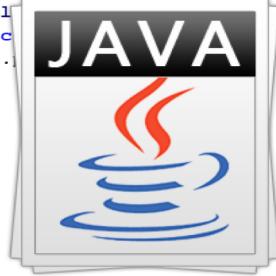
Características de Java

- ◆ Robusto
 - ◆ Java es un lenguaje fuertemente tipado y estricto. Por ejemplo, la declaración de las variables debe ser obligatoriamente explícita en Java. Se verifica el código (sintaxis, tipos) en el momento de la compilación y también de la ejecución, lo que permite reducir los errores y los problemas de incompatibilidad de versiones.
- ◆ Securizado
 - ◆ Dado los campos de aplicación de Java, es muy importante que haya un mecanismo que vigile la seguridad de las aplicaciones y los sistemas. El motor de ejecución de Java (JRE) es el encargado de esta tarea.
- ◆ Independiente de las arquitecturas
 - ◆ El compilador Java no produce un código específico para un tipo de arquitectura. De hecho, el compilador genera bytecode (lenguaje binario intermedio) que es independiente de cualquier arquitectura, de todo sistema operativo y de todo dispositivo de gestión de la interfaz gráfica de usuario (GUI).

Características de Java

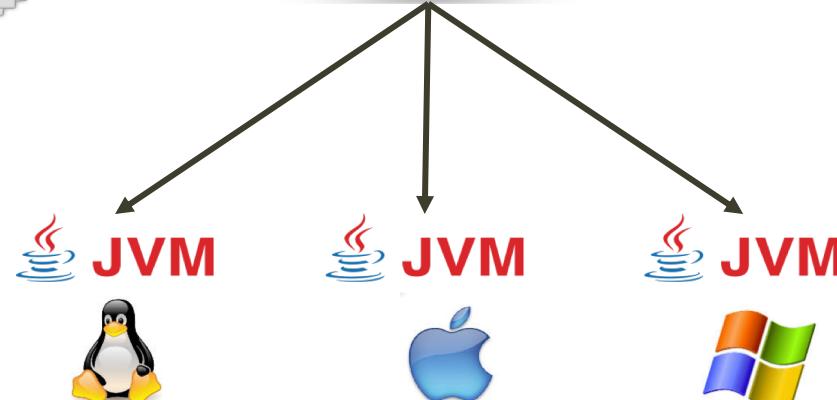
- ◆ **Portable**
 - ◆ Java es portable gracias a que se trata de un lenguaje interpretado.
- ◆ **Eficaz**
 - ◆ Incluso si un programa Java es interpretado, lo que es más lento que un programa nativo, Java pone en marcha un proceso de optimización de la interpretación del código, llamado JIT (Just In Time) o Hot Spot. Este proceso compila el bytecode Java en código nativo en tiempo de ejecución, lo que permite alcanzar el mismo rendimiento que un programa escrito en lenguaje C o C++.
- ◆ **Multihilo**
 - ◆ Java permite desarrollar aplicaciones que ponen en marcha la ejecución simultánea de varios hilos (o procesos ligeros). Esto permite efectuar simultáneamente varias tareas, con el fin de aumentar la velocidad de las aplicaciones

```
1 // HelloWorld.java  
2 // The traditional first program for novice programmers!  
3 class HelloWorld  
4     public static void main(String[] args) {  
5         System.out.println("Hello World!");  
6     }  
7 }
```



bytecode

javac



Ediciones Java

- Es el núcleo que contiene todas las herramientas necesarias para crear una aplicación java sencilla que puede ser un applet para navegador web o una aplicación desktop

Java Standard Edition



- Posee funcionalidades diseñadas para aplicaciones empresariales de mayor complejidad que incluye servicios web, XML,JMIS,RMI.

Java EE



- Proporciona un entorno robusto y flexible para las aplicaciones que se ejecutan en dispositivos embebidos y móviles en el Internet de las Cosas.

Java Embedded



- Creada para dispositivos de poca capacidad o potencia como teléfonos móviles o Smartphone, proporciona funcionalidades adicionales al core.

Java ME



Principales Diferencias con C y C++

Java	C++
Es tanto un lenguaje de programación como una plataforma de software.	Es sólo un lenguaje de programación.
Es un lenguaje puramente orientado a objetos.	Da soporte tanto a la programación estructurada como a la programación orientada a objetos.
Todas las declaraciones de variables y métodos deben estar dentro de la definición de la clase.	Las declaraciones de variables y funciones pueden estar presentes fuera de las definiciones de las clases. No es necesario para un programa en C++ tener una clase.
El lenguaje es independiente de la plataforma. El código Java, una vez escrito, puede ser ejecutado en cualquier plataforma.	El código C++, una vez escrito para una plataforma, necesita ser compilado de nuevo, y el código objeto reenlazado para ser ejecutado en otra plataforma diferente.
Maneja la memoria automáticamente.	Los programadores tienen que hacerse cargo de liberar la memoria no utilizada.

Principales Diferencias con C y C++

Java	C++
No se soporta características como sobrecarga de operadores y conversiones automáticas en ambos sentidos.	Da soporte a características como sobrecarga de operadores y conversiones automáticas en ambos sentidos.
Una clase no puede heredar directamente de más de una clase. Se da soporte a la herencia múltiple usando interfaces.	Una clase puede heredar directamente de más de una clase.
Tiene rutinas de librerías extensibles.	Sus rutinas de librerías no son extensibles.
La programación de redes es más fácil. Los objetos pueden ser accedidos a través de la red usando URLs.	La programación para redes es compleja, a menos que se usen APIs de terceros. C++, es un lenguaje, que no brinda soporte incorporado para programación de redes.
Los programadores no pueden usar apuntadores. Los apuntadores se usan internamente.	Los programadores pueden usar apuntadores.
Implementa arreglos verdaderos.	Se implementan los arreglos con aritmética de apuntadores.

Eclipse

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge y actualmente es desarrollado por la Fundación Eclipse, bajo licencia Software Libre.

Eclipse brinda soporte para varios lenguajes de programación como: c/C++, PHP, SED además de numerosos plugins para distintas tecnologías.

Eclipse está compuesto por lo siguiente:

- Plataforma principal - inicio de Eclipse, ejecución de plugins
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes

2.

Sintaxis Java y Repaso del concepto de Clases



Java Sintaxis - Descripción

- Java es un lenguaje orientado a objetos
- El estilo de orientación a objeto es una abstracción del mundo que nos permite identificar objetos reales y modelarlos dentro del lenguaje para la resolución de problemas
- Todo lo que se maneja dentro del sistema se basa en la interrelación entre objetos y la forma de comunicarse unos con otros

Práctica de clases y objetos

- ◆ Indra quiere crear un software para gestión de aerolíneas, y las aerolíneas podrán ofrecer vuelos para distintos destinos.

Salida estándar en Java

Para la escritura en la consola se utiliza el método `System.out` para el flujo de salida "estándar".

Este flujo ya está abierto y listo para aceptar datos de salida.

Por lo general este flujo corresponde a mostrar la salida por consola o en otro destino de salida especificado por el entorno o el usuario.

Para aplicaciones simples Java, una forma típica de escribir una línea de datos de salida es:

```
System.out.println( "mensaje a mostrar"); // escribe y cambia de línea  
System.out.println(); //cambia de línea, se utiliza para dejar una línea en blanco  
System.out.print ("mensaje a mostrar"); // escribe y no cambia de línea
```

Tipos de Datos

Los tipos de datos primitivos se usan para definir variables en Java el cual posee ocho tipos de datos primitivos.

- Tipo primitivo
 - Tipo entero
 - Byte
 - Int
 - Short
 - Long
 - char
 - Tipo punto flotante
 - Float
 - double
 - Boolean

Tipos de Datos

Tipos de Datos Enteros

Los tipos de datos enteros se usan para almacenar valores enteros, existen cinco tipos diferentes de tipos de datos enteros, estos son: byte, short, int, long, y char.

El rango de valores que estos tipos de datos primitivos pueden almacenar se muestra a continuación:

byte	-2 ⁷ hasta 2 ⁷ - 1
short	-2 ¹⁵ hasta 2 ¹⁵ - 1
int	-2 ³¹ hasta 2 ³¹ - 1
long	-2 ⁶³ hasta 2 ⁶³ - 1

Las diferentes variables de los tipos de datos enteros se declaran como se muestra en el siguiente código:

```
byte b = 1;  
short s = 1;  
int i = 1;  
long l = 1;
```

Tipos de Datos

Tipos de Datos de Punto Flotante

Los tipos de datos de punto flotante se usan para almacenar valores de punto flotante. Los dos tipos de tipos de datos para punto flotante son float y double.

El rango de estos tipos de datos primitivos es el siguiente:

float	-3.4 * 10 ³⁸ hasta 3.4 * 10 ³⁸
double	-1.8 * 10 ³⁰⁸ hasta 1.8 * 10 ³⁰⁸

El siguiente código muestra la declaración de dos variables pertenecientes al tipo de datos de punto flotante:

```
float a = 1.1f;  
double d = 1.1;
```

Las variables literales float deben terminar con f o F.

Tipos de Datos

Tipo de Datos boolean

El tipo de datos boolean se usa para almacenar valores booleanos, es decir, true o false. La palabra clave boolean se usa para denotar el tipo de datos booleano.

El siguiente código muestra la declaración de una variable boolean:

Tipos de Datos

- La clase String soporta alguna sintaxis no estándar.
- Se puede instanciar un objeto String sin utilizar la palabra clave new; se prefiere esto:

```
String name1 = "Juan José";
```

- Se puede utilizar la palabra clave new, pero *no* se recomienda:

```
String name1 = new String("Córdoba");
```

- Un objeto String es inmutable; su valor no se puede cambiar.
- Un objeto String se puede utilizar con el símbolo del operador de concatenación de cadenas (+) para la concatenación.
- La concatenación crea una nueva cadena y la referencia

Comparativa entre Strings

Cuando se utiliza el operador `==` para comparar referencias de objetos con objetos `String`, el operador prueba si las direcciones de las referencias de objetos `String` de la memoria son iguales, no su contenido.

Para probar la igualdad entre cadenas de caracteres es necesario el uso del método `equals` de la clase `String`. La clase del ejemplo contiene dos nombres de empleados y un método para comparar los nombres

```
public class Employees {  
    public String name1 = "Juan José";  
    public String name2 = "Cordoba";  
    public void areNamesEqual() {  
        if (name1.equals(name2)) {  
            System.out.println("Same name.");  
        } else {  
            System.out.println("Different name.");  
        }  
    }  
}
```

StringBuilder

StringBuilder proporciona una alternativa variable a String, consiste en una clase normal Java con las siguientes características:

- ❑ Tiene un amplio juego de métodos para agregar, insertar y suprimir.
- ❑ Tiene muchos métodos para devolver una referencia al objeto actual.
- ❑ Se puede crear con la capacidad inicial que mejor se adapte a las necesidades.

Sin embargo String sigue siendo necesaria debido a que:

- ❑ Su uso puede ser más seguro que un objeto inmutable.
- ❑ Una clase de la API puede necesitar una cadena.
- ❑ Tiene muchos más métodos no disponibles en StringBuilder.

Variables

Variables Final

Las constantes en Java son definidas utilizando el modificador **final** que indica que el miembro tiene un valor constante.

- Características
 - Un miembro con un valor constante no puede ser modificado, si se trata de cambiar el valor se generará un error en tiempo de compilación.
 - Deben ser inicializadas con un valor en el momento de su declaración de lo contrario arrojará un mensaje de error, diciendo que la variable puede no haber sido inicializada.
 - El modificador **static** puede ser combinado con la variable final para declarar constantes de librerías. Esta combinación es particularmente útil al declarar constantes como: PI, E, etc.

El nombre de una variable final se representa en letras mayúsculas por convención tal como se muestra a continuación: **final float PI = 3.14;**

Operadores

En Java existen expresiones que hacen uso de varios operadores y son similares a expresiones aritméticas que usan muchos operadores.

Las variables usadas en estas expresiones se llaman *operandos*, existen 3 tipos de operandos:

- Los que solo necesitan un operando, son llamados operadores *unarios* (ejemplo: `++`).
- Los que requieren dos operandos se llaman operadores *binarios* (por ejemplo, `+`).
- Los que requieren tres operandos se llaman operadores *ternarios* (ejemplo: `?:?`).

A su vez Java soporta los siguientes tipos de operadores:

- OPERADORES
 - Operadores Aritméticos.
 - Operadores Relacionales.
 - Operadores Condicionales.
 - Operadores Ternarios.
 - Operadores de Bits.
 - Operadores de Asignación.

Operadores Aritméticos

OPERADOR	USO	DESCRIPCION
+	$a + b$	Permite adicionar valores almacenados en dos o más variables
-	$a - b$	Permite restar un valor almacenado en una variable de un valor almacenado en otra variable.
*	$a * b$	Permite multiplicar valores almacenados en dos variables.
/	a / b	Permite dividir un valor almacenado en una variable entre un valor almacenado en otra variable.
%	$a \% b$	Permite encontrar el residuo resultante de dividir un valor almacenado en una variable entre el valor almacenado en otra variable.

Precedencia de Operadores

Reglas de prioridad:

1. Operadores delimitados por un par de paréntesis.
2. Operadores de aumento y disminución.
3. Operadores de multiplicación y división, evaluados de izquierda a derecha.
4. Operadores de suma y resta, evaluados de izquierda a derecha.

Ejemplo:

$$c = 25 - ((5 * 4) / 2)) - 10 + 4;$$

El resultado real de la expresión cuando se evalúa según las reglas de prioridad, indicadas por los paréntesis: 9

Creación de construcciones if e if/else

Una sentencia if, o una construcción if, ejecuta un bloque de código si una expresión es *true*.

Sintaxis:

```
if (boolean_expression) {  
    ...code_block;  
} // end of if construct  
// program continues here
```

En donde:

- *boolean_expression* es una combinación de operadores relacionales, operadores condicionales y valores cuyo resultado es un valor true o false.
- *code_block* representa las líneas de código que se ejecutan si la expresión es true.

Condiciones Switch

switch indica una sentencia switch

case indica un valor que está probando

```
switch (variable) {  
    case literal_value:  
        <code_block>  
        [break;]  
    case another_literal_value:  
        <code_block>  
        [break;]  
    [default:]  
        <code_block>  
}
```

variable es la variable cuyo valor desea probar

literal_value es cualquier valor válido que puede contener una variable

break es una palabra clave opcional que hace que el flujo de código salga inmediatamente de la sentencia switch

Declaración, instanciación e inicialización de una matriz unidimensional

- ❑ Una matriz contiene una serie de variables que puede ser cero a N.
- ❑ Las variables contenidas en la matriz no tienen nombre y son consideradas componentes de la matriz.
- ❑ Si una matriz tiene n componentes, los componentes de la matriz se referencian usando índices enteros de 0 a n - 1, inclusive.

Sintaxis:

```
tipo identificador;
```

En donde:

tipo es el tipo de dato que contendrá la matriz .

identificador es el nombre de la matriz.

Recorriendo una matriz

```
static float media (float datos[]) {  
    int i;  
    int n = datos.length;  
    float suma = 0;  
    for (i=0; i<n; i++)  
        suma = suma + datos[i];  
    return suma/n;  
}
```

Ciclos While

La sentencia while ejecuta continuamente un bloque de instrucciones mientras una condición particular, sea verdadera.

Sintaxis:

while (expression) { statement(s) }

Ejemplo:

```
class WhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        while (count < 11) {  
  
            System.out.println("Count is: " + count);  
            count++;  
        }  
    }  
}
```

Objetos



Java Sintaxis - Qué es un objeto

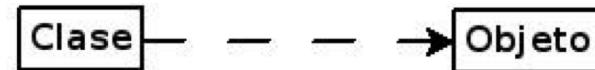
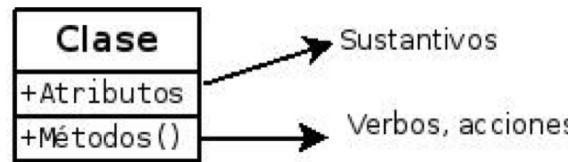
- A efectos de desarrollo de sistemas, un objeto es una entidad de programa que contiene dos características fundamentales: estado y comportamiento
- Los estados se almacenan en los campos, atributos o propiedades
- El comportamiento se define a través de los métodos. Los métodos actúan sobre los atributos internos el objeto y proveen los mecanismos de comunicación de objeto a objeto.

Java Sintaxis - ¿Y para qué necesitamos las clases?

- Los objetos son los elementos básicos para la programación empleando el paradigma OOP
- Las clases son los modelos o patrones a partir de las cuales se crean los objetos
- Los estados se pueden definir como tres tipos de posibles atributos: variables locales, variables de instancia y variables de clases
- Una clase puede definir el comportamiento a través de cualquier número de métodos que pueden acceder a cualquier tipo de valor de las variables.

Clases

- ◆ Una clase es un prototipo que se usa para definir las características y los comportamientos que son comunes a todos los objetos de un mismo tipo.



Definición
Tiempo de programación

Instancia
Tiempo de ejecución.

Clases

```
public class Persona {
```

Declaración de la
clase

```
public double altura =  
1.70
```

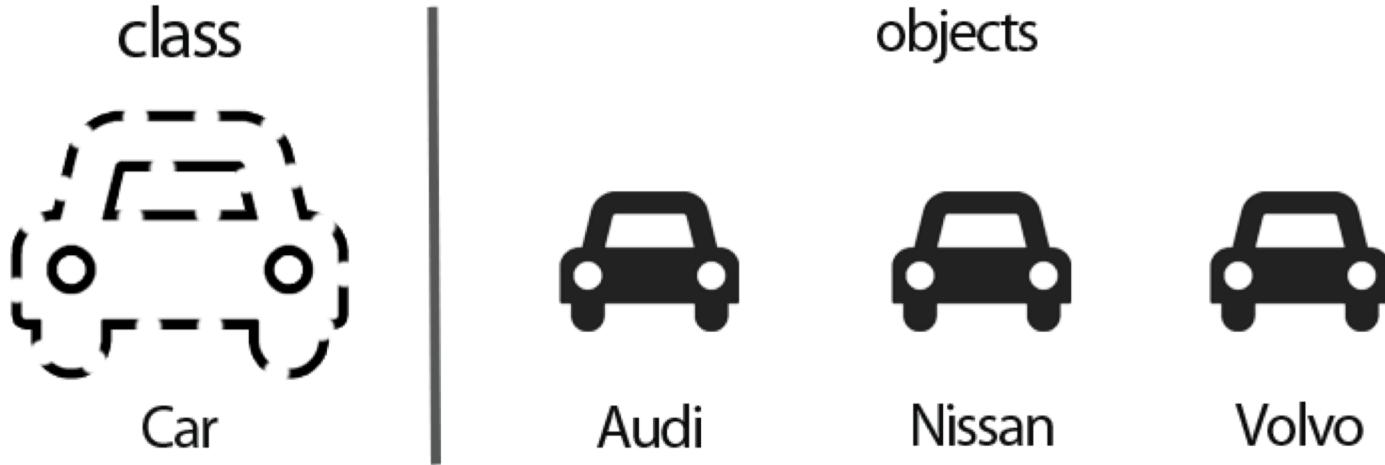
Declaración de
campo

```
public void respirar() {  
} // fin del método respirar
```

Método

Comentario

Java Sintaxis - ¿Y cómo vemos las clases?



Java Sintaxis - ¿Qué es un constructor?

- Una clase especial de método perteneciente a cada clase. Cada clase tiene al menos un constructor.
- Si no definimos un constructor, el compilador de Java construye un constructor por defecto que inicializa todas las variables a cero.
- Al momento de crear un objeto a partir de la clase al menos un constructor es invocado.
- La regla principal de un constructor es que debe tener el mismo nombre de la clase.
- Una clase puede tener más de un constructor.
- Los constructores se emplean de típicamente para inicializar variables de instancia o efectuar cualquier operación de inicio del objeto.

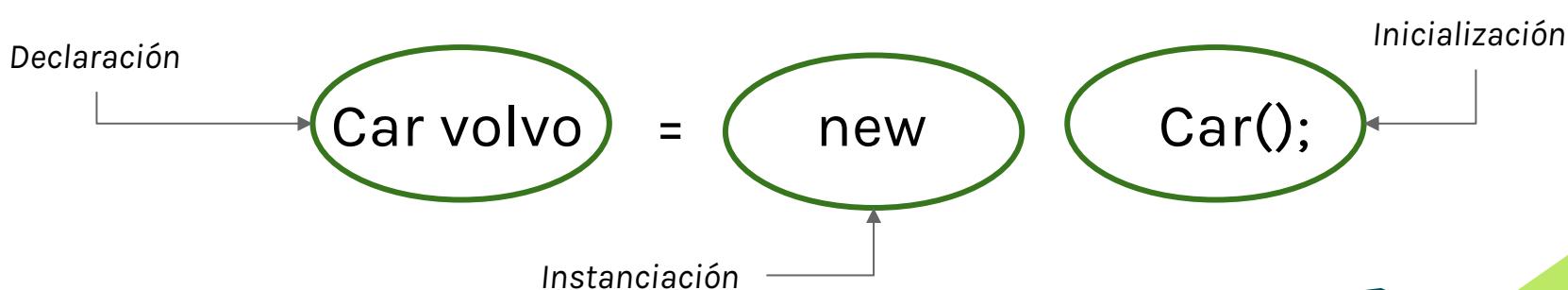
Identificación de Objetos

- ◆ Los objetos pueden ser físicos o conceptuales.
- ◆ Los objetos tienen características como nombre, tipo, tamaño, forma estas características son llamadas **atributos**.
- ◆ Los objetos pueden realizar acciones como definir un valor, visualizar una pantalla, etc. Estas acciones son llamados **métodos**.
- ◆ El valor de todos los atributos de un objeto se suele denominar **estado actual** del objeto

Java Sintaxis - ¿Cómo creamos un objeto?

Se requieren tres elementos para crear un objeto a partir de una clase:

- **Declaración:** Una declaración de una variable + el tipo de objeto que se desea crear.
- **Instanciación:** La palabra clave ‘new’ se emplea para crear el objeto.
- **Inicialización:** La palabra clave ‘new’ seguida de una llamada a un constructor realiza la inicialización del objeto de la clase seleccionada.



Reconocimiento de los criterios para definir objetos

Existen 2 criterios importantes para reconocer objetos:

1. Importancia del dominio de problemas:

- ¿Existe el objeto en los límites del dominio de problemas?
- ¿Es necesario el objeto para que se termine la solución?
- ¿Es necesario el objeto como parte de una interacción entre un usuario y el sistema?

2. Existencia independiente

Para que un elemento sea un objeto y no un atributo de otro objeto, debe existir independientemente en el contexto del dominio de problemas.

Java Sintaxis - ¿Cómo utilizamos los recursos del objeto?

Para acceder a las variable de instancia y a los métodos disponibles lo realizamos a través del objeto creado a partir de una clase:

- **Creamos un objeto:** NombreClase objeto = new ConstructorDeClase();
- **Podemos acceder a una variable de instancia de la siguiente forma:**
objeto.nombreVariableInstancia;
- **Podemos acceder a un método de la siguiente forma:**
objeto.nombreDelMetodo();

Práctica Individual

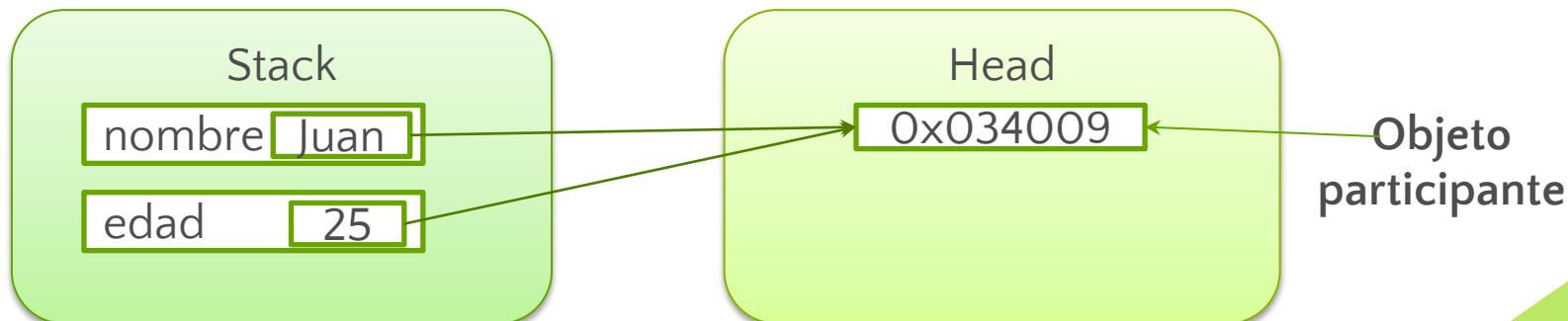
Se requiere el diseño de un aplicativo Java que permita realizar las operaciones matemáticas básicas a saber : suma, resta, multiplicación y división.

Define lo siguiente:

- Clases.
- Atributos.
- Métodos
- Comentarios

Almacenamiento de objetos en memoria

- En Java, la memoria se asigna cuando un objeto es creado a partir de una clase. Java da soporte a un manejo de memoria automático. Esto significa que la memoria asignada a un objeto está disponible para este, mientras el objeto esté siendo usado en el programa Java. Después que el objeto se usa en un programa, el recolector de basura(Garbage Collector) lo reclama y la memoria asignada a éste es liberada.



La sentencia import indica al compilador dónde están ubicadas las clases que estamos utilizando.

Para importar sólo una clase de un paquete:

```
import <nombre.paquete>.<NombreClase>;
```

Para importar todas las clases de un paquete:

```
import <nombre.paquete>.*;
```

El compilador añade a todos los ficheros la línea

```
import java.lang.*;
```

El paquete que contiene las clases fundamentales para programar en Java (System, String, Object...)

Métodos Estáticos

- ❑ Los métodos estáticos son definidos utilizando el modificador static, definiendo así un método de clase.
- ❑ Para acceder a un método de clase, no se necesita crear una instancia de la clase.
- ❑ Un método de clase no puede acceder a variables de instancia o métodos de instancia.
- ❑ Acceder a variables o métodos no estáticos desde un método estático genera un error en tiempo de compilación, aunque un método de clase puede ser accedido a través de una instancia.

Ejemplo:

```
static public void agregar(int j, int k) {  
    n = j + k;  
    System.out.println(n);  
}
```

Variables Estáticas

Una variable estática es una variable definida para la clase (compartida entre todas las instancias de una clase usando el modificador static..

Ejemplo:

```
public class Error{  
    int x ;  
    public static y ;  
  
    public static void main (String args[])  
    {  
        y = 15 ;  
        x = 20 ; //Error  
    }  
}
```

3. Encapsulamiento y Herencia (Subclassing)



Encapsulamiento y Subclases - Encapsulamiento

- Existen 4 principios fundamentales de la POO (OOP por sus siglas en inglés): **encapsulamiento, herencia (subclassing), polimorfismo y abstracción.**
- El **encapsulamiento** en Java es el mecanismo mediante el cual se empaquetan los datos (variables) y el código que actúa sobre los mismos (métodos) en una misma unidad funcional.
- Empleando este principio se colocan las variables dentro de una clase “escondidas” de las demás clases que solo podrán acceder a sus valores a través de métodos existentes en la misma clase, específicamente programados a tal fin.

Modificadores de Acceso

Se pueden usar los modificadores de acceso de Java para proteger las variables y métodos de una clase de ser accedidos sin garantía por parte de otras clases.

Los modificadores de acceso permitidos por Java son:

Modificadores de Acceso

Public: Accesibles en cualquier lugar desde donde la clase es accesible. Así mismo, son heredados por las subclases.

Protected: accesibles en las subclases y en el código del mismo paquete.

Package: Accesible en la clase y el paquete.

Modificadores de Acceso

Modificador	Clase	Subclase mismo paquete	Clase mismo Paquete	Subclase otro paquete	Clase otro paquete
private	si	no	no	no	no
protected	si	si	si	si	no
public	si	si	si	si	si
package	si	si	si	no	no

Encapsulamiento y Subclases - ¿Cómo se logra el encapsulamiento?

Siguiendo dos sencillos pasos:

1. Declarando todas las variables de una clase como privadas (private)
2. Creando dos tipos de métodos conocidos como getter and setter para cada variable, que son los únicos que podrán leer o escribir sobre dichas variables

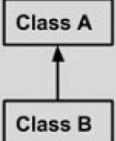
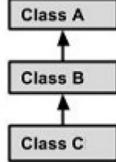
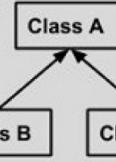
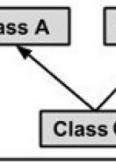
Encapsulamiento y Subclases - ¿Qué beneficios brinda el encapsulamiento?

- Para efectos de seguridad pudiéramos definir las variables de solo-lectura o solo- escritura.
- Una clase tiene el control total sobre la forma y manera en que se accede a los datos contenidos en ella.

Encapsulamiento y Subclases - Herencia o Subclassing

- La **herencia** es el proceso mediante el cual una clase adquiere (“hereda”) los atributos y métodos de otra clase. Este proceso permite el ordenamiento jerárquico de la información.
- La clase que hereda propiedades de otra clase se le llama subclase, aquella de la que se heredan las propiedades se le llama superclase.
- Palabras clave: **extends, implements, super, instanceof**
- Hay que hacer notar que Java NO SOPORTA la herencia múltiple (una clase heredando atributos de varias superclases) pero sí permite el mecanismo de implementar estructuras desde varias interfaces (lo cual le permite simular un tipo de herencia múltiple).

Encapsulamiento y Subclases - Tipos de Herencia

Single Inheritance		public class A { } public class B extends A { }
Multi Level Inheritance		public class A { } public class B extends A { } public class C extends B { }
Hierarchical Inheritance		public class A { } public class B extends A { } public class C extends A { }
Multiple Inheritance		public class A { } public class B { } public class C extends A,B { } } // Java does not support multiple inheritance

Práctica

A continuación los pasos a seguir:

-

Crear una superclase “Persona” que tenga los atributos generales de una persona y que “Pasajero” herede de persona añadiendo los atributos propios del pasajero (Ej: vuelos).

•Añadir en aerolínea métodos sobrecargados para consultarVuelos que pueda recibir:

1. solo origen
2. origen y destino

Sustitución de métodos

La sustitución de métodos ocurre cuando una subclases implementa métodos que ya tienen implantaciones en la superclase.

En este caso, las implantaciones de método en la subclase sustituyen la implantación de método de la superclase.

La sustitución de métodos se puede dar en los siguientes casos:

- Puede que los métodos que existen en la superclase:
 - No estén implantados en la subclase.
 - El método declarado en la superclase se utiliza en tiempo de ejecución.
 - Estén implantados en la subclase.
 - El método declarado en la subclase se utiliza en tiempo de ejecución.

Sobrecarga de constructores

- Java permite tener tantos constructores como se requiera para una clase, la única diferencia será en el número o el tipo de argumentos para una clase.
- Esto se denomina *sobrecarga del constructor*.
- Los constructores sobrecargados tienen un número diferente de parámetros o diferentes tipos de datos en los parámetros.
- Ésta es una forma de lograr polimorfismo en Java.

```
public Estudiante(){  
    public Estudiante(int nuevoId){  
        id = nuevoId;  
    }  
    public Estudiante(int nuevoId, String nuevoNombre){ //constructor  
sobrecargado  
        id = nuevoId;  
        nombre = nuevoNombre;  
    }  
}
```

Operador instanceof

El operador instanceof es una expresión cuyo tipo es un tipo de referencia y se utiliza para saber si la clase del objeto al que hace referencia el valor de la expresión en tiempo de ejecución es la asignación compatible con algún otro tipo de referencia.

El operador instanceof devuelve true o false basado en si el operando izquierdo es una instancia del operando derecho.

Sintaxis:

(Object reference variable) instanceof (class/interface type)

4.

Sobreescritura de Métodos, Polimorfismo y Clases Estáticas



Sobreescritura de Métodos, Polimorfismo y Clases Estáticas

- Sobreescritura de Métodos

- La sobreescritura de métodos se refiere a la posibilidad de redefinir el comportamiento de un método heredado desde una clase padre.
- Reglas:
 - ✓ Debe tener el mismo nombre, el mismo número de argumentos y tipo del valor de retorno del método original (o al menos un subtipo).
 - ✓ No puede tener un acceso más restrictivo que el original.
 - ✓ No se pueden sobreescribir tipos final o static.
 - ✓ No es obligado pero sí recomendable colocar la anotación “@override” para que el compilador identifique qué es lo que estamos haciendo y nos ayude a valorar errores.

Sobreescritura de Métodos, Polimorfismo y Clases Estáticas

- ¿Es la **sobreescritura** lo mismo que la **sobrecarga**?

- La sobrecarga de métodos se refiere a la posibilidad de generar métodos con el mismo nombre pero con una firma (juego de parámetros) distintos.
- Reglas:
 - ✓ No pueden haber dos métodos con el mismo nombre y los mismos parámetros.
 - ✓ Pueden existir n-número de métodos siempre y cuando tengan diferentes parámetros.
- Ejemplo: clase motor, método ignición. Pueden existir métodos sobrecargados para ignición dependiendo del tipo de motor: eléctrico, gasolina, diesel, etc.

Sobrecarga

```
public class DataArtist {  
    ...  
    public void draw(String s) {  
        ...  
    }  
    public void draw(int i) {  
        ...  
    }  
    public void draw(double f) {  
        ...  
    }  
}
```

Sobreescritura de Métodos, Polimorfismo y Clases Estáticas

- Polimorfismo

- El polimorfismo se refiere a la habilidad de los objetos de manifestar o “comportarse” como otros objetos. Dentro de la OOP usualmente se da cuando empleamos referencias a un objeto padre para referirnos a un objeto hijo.
- Reglas:
 - ✓ En Java una misma variable referenciada puede hacer referencia a más de un tipo de clase.
 - ✓ El conjunto de las que pueden ser referenciadas está restringido por la herencia o la implementación (si el tipo de clase hereda o implementa el tipo de la variable referenciada)

Sobreescritura de Métodos, Polimorfismo y Clases Estáticas

- Clases Estáticas

- Las clases estáticas son una forma de agrupar clases en Java.
- Java no permite crear una clase de nivel superior como estática.
- Solamente clases internas a una clase superior pueden ser definidas como estáticas.
- La creación de variables del tipo de una clase estática puede ser independiente de la creación de la clase de nivel superior que la contiene.
- Las clases internas estáticas pueden acceder a atributos estáticos de la clase que la contiene. NO podrá acceder a métodos o atributos que no sean estáticos.
- Por tanto, las clases internas NO estáticas difieren de las estáticas, en que mientras una clase estática puede instanciarse independientemente de la clase que la contiene, aquellas NO estáticas pertenecen y requieren la creación de un objeto de la clase contenedora.

Práctica

- ❑ Crear una clase Área que contenga métodos estáticos sobrecargados con el nombre area() que pueden ser usados para calcular el área de un círculo, triángulo, rectángulo y un cilindro.
- ❑ Cree una clase Usuario que utilice los métodos de Área para calcular el área de diferentes figuras geométricas y la imprime en la salida est\'andar.

Métodos de argumentos variables

- A partir de Java 5 se permite especificar métodos que tengan un número variable de argumentos de un tipo especificado.
- Estos métodos se denominan varargs y pueden ser llamados con cero o más argumentos a la posición variable de argumentos.
- Un parámetro varargs es tratado como una matriz en el elemento que define.
- Si no sabemos cuántos argumentos tendremos que pasar en el método, varargs es el mejor enfoque.

Ejemplo:

Métodos de argumentos variables

```
class VarargsEjemplo{  
  
    static void display(String... values){  
        System.out.println("invocacion del metodo display");  
    }  
  
    public static void main(String args[]){  
  
        display(); //cero argumentos  
        display("mi","nombre","es","varargs"); //cuatro argumentos  
    }  
}
```

5.

Clases abstractas y anidadas



Clases anidadas

- Una clase anidada es una clase cuya definición está dentro de otra clase.
- Si una clase contiene a una clase anidada, es conocida como *clase exterior*.
- Estas clases son consideradas miembro de la clase contenedora y pueden ser final o abstract.

Existen dos tipos de *Clases Anidadas*:

● **Clase Anidada Estática:** esta clase se declara como miembro estático (*static*) de la clase exterior y tiene todas las características de cualquier otro miembro estático.

Clase Interna: Esta clase se declara como miembro no-estático de la clase exterior y tienen todas las características de cualquier otro miembro de instancia. Una instancia de alguna clase interna sólo puede existir si una instancia de la clase contenedora existe. La clase interna no puede tener miembros estáticos propios.

Clases anidadas

- Las clases anidadas tienen acceso total a los miembros de su clase exterior, incluso a los miembros privados.
- Ahora como miembro también puede ser private, protected, además de public o package que es una posibilidad para cualquier clase.

```
class ClaseExterna{  
    //miembros de ClaseExterna  
  
    ...  
    public static class ClaseAnidadaEstatica {  
        //miembros de ClaseAnidadaEstatica  
  
        ...  
    }  
    private class ClaseInterna {  
        //miembros de ClaseInterna  
    }  
}
```

Clases abstractas y anidadas - Abstractas

- Una clase abstracta se parece en su diseño a una clase normal, pero sirve para servir de estructura padre a otras clases.
- Una clase abstracta se define con la palabra clave “abstract”.
- Características:
 - ✓ No puede ser instanciada (no se puede emplear “new”).
 - ✓ Si un método se considera o define como abstracto entonces la clase debe ser definida como abstracta.
 - ✓ Los métodos definidos como “abstract” no tienen cuerpo.
 - ✓ La clase que herede de una clase abstracta debe definir todos los métodos de la clase padre.

Clases Abstractas

- Las clases abstractas pueden tener dos tipos de métodos: métodos que tienen implementación y métodos que no tiene implementación.
- Los métodos implementados en una clase abstracta son métodos concretos.
- Los métodos que no tienen implementación son métodos abstractos.

Sintaxis:

```
abstract tipoRetorno nombreMetodo(argumentos);
```

Ejemplo:

```
abstract class U {  
    abstract void metodo1();  
    void metodo2() {  
        System.out.println("Dentro de metodo2 en la clase U");  
    }  
}
```

Comparativa

Clases Abstractas	Clases Concretas
Se usan para generalizar clases concretas.	Se usan para representar objetos del mundo real.
No pueden ser instanciadas.	Pueden ser instanciadas.
Definen métodos abstractos para los cuales la implementación no es provista.	No definen métodos abstractos.
Pueden o no proveer implementación para cualquiera de los métodos.	Deben proveer implementación para todos sus métodos.

Interfaces

- Las interfaces son similares a las clases, en términos de sintaxis, pero no soportan ninguna variable de instancia.
- Una interfaz especifica el prototipo o comportamiento de una clase.
- Todos los métodos declarados en una interfaz son métodos abstractos, es decir, no se les proporciona implementación.
- Una interfaz no puede ser instanciada.

Sintaxis:

```
modificadorDeAcceso interface nombreInterfaz {  
  
    tipoDeDatos nombreVar1 = valor;  
    tipoDeRetorno nombreMetodo1(listaDeArgumentos);  
  
}
```

Interfaces

Tanto las clases abstractas como las concretas pueden implementar una interfaz.

Una clase concreta que implemente la interfaz tiene que proveer la implementación para todos los métodos declarados en la interfaz.

La siguiente es la estructura general de la declaración de clase para la clase que implementa una interfaz:

```
modificadorDeAcceso class nombreClase [extends nombreSuperClase]  
[implements interfaz1, interfaz2, ...] {  
  
    // cuerpo de la clase
```

Manejo de Errores



Manejo de Excepciones

En java los errores son llamados excepciones y se encuentran de 2 tipos: excepciones verificadas y excepciones no verificadas.

Excepciones verificadas

- Son aquellas que el programador debe capturar y manejar dentro de la aplicación.
- Si el programador no captura una excepción verificada se producirá un error en tiempo de compilación.

Excepciones no verificadas

- Las excepciones no verificadas son excepciones del tiempo de ejecución, las cuales son detectadas por la JVM.
- Las excepciones no verificadas son lanzadas en un programa Java cuando hay un problema en la JVM.

Manejo de Excepciones

Java tiene incorporado la capacidad para asegurar que las excepciones son manejadas dentro del programa.

Para el manejo de excepciones se utilizan los siguientes recursos:

Manejo de errores

Bloques try y catch

Sentencia throw

Bloque finally

La Clase Throwable

Tipos de Excepciones

- ◆ Existen tres tipos de excepciones: errores, comprobadas (en adelante *checked*) y no comprobadas (en adelante *unchecked*).
 - ◆
 - ◆ Una excepción de tipo checked representa un error del cual técnicamente podemos recuperarnos. Por ejemplo, una operación de lectura/escritura en disco puede fallar porque el fichero no exista, porque este se encuentre bloqueado por otra aplicación, etc.
 - ◆
 - ◆ Todos estas situaciones, además de ser inherentes al propósito del código que las lanza (lectura/escritura en disco) son totalmente ajenas al propio código, y deben ser (y de hecho son) declaradas y manejadas mediante excepciones de tipo checked y sus mecanismos de control.

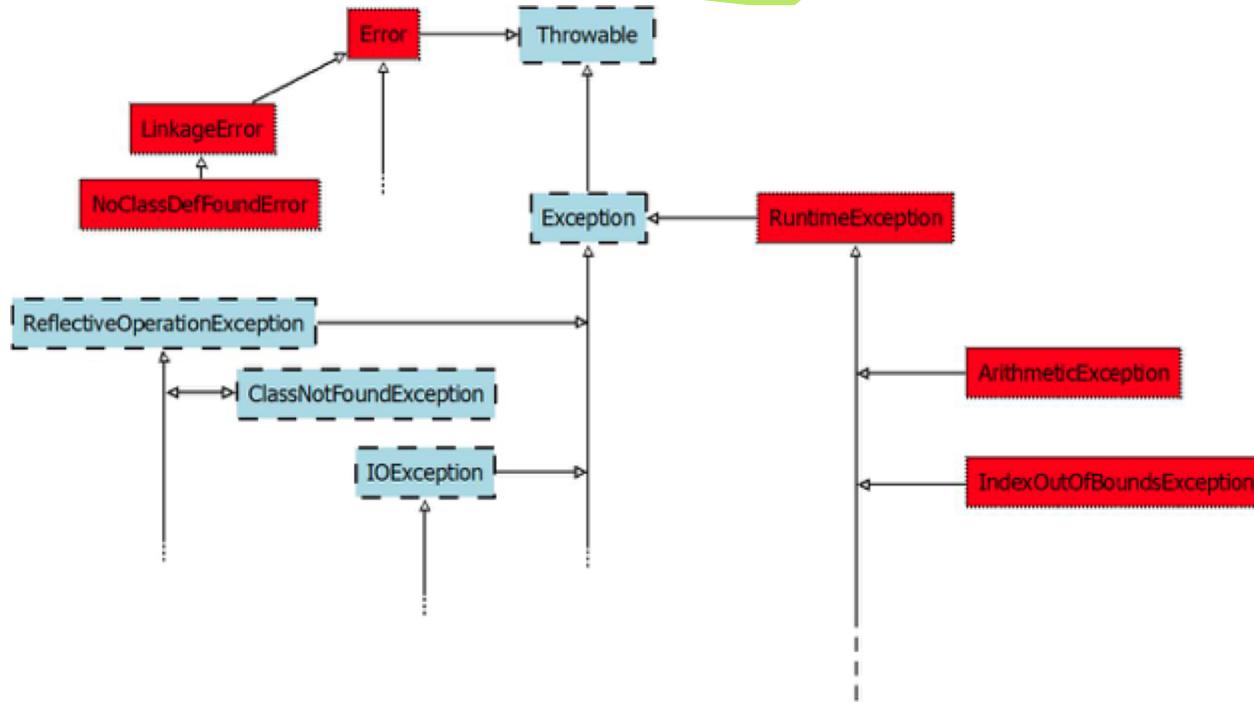
Tipos de Excepciones

- ◆ Una excepción de tipo unchecked representa un error de programación. Uno de los ejemplos más típicos es el de intentar leer en un array de N elementos un elemento que se encuentra en una posición mayor que N

```
int[] numerosPrimos = {1, 3, 5, 7, 9, 11, 13, 17, 19, 23};  
// Array de diez elementos  
  
int undecimoPrimo = numerosPrimos[10];  
// Accedemos al undécimo elemento mediante el literal numérico 10
```

- ◆ El aspecto más destacado de las excepciones de tipo unchecked es que no deben ser forzosamente declaradas ni capturadas (en otras palabras, no son comprobadas). Por ello no son necesarios bloques try-catch ni declarar formalmente en la firma del método el lanzamiento de excepciones de este tipo.

Tipos de Excepciones



Manejo de excepciones

- ◆ Para el manejo de excepciones los lenguajes proveen ciertas palabras reservadas, que nos permiten manejar las excepciones que puedan surgir y tomar acciones de recuperación para evitar la interrupción del programa o, al menos, para realizar algunas acciones adicionales antes de interrumpir el programa.
- ◆ Lo más común es encontrar: try, catch y finally.
- ◆ Dentro del bloque try se ubica todo el código que pueda llegar a *levantar* una excepción, se utiliza el término *levantar* o *lanzar* para referirse a la acción de generar una excepción.

Manejo de excepciones

- ◆ A continuación se ubica el bloque catch, que se encarga de capturar la excepción y nos da la oportunidad de procesarla mostrando por ejemplo un mensaje adecuado al usuario.
- ◆ Dado que dentro de un mismo bloque try pueden producirse excepciones de distinto tipo, es posible utilizar varios bloques catch, cada uno para capturar un tipo distinto de excepción.
- ◆ Finalmente, puede ubicarse un bloque finally donde se escriben las sentencias de finalización, que son típicamente acciones de limpieza.
- ◆ La particularidad del bloque finally es que se ejecuta siempre, haya surgido una excepción o no.

Procesamiento y propagación de excepciones

- ◆ Vamos a ver qué se supone que hagamos al atrapar una excepción. En primer lugar podríamos ejecutar alguna lógica particular del caso como: cerrar un archivo, realizar una procesamiento alternativo al del bloque try, etc.
- ◆ Pero más allá de esto tenemos algunas opciones genéricas que consisten en: dejar constancia de la ocurrencia de la excepción, propagar la excepción o, incluso, hacer ambas cosas.
- ◆ Es posible, por otra parte, que luego de realizar algún procesamiento particular del caso se quiera que la excepción se propague hacia la función que había invocado a la función actual. Para hacer esto Python nos brinda la instrucción raise y en Java throw

Procesamiento y propagación de excepciones

- ◆ También podría ocurrir que en lugar de propagar la excepción tal cual fue atrapada, quisiéramos lanzar una excepción distinta, más significativa para quien invocó a la función actual y que posiblemente contenga cierta información de contexto.

Para levantar una excepción de cualquier tipo, utilizamos también la sentencia `throw`, pero indicándole el tipo de excepción que deseamos lanzar y pasando a la excepción los parámetros con información adicional que queramos brindar.

Práctica de Propagación de Excepciones

```
public static void main (String args[]) {  
    try {  
        createFile();  
    }  
    catch (IOException ioe) {  
        System.out.println(ioe);  
    }  
}  
  
public static void createFile() throws IOException {  
    File testF = new File("c:/notWriteableDir");  
    File tempF = testF.createTempFile("te", null, testF);  
    System.out.println("Temp filename is " +  
        TempFile.getPath());  
    int myInt[] = new int[5];  
    myInt[5] = 25;
```

Procesamiento y propagación de excepciones

- ◆ Hay que tener presente que cuando se relanza una excepción estamos forzando al código cliente de nuestro método a capturarla o relanzarla. Una excepción que sea relanzada una y otra vez *hacia arriba* terminará llegando al método primigenio y, en caso de no ser capturada por éste, producirá la finalización de su hilo de ejecución (thread).
- ◆ La dos preguntas que debemos hacernos en este momento es:
 - ◆ ¿Cuándo capturar una excepción?
 - ◆ ¿Cuándo relanzarla?

La respuesta es muy simple.

Procesamiento y propagación de excepciones

- ◆ Capturamos una excepción cuando:

- ◆ Podemos recuperarnos del error y continuar con la ejecución

- ◆ Queremos registrar el error

- ◆ Queremos relanzar el error con un tipo de excepción distinto

- ◆ En definitiva, cuando tenemos que realizar algún tratamiento del propio error. Por contra, relanzamos una excepción cuando:

- ◆ No es competencia nuestra ningún tratamiento de ningún tipo sobre el error que se ha producido

Malas prácticas de uso

```
try { // Código que declara lanzar excepciones }
catch(Exception ex) {}
```

- ◆ El código anterior ignorará cualquier excepción que se lance dentro del bloque try, o mejor dicho, capturará toda excepción lanzada dentro del bloque try pero la silenciará no haciendo nada (frustrando así el principal propósito de la gestión de excepciones checked: *gestiónala o reláñzala*).
- ◆ Cualquier error de diseño, de programación o de funcionamiento en esas líneas de código pasará inadvertido tanto para el programador como para el usuario.

Malas prácticas de uso

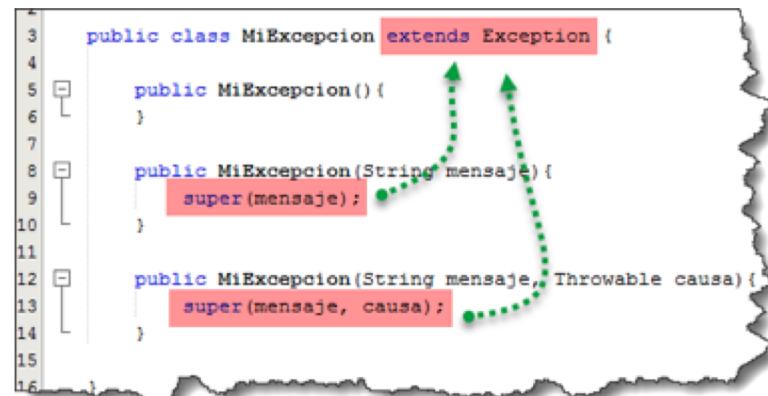
- ◆ Otro abuso del mecanismo de tratamiento de excepciones es cuando se está intentando escribir código que mejore el rendimiento de la aplicación:

```
try {  
    int i = 0;  
    while(true) {  
        System.out.println(numerosPrimos[i++]);  
    }  
} catch (ArrayIndexOutOfBoundsException aioobex)  
{...}
```

- ◆ El ejemplo anterior itera el array de números primos sin preocuparse de los límites del array hasta sobrepasar el índice máximo, momento en el cual se lanzará una excepción de tipo `ArrayIndexOutOfBoundsException` que será capturada y silenciada.

Excepciones Propias

- ◆ Excepción definida por el usuario o excepción personalizada es crear su propia clase de excepción y arroja esa excepción usando la palabra clave 'throw'. Esto puede hacerse extendiendo la clase Exception.
- ◆ Para crear nuestra propia excepción, no necesitamos mucho.



A screenshot of a Java code editor showing a class named 'MiExcepcion' extending the 'Exception' class. The code includes three constructors: one with no parameters, one taking a single string message, and one taking both a message and a Throwable cause. The 'super' calls in the constructors are highlighted with red boxes and connected by green dotted arrows. The code is numbered from 3 to 15 on the left.

```
3  public class MiExcepcion extends Exception {  
4  
5      public MiExcepcion(){  
6      }  
7  
8      public MiExcepcion(String mensaje){  
9          super(mensaje);  
10     }  
11  
12     public MiExcepcion(String mensaje, Throwable causa){  
13         super(mensaje, causa);  
14     }  
15  
16 }
```

Práctica de Excepciones Propias

- ◆ Crear excepciones propias para los casos
- ◆ Error de login
- ◆ Error de registro
- ◆ Error de reset de password
- ◆ Super clase para las 3 clases anteriores: GestionSignUp
- ◆ Validar el tipo de excepción en un método main

Buenas prácticas de uso

- ◆ Una de las novedades que incorporó Java 7 es la sentencia *try-with-resources* con el objetivo de cerrar los recursos de forma automática en la sentencia *try-catch-finally* y hacer más simple el código.
- ◆ Aquellas variables cuyas clases implementan la interfaz AutoCloseable pueden declararse en el bloque de inicialización de la sentencia *try-with-resources* y sus métodos close() serán llamados después del bloque *finally* como si su código estuviese de forma explícita.

```
public static String readFirstLineFromFile(String path) throws IOException {  
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {  
        return br.readLine();  
    }  
}
```

Recomendaciones de uso

- ◆ Un buen uso del tratamiento de excepciones es usar excepciones que ya existen, en lugar de crear las tuyas propias, siempre que ambas fueran a cumplir el mismo cometido (que es básicamente informar y, en caso de las checked, obligar a gestionar). Se suelen usar excepciones que ya existen cuando se dispone de un profundo conocimiento del API que se está usando (en otras palabras, *experiencia*).
- ◆ Si un argumento pasado a uno de tus métodos no es del tipo esperado, o no tiene el formato correcto, primero revisa si puedes usar una excepción existente: `IllegalArgumentException` en lugar de crear tu propia excepción. Esto es bueno porque:
 - ◆ Uno de los pilares de Java es la reutilización de código (no reinventes la rueda)
 - ◆ Tu código es más universal (`FormatoInvalidoException` puede no significar nada para otra personas)

Función multi-catch

Excepciones multi-catch se han añadido a Java SE 7 para facilitar el manejo de excepciones de forma más fácil y más concisa, las mismas se agregan en un solo bloque catch y se separan con |:

```
try{  
    // Código que puede generar las excepciones  
}  
catch( ParseException | IOException exception){  
    // Código para manejo de la excepción  
}
```

Colecciones y Generics



Generics

- ◆ Los *generics* son importantes ya que permiten al compilador informar de muchos errores de compilación que hasta el momento solo se descubrirían en tiempo de ejecución
- ◆ Permiten eliminar los *cast* simplificando, reduciendo la repetición y aumentando la legibilidad el código.
- ◆ Los *generics* permiten usar tipos para parametrizar las clases, interfaces y métodos al definirlas.
- ◆ Los beneficios:
- ◆ Comprobación de tipos más fuerte en tiempo de compilación.
- ◆ Eliminación de *casts* aumentando la legibilidad del código.
- ◆ Posibilidad de implementar algoritmos genéricos, con tipado seguro

Generics

```
public class Limite<T> {  
  
    private T t;  
  
    public T get() { return t; }  
  
    public void set(T t) { this.t = t; }  
  
}
```

```
Limite <Integer> caja1 = new Limite <Integer>();
```

```
Limite <Integer> caja2 = new Limite <>();
```

Práctica de Generics

```
public interface Pair<K, V> {  
    public K getKey();  
    public V getValue();  
}
```

```
public class OrderedPair<K, V> implements Pair<K, V> {  
    private K key;  
    private V value;  
    public OrderedPair(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
    public K getKey() { return key; }  
    public V getValue() { return value; }  
}
```

```
Pair<Integer, String> p1 = new OrderedPair<>(1, "rojo");  
Pair<Integer, String> p2 = new OrderedPair<>(2, "azul");
```

Generics

- ◆ Para nombrar Generics se acostumbra a usar:
 - ◆ E: elemento de una colección.
 - ◆ K: clave.
 - ◆ N: número.
 - ◆ T: tipo.
 - ◆ V: valor.
 - ◆ S, U, V etc: para segundos, terceros y cuartos tipos.

Generics

- ◆ En los *generics* se puede usar un parámetro para un tipo con ? Cuando es un tipo desconocido.
- ◆ Son usados para reducir las restricciones de un tipo de modo que un método pueda funcionar con una lista de *List<Integer>*, *List<Double>* y *List<Number>*.
- ◆ El término *List<Number>* es más restrictivo que *List<? extends Number>* porque el primero solo acepta una lista de *Number* y el segundo una lista de *Number* o de sus subtipos.
- ◆ *List<? extends Number>* es un *upper bounded wildcard*.
- ◆ `public static void process(List<? extends Number> list) { /* ... */ }`

Genéricos upper bounded

```
◆ public class Limite<T> {  
    ◆ private T t;  
    ◆ public void set(T t) {  
        ◆     this.t = t;  
    ◆ }  
    ◆ public T get() {  
        ◆     return t;  
    ◆ }  
    ◆ public <U extends Number> void revision(U u){  
        ◆     sout("T: " + t.getClass().getName());  
    ◆ }  
}
```

```
public static void main(String[] args) {  
    Limite<Integer> obj = new  
    Limite<Integer>();  
    obj.set(new Integer(10));  
    obj.revision("Cadena ejemplo"); // error!!  
}
```

Generics Lower Bounded

- ◆ Otro caso:
- ◆ queremos definir un método que inserte objetos *Integer* en un *List*.
- ◆ Para mayor flexibilidad queremos que ese método pueda trabajar con cualquier tipo de lista que permita contener *Integer*, ya sea *List<Integer>*, *List<Number>* y *List<Object>* Es decir, clases padre de *Integer*.
- ◆ Lo podemos conseguir definiendo *List<? super Integer>* que se conoce como *Lower Bounded Wildcard*.

PRÁCTICA: crear variante del ejercicio anterior para usar la presente estrategia

Colecciones

En Java se dispone de interfaces para trabajar con colecciones las cuales son: Collection, Set, List, Queue y Map:

- **Collection<E>**: Un grupo de elementos individuales, frecuentemente con alguna regla aplicada a ellos.
- **List<E>**: Elementos en una secuencia particular que mantienen un orden y permite duplicados. La lista puede ser recorrida en ambas direcciones con un ListIterator.

Hay 3 tipos de constructores:

- **ArrayList<E>**: Su ventaja es que el acceso a un elemento en particular es ínfimo. Su desventaja es que para eliminar un elemento, se ha de mover toda la lista para eliminar ese “hueco”.
- **Vector<E>**: Es igual que ArrayList, pero sincronizado. Es decir, que si usamos varios hilos, no tendremos de qué preocuparnos hasta cierto punto.
- **LinkedList<E>**: En esta, los elementos están conectados con el anterior y el posterior. La ventaja es que es fácil mover/eliminar elementos de la lista, simplemente moviendo/eliminando sus referencias hacia otros elementos. La desventaja es que para usar el elemento N de la lista, debemos realizar N movimientos a través de la lista.

Colecciones

- **Set<E>**: No puede haber duplicados. Cada elemento debe ser único, por lo que si existe uno duplicado, no se agrega. Por regla general, cuando se redefine equals(), se debe redefinir hashCode(). Es necesario redefinir hashCode() cuando la clase definida será colocada en un HashSet.
- **Map<K,V>**: Un grupo de pares objeto clave-valor, que no permite duplicados en sus claves. Es quizás el más sencillo, y no utiliza la interfaz Collection. Los principales métodos son: put(), get(), remove().
- **HashMap<K,V>**: Se basa en una tabla hash, pero no es sincronizado.
- **HashTable<K,V>**: Es sincronizado, aunque que no permite null como clave.
- **LinkedHashMap<K,V>**: Extiende de HashMap y utiliza una lista doblemente enlazada para recorrerla en el orden en que se añadieron. Es ligeramente más rápida a la hora de acceder a los elementos que su superclase, pero más lenta a la hora de añadirlos.
- **TreeMap<K,V>**: Se basa en una implementación de árboles en el que ordena los valores según las claves. Es la clase más lenta.

Creación e inicialización de una ArrayList

Un ArrayList es una colección de objetos redimensionable en la que están disponibles los métodos más habituales para el manejo de matrices.

Esta clase pertenece a la biblioteca java.util y contiene métodos para agregar, borrar, clorar, modificar, obtener el índice entre otros.

Ejemplo:

```
ArrayList<String> nombreArrayList = new ArrayList<String>(); // Declaración de un ArrayList  
nombreArrayList.add("Elemento"); // Añade el elemento al ArrayList  
nombreArrayList.add(n, "Elemento 2"); // Añade el elemento al ArrayList en la posición 'n'  
nombreArrayList.size(); // Devuelve el numero de elementos del ArrayList  
nombreArrayList.get(2); // Devuelve el elemento que esta en la posición '2' del ArrayList
```

Acceso a un valor de una matriz o ArrayList

Para acceder a los valores contenidos en un ArrayList existen métodos útiles para tal objetivo:

- **get(i)**: Obtiene el elemento en la posición i del ArrayList.
- **size()**: Retorna el tamaño del ArrayList.
- **contains(X)**: Retorna true si existe el elemento X en el ArrayList.

Ejemplo:

```
ArrayList <Integer> miarray= new ArrayList<>();
    for (int i = 0; i < miarray.size(); i++) {
        int value = miarray.get(i);
        System.out.println("miarray: " + value);
    }
    int count = miarray.size();
    System.out.println("Count: " + count);
```

Ejercicio de Colecciones: Mapas

- ◆ Necesitamos que el sistema de la aerolínea pueda imprimir los billetes de cada pasajero para cada vuelo. Un billete solo tiene un vuelo asociado y la norma es: un billete por pasajero. Si viajan en familia cada uno tendrá su billete independiente.
- ◆ Los billetes tienen un número de asiento asociado.
- ◆ La aerolínea debe tener una forma de poder ver los billetes emitidos por fecha, es decir, poder tener para X día todos los billetes emitidos (Requerimiento obligatorio para este punto: usar mapas y optimizar lo mejor posible el código para no tener mapas demasiado grandes)
- ◆ Para ver los billetes emitidos, la aerolínea llamará a `verBilletesPorFecha` y solo se mostrarán los de determinada fecha

Depuración de Errores



Depuración de Errores

- ◆ Existen muchos casos en los que se hace difícil entender lo que pasa en un programa. La solución es depurar (debug) el código del programa.
- ◆ La máquina virtual de Java, tiene capacidades de depuración de código incluso desde una máquina remota. Eclipse hace que la depuración sea sencilla.
- ◆ Depurar un programa es recorrerlo paso a paso, viendo el valor de las variables del programa. Esto nos permite ver exactamente qué está pasando en nuestro programa cuando se está ejecutando una línea de código específica.

Depuración de Errores

- ◆ El depurador permite controlar la ejecución, es decir, decidir exactamente dónde desea pausar todos los subprocessos del proceso e inspeccionar el estado en dicho punto.
- ◆ Puede dividir todo en cualquier momento, saltar las instrucciones, entrar o saltar las funciones, ejecutar hasta el cursor, editar y continuar, y la favorita de todo el mundo, establecer puntos de interrupción.

Depuración de Errores

- ◆ Un punto de interrupción es un comando que se ejecuta al llegar la ejecución de un programa a la línea especificada, y que interrumpe su ejecución en este punto.
- ◆ Los puntos de interrupción permiten investigar el comportamiento del programa en un segmento determinado: ver los valores de las variables, la pila de funciones, etcétera. Luego podemos reanudar o finalizar el proceso de depuración.
- ◆ Se debe establecer al menos un punto de interrupción en el código del programa antes de iniciar la depuración.

Depuración de Errores

- ◆ En cuanto se pause la los depuradores ofrecerán muchas maneras de inspeccionar el valor de las variables para formar o comprobar una hipótesis.
- ◆ Podemos supervisar un valor mientras revisamos el código. Esto nos permite ver las variables locales y evaluar expresiones complejas (todo sin abandonar el depurador).
- ◆ Incluso puede consultarse de manera interactiva niveles profundos de la estructura de datos.

I/O & Threads



La plataforma Java es compatible con interacción a desde la consola de comandos de dos maneras:

- **Flujos estándares**

Los flujos estándar son una característica de muchos sistemas operativos de forma predeterminada leen la entrada desde el teclado y escriben la salida a la pantalla. La plataforma Java es compatible con tres flujos estándar:

- **Entrada estándar:** Esto se utiliza para alimentar con datos al programa del usuario y por lo general un teclado se utiliza como flujo de entrada estándar y representa como **System.in**.

La plataforma Java es compatible con interacción a desde la consola de comandos de dos maneras:

- **Flujos estándares**
- **Salida estándar:** Esto se utiliza para enviar los datos producidos por el programa del usuario y por lo general una pantalla de ordenador se utiliza para flujo de salida estándar y representa como `System.out`.
- **Error estándar:** Esto se utiliza para enviar los datos de error producidos por el programa del usuario y por lo general una pantalla de ordenador se utiliza para flujo de error estándar y representa como `System.err`.

Lectura del flujo de entrada de la consola

```
import java.io.*;  
  
public class ReadConsole {  
    public static void main(String args[]) throws IOException{  
        InputStreamReader cin = null;  
        try {  
            cin = new InputStreamReader(System.in);  
            System.out.println("Ingrese texto!, 'q' para salir.");  
            char c;  
            do {  
                c = (char) cin.read();  
                System.out.print(c);  
            } while(c != 'q');  
        }finally {  
            if (cin != null) {  
                cin.close();  
            }  
        }  
    }  
}
```

Programación de tareas del sistema operativo

Java soporta la programación concurrente o multihilo, con el apoyo de concurrencia básica en el lenguaje de programación Java y las bibliotecas de clases de Java.

Desde la versión 5.0, la plataforma Java ha incluido en su API librerías de concurrencia de alto nivel en los paquetes `java.util.concurrent`.

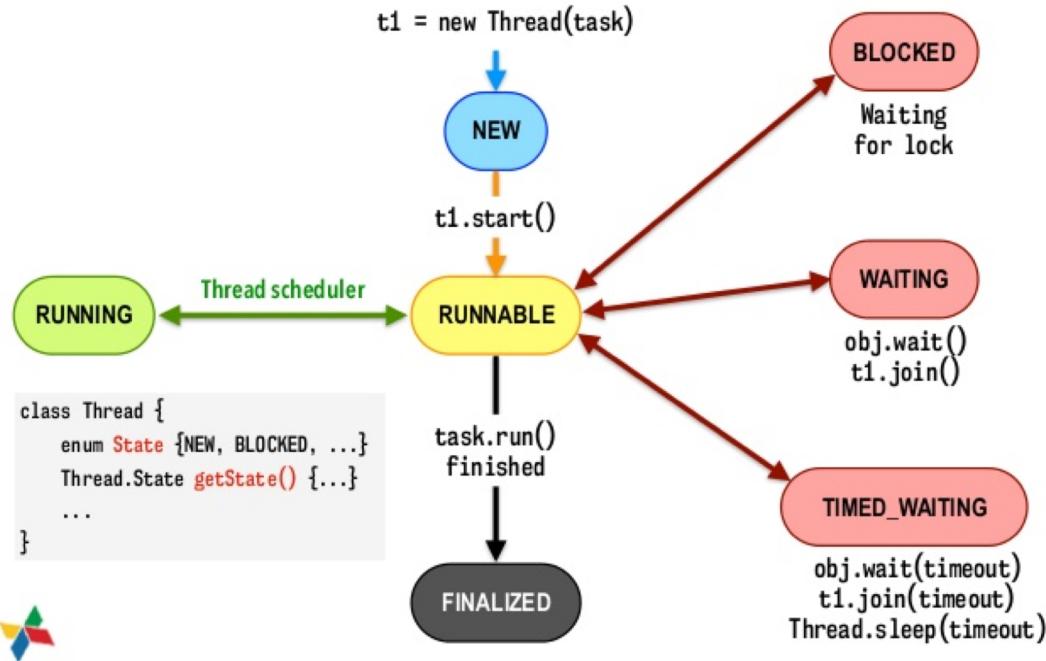
Un programa multihilo contiene dos o más partes que se pueden ejecutar al mismo tiempo y cada parte puede manejar diferentes tareas al mismo tiempo, haciendo un uso óptimo de los recursos disponibles, especialmente cuando el equipo tiene varias CPU.

Reconocimiento de entornos multithread

En la programación concurrente, hay dos unidades básicas de ejecución: procesos y subprocesos(hilos).

- **Procesos**
 - Un proceso tiene un entorno de ejecución autónomo.
 - Un proceso en general, tiene un juego completo, privado de los recursos básicos de tiempo de ejecución; en particular, cada proceso tiene su propio espacio de memoria.
-
- **Hilos**
 - La creación de un nuevo hilo requiere menos recursos que la creación de un nuevo proceso.
 - Un hilo debe existir dentro de un proceso - cada proceso tiene al menos uno.
 - Los hilos comparten los recursos del proceso, incluyendo la memoria y los archivos abiertos.
Esto hace que la comunicación sea mas eficiente, pero potencialmente problemática.

Ciclo de Vida de un hilo



Creación de soluciones multithread

Existen dos maneras de implementar multithread y crear hilos:

- Heredando de la clase Thread, y sobrescribir su método run().

```
class Hilo extends Thread{  
    public void run(){  
        ....  
    }  
}
```

- Implementando la interfaz Runnable proporcionando la implementación solo para su método run().

```
class Hilo implements Runnable{  
    public void run(){  
        .....  
    }  
}
```

Arranque de un Thread

Las aplicaciones ejecutan *main()* tras arrancar. Esta es la razón de que *main()* sea el lugar natural para crear y arrancar otros threads.

```
Hilo hijo = new Hilo("Hijo");
hijo.start();
System.out.println("main en ejecucion!");
```

Manipulación de un Thread

Todo lo que queramos que haga el thread ha de estar dentro de *run()*, por ejemplo, intentamos esperar durante una cantidad de tiempo en segundos

```
hijo.sleep(1000);
```

Suspensión de un Thread

Puede resultar útil suspender la ejecución de un thread sin marcar un límite de tiempo. Para este tipo de control de thread se puede utilizar el método suspend().

```
hijo.suspend();
```

Este método no detiene la ejecución permanentemente. El thread es suspendido indefinidamente y para volver a activarlo de nuevo necesitamos realizar una invocación al método resume

```
hijo.resume();
```

Parada de un Thread

El último elemento de control que se necesita sobre threads es el método *stop()*. Se utiliza para terminar la ejecución de un thread:

```
hijo.stop();
```

Stop() no destruye el thread, sino que detiene su ejecución. La ejecución no se puede reanudar con *start()*. Cuando se liberen las variables que se usan en el thread, el objeto thread quedará marcado para eliminarlo y el *garbage collector* se encargará de liberar la memoria que utilizaba.

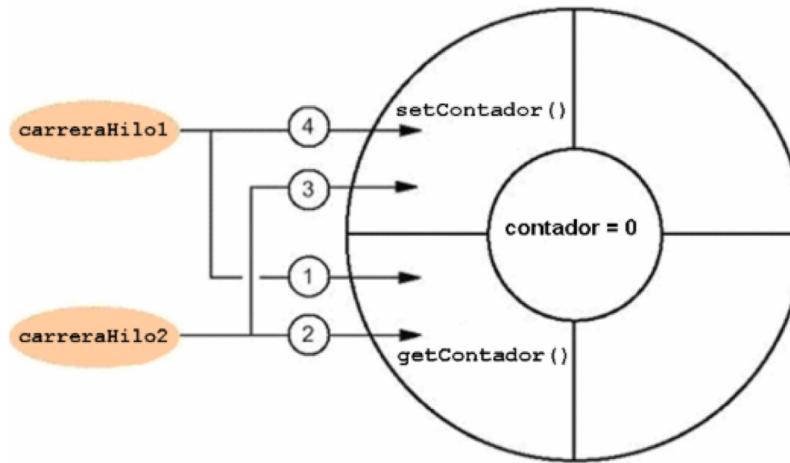
Si se necesita, se puede comprobar si un thread está vivo o no; considerando vivo un thread que ha comenzado y no ha sido detenido.

```
hijo.isAlive();
```

Uso compartido de datos en threads

A la hora de acceder a datos comunes los thread necesitan cierto orden para asegurarse que los hilos concurrentes no interfieren entre si y operan correctamente con los datos.

En la figura de ejemplo se observa 2 hilos que acceden al mismo recurso y como no esta sincronizado el acceso a ese dato compartido presenta inconsistencia.



Después de ejecutar ambos hilos y actualizar contador, esta incorrectamente en 1 en vez de 2

Sincronización e interbloqueos

- Los hilos se comunican principalmente mediante el intercambio de acceso a los campos y los objetos de referencia se refieren a los campos.
- Esta forma de comunicación es muy eficiente, pero hace dos tipos de errores posibles:
 - Interferencia hilo
 - Errores de coherencia de memoria.
- La herramienta necesaria para evitar estos errores es la sincronización.
- Sin embargo, la sincronización puede introducir discordia de hilo, que se produce cuando dos o más hilos intentan acceder al mismo recurso al mismo tiempo y pueden hacer que el tiempo de ejecución de Java para ejecutar uno o más hilos sea más lento, o incluso suspender su ejecución.

Sincronización

La clave para la **sincronización en Java** es el concepto de **monitor**, que controla el acceso a un objeto. Un *monitor* funciona implementando el concepto de **bloqueo** (*lock*). Cuando un objeto está bloqueado por un hilo, ningún otro hilo puede obtener acceso al objeto. Cuando el hilo sale, el objeto está desbloqueado y está disponible para ser utilizado por otro hilo.

Todos los objetos en Java tienen un monitor. Esta característica está integrada en el lenguaje Java en sí. Por lo tanto, **todos los objetos se pueden sincronizar.** La sincronización está respaldada por la palabra clave **synchronized** y algunos métodos bien definidos que tienen todos los objetos.

Sincronización

Podemos sincronizar el acceso a un método modificándolo con la palabra clave **synchronized**. Cuando se llama a ese método, el hilo de llamada entra en el monitor del objeto, que luego bloquea el objeto.

- Mientras está bloqueado, ningún otro hilo puede ingresar al método, o ingresar cualquier otro método sincronizado definido por la clase del objeto.
- Cuando el hilo retorna del método, el monitor desbloquea el objeto, permitiendo que sea utilizado por el siguiente hilo. Por lo tanto, la sincronización se logra con prácticamente ningún esfuerzo de programación de tu parte.

```
class sumArray{  
    private int sum;  
  
    //sumArray está sincronizado  
    synchronized int sumArray(int nums[]){  
        sum=0;  
        for (int i=0; i<nums.length;i++){  
            sum+=nums[i];  
            System.out.println("Total acumulado de "+Thread.currentThread().getName()+" es  
"+sum);  
        try {  
            Thread.sleep(10);//permitir el cambio de tarea  
        }catch (InterruptedException exc){  
            System.out.println("Hilo interrumpido");  
        }  
    }  
    return sum;  
}
```

```
class MiHilo implements Runnable{  
    Thread hilo;  
    static sumArray sumarray= new sumArray();  
    int a[];  
    int resp;  
  
    //Construye un nuevo hilo.  
    MiHilo(String nombre, int nums[]){  
        hilo= new Thread(this,nombre);  
        a=nums;  
    }  
  
    //Un método que crea e inicia un hilo  
    public static MiHilo creaEInicia (String nombre,int nums[]){  
        MiHilo miHilo=new MiHilo(nombre,nums);  
  
        miHilo.hilo.start(); //Inicia el hilo  
        return miHilo;  
    }  
    //Punto de entrada del hilo  
    public void run(){  
        int sum;  
        System.out.println(hilo.getName()+" iniciando.");  
  
        resp=sumarray.sumArray(a);  
        System.out.println("Suma para "+hilo.getName()+" es "+resp);  
        System.out.println(hilo.getName()+" terminado.");  
    }  
}
```



```
class Sincronizacion {  
    public static void main(String[] args) {  
        int a[]={1,2,3,4,5};  
        MiHilo mh1 = MiHilo.creaEInicia("#1",a);  
        MiHilo mh2 = MiHilo.creaEInicia("#2",a);  
  
        try {  
            mh1.hilo.join();  
            mh2.hilo.join();  
        }catch (InterruptedException exc){  
            System.out.println("Hilo principal interrumpido.");  
        }  
    }  
}
```



Practica en Pair-Programming

- ◆ Se requiere crear una aplicación de carrito de compra.
- ◆ La empresa tiene Productos, los productos tienen nombre, descripción, categoría y precio.
- ◆ Las categorías de los productos son: Alimentación, Perfumería, Limpieza
- ◆ La empresa puede incrementar o disminuir la cantidad de stock de los productos que tienen, lo puede hacer para cada uno.
- ◆ La empresa tiene clientes y los clientes tendrán nombres, apellidos, dirección y teléfono.
- ◆ Además, la empresa debe guardar un historial de compra de los clientes. La compra debe tener una fecha asociada y un total.
- ◆ Los clientes pueden comprar la cantidad de productos que quieran siempre y cuando haya stock

Database



- ◆ Para conectarnos a una base de datos necesitamos el driver del fabricante del manejador de la base de datos. Por ejemplo:
- ◆ <https://dev.mysql.com/downloads/connector/j/5.1.html>
- ◆ Descargamos, extraemos el jar y lo añadimos como librería externa al proyecto

JDBC

- ◆ La conexión a Base de Datos requiere 3 elementos:
- ◆ URL: jdbc:[manejador]://[host]:[puerto]/[db]
 - ◆ jdbc:mysql://sql7.freemysqlhosting.net:3306/sql7264745
- ◆ Usuario
- ◆ Contraseña

JDBC – MySQL Server

- ◆ Servidor online gratuito con MySQL:
<https://www.freemysqlhosting.net/>
- ◆ Guardar parámetros de conexión

Conexión a BD

```
Connection conn1 = null;  
try {  
    String url1 = "jdbc:mysql://sql7.freemysqlhosting.net:3306/sql7264745";  
    String user = "sql7264745";  
    String password = "Zb1XGdwrD6";  
  
    conn1 = DriverManager.getConnection(url1, user, password);  
    if (conn1 != null) {  
        System.out.println("Connected to the database sql7264745");  
    }  
  
} catch (SQLException ex) {  
    System.out.println("Error en la conexión, usuario /password incorrecto");  
    ex.printStackTrace();  
}
```

Almacenamiento

```
String query = " insert into categorias values (?, ?);  
PreparedStatement preparedStmt =  
    conn1.prepareStatement(query);  
preparedStmt.setInt (1, 10);  
preparedStmt.setString (2, "Alimentacion");  
  
preparedStmt.execute();  
  
conn1.close();
```

Consulta

```
String query = "SELECT * FROM categorias";
Statement st = conn1.createStatement();
ResultSet rs = st.executeQuery(query);
while (rs.next())
{
    int id = rs.getInt("id");
    String nombre = rs.getString("nombre");
    System.out.format("%s, %s \n", id, nombre);
}
st.close();
```

Maven



Apache Maven

- ◆ Maven es una herramienta principalmente utilizada en el desarrollo de software Java. Aparece ante la necesidad de modelar el concepto de "proyecto" y artefacto en forma estándar independientemente del IDE de desarrollo.



Apache Maven

- ◆ Se utiliza para
 - ◆ el versionado de nuestros proyectos,
 - ◆ las dependencias con proyectos nuestros y librerías de terceros
 - ◆ y para la automatización de tareas relativas al ambiente de desarrollo y construcción del artefacto (compilación, generación de código, empaquetado, etc)

Apache Maven

- ◆ Maven usa el concepto **build lifecycle**, este concepto indica que el proceso de construcción y distribución de un artefacto(proyecto) está claramente definido.
- ◆ Maven define desde la estructura de un proyecto hasta los pasos de construcción para que podamos construir nuestro proyecto con el minimo de configuraciones y de forma facil.
- ◆ Maven viene integrado con tres lifecycles: **default, clean y site**.
 - ◆ El lifecycle **default** maneja el despliegue de tu proyecto,
 - ◆ el lifecycle **clean** maneja limpiezas del proyecto
 - ◆ el lifecycle **site** maneja la creación de los sitios de documentación de tu proyecto.
- ◆ Cada uno de estos lifecycle está compuesto por una lista de **build phases**, un build phase se define como una etapa en el lifecycle.

Build Phases

- ◆ **validate** - Valida que el proyecto esta correcto y toda la información requerida está disponible
- ◆ **compile** - Compila el codigo fuente del proyecto.
- ◆ **test** - Prueba el codigo fuente compilado usando un framework de testing unitario.
- ◆ **package** - Toma el código fuente compilado y lo empaqueta dentro de un formato distribuible, tal com JAR o WAR.
- ◆ **verify** - Ejecuta algunas verificaciones sobre los resultados de pruebas de integración para asegurar que pasen los criterios de calidad.
- ◆ **install** - Instala el package (jar) en el repositorio local, para que sea usado como dependencia en otros proyectos localmente.
- ◆ **deploy** - Se realiza en el ambiente de producción, copia el package final al repositorio remoto para ser compartido con otros desarrolladores y proyectos.

Apache Maven

- ◆ Todo proyecto maven se constituye de un archivo XML llamado pom.xml (de *Project Object Model*)
- ◆ El archivo pom.xml de Maven contiene las configuraciones para construir y gestionar un proyecto.
- ◆ Basicamente aquí se definen las propiedades del proyecto, las dependencias del proyecto(librerias **jars** de terceros o propios) y los plugins de maven a ejecutar.

Apache Maven – pom.xml

- ◆ Lo más importante es que se **declara** un identificador único de nuestro proyecto/artefacto, que resulta de la unión de tres identificadores:
 - ◆ **groupId** : representa la organización autora/dueña del artefacto. Por ejemplo: com.ibm, org.apache, org.jboss, etc. Este id por sí mismo no identifica un artefacto por las razones obvias de que una organización puede tener más de un artefacto.
 - ◆ **artifactId** : nombre del proyecto/artefacto actual. Por ejemplo: http-client, hibernate, tomcat, commons-collections, etc.
 - ◆ **version** : como su nombre dice, identifica al número de versión del artefacto.

Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>es.indra</groupId>
    <artifactId>carrito</artifactId>
    <version>1.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Carrito Project</name>

</project>
```

Repositorios Maven

- ◆ Un repositorio es el lugar donde los artefactos (jar, war, ear) construidos son almacenados.
- ◆ Hay dos tipos de repositorios, local y remoto. Los repositorios locales están en el directorio del usuario del sistema, más específico, en Windows la ruta por defecto es "C:\Users\{usuario}\m2\repository". En este directorio se almacena una copia de todos los archivos que son usados en nuestro proyecto como **dependencias**.
- ◆ El repositorio remoto se encuentra en un servidor remoto conectado a la red Internet, este es gestionado por un tercero.

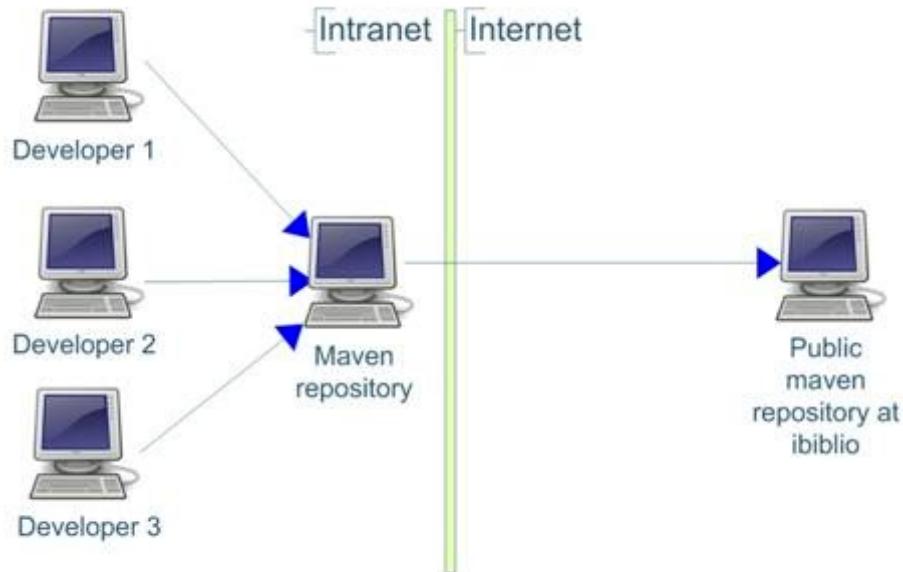
<https://mvnrepository.com/>

- ◆ Este sitio es básicamente un buscador de proyectos/artefactos maven, que estén públicos en internet.
- ◆ Un artefacto bastante popular es el framework **spring**. En <https://mvnrepository.com/artifact/org.springframework/spring-core>
- ◆ Maven establece un estándar a nivel mundial. Tanto nuestros nuevos proyectos como todos los de la comunidad java+maven van a tener la misma estructura e identificación:
 - ◆ <groupId>org.springframework</groupId>
 - ◆ <artifactId>spring-core</artifactId>
 - ◆ <version>5.1.2.RELEASE</version>

Repository Empresarial

- ◆ **Repository "Empresarial" (o repo remoto):** en esta arquitectura, la empresa u organización requiere tener un repositorio maven propio, ya sea para publicar sus propios artefactos solo dentro de la organización, sin hacerlo público a toda internet, o bien para evitar que cada maven de cada desarrollador acceda a internet para bajarse los artefactos.
- ◆ Básicamente, se instalará una aplicación en un servidor, y luego se configura maven en cada máquina de los desarrolladores para que utilicen este repositorio.

Repositorio Empresarial



Dependencias Maven

- ◆ Dependencias son aquellas librerías (jars) externas que nuestro proyecto requiere, por ejemplo el famoso Log4J, librería para escribir registros.
- ◆ Toda dependencia que nuestro proyecto requiera debe ser declarado en nuestro POM (pom.xml).
- ◆ Maven buscará estas dependencias en nuestro repositorio local, si no los encuentra entonces los descargará desde el repositorio remoto y los guardará en el repositorio local.
- ◆ Para agregar una dependencia tenemos un elemento disponible, el elemento dependency.

Ejemplo de Dependencias

- ◆ <dependency>
- ◆ <groupId>log4j</groupId>
- ◆ <artifactId>log4j</artifactId>
- ◆ <version>1.2.17</version>
- ◆ <type>jar</type>
- ◆ <dependency>
- ◆ Generalmente para agregar una dependencia lo primero será asegurarnos que se encuentra en el repositorio central de Maven, definirlo en el POM incluyendo el groupId, artifactId, version, type y scope.
- ◆ Lo que yo hacemos es buscar el artifactId en el repositorio central de Maven, copiar la porcion del xml y pegarlo en mi POM.

Estructura de un proyecto en Maven

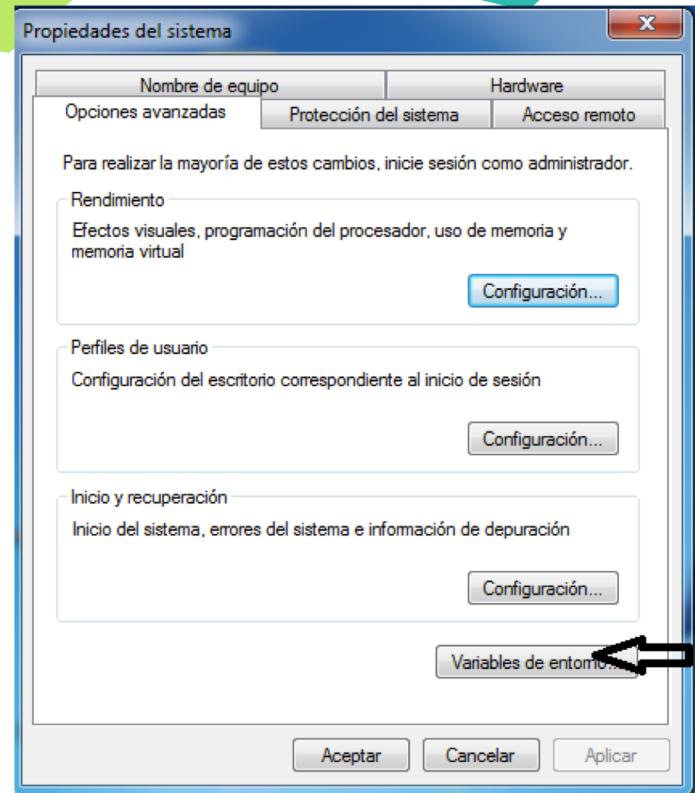
- ◆ Maven tiene bien definido una convención para la estructura de directorios de un proyecto.
- ◆ Si seguimos esta convención no necesitaremos configuraciones adicionales para definir donde están localizados los elementos.
- ◆ Maven conoce donde se encuentra el código fuente, casos de pruebas, recursos, etc.,
- ◆ Las carpetas más importantes son: src, resources, webapp, test, target.

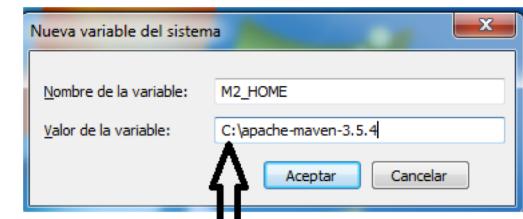
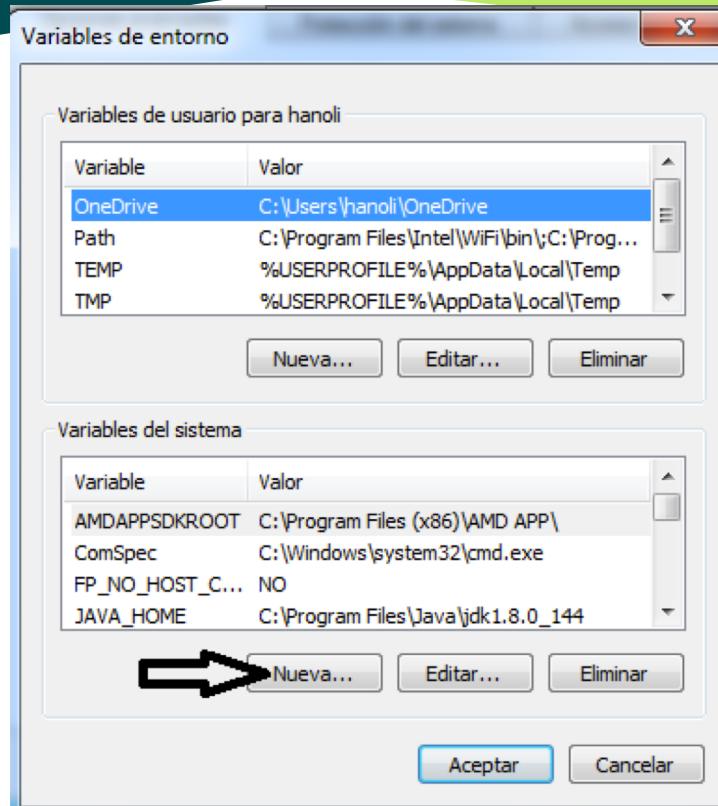
Instalación de Maven

- ◆ Maven es un proyecto Open Source que se encuentra disponible en su pagina oficial y para ello solo tenemos que dirigirnos al siguiente link:
 - ◆ <https://maven.apache.org/download.cgi>
- ◆ Descargamos el ZIP
- ◆ Una vez que tenemos descargado Maven y descomprimido en nuestro disco local C, procederemos a crear nuestra variable de entorno

Creación de Variable de entorno

- ◆ Para crear nuestra variable de entorno con la que trabajara Windows con maven nos dirigimos a
 - ◆ Inicio -> Equipo -> Click Derecho -> Propiedades
- ◆ Nos dirigimos a “Configuración Avanzadas del sistema”
- ◆ Nos aparecerá la ventana de “Propiedades del sistema”, damos click en Variables de entorno.
- ◆ Nos aparecerá la ventana de variables de entorno y damos click en el botón “Nueva”, para crear una nueva variable.





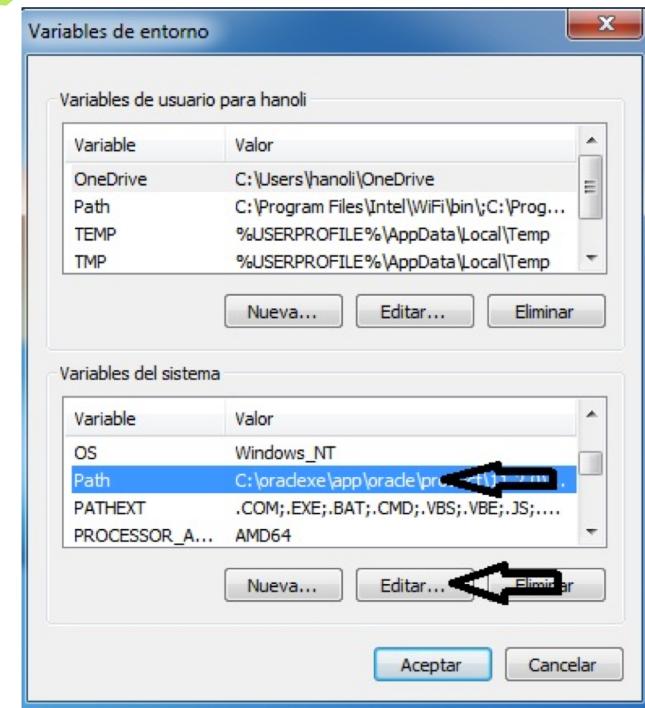
Ruta donde descomprimimos la carpeta de Maven

Una vez le hemos dado click en “Nueva” nos lanzara el cuadro donde ingresaremos el nombre de la variable, la cual llamaremos M2_HOME y la ruta donde se encuentra nuestra carpeta de Maven.

Salimos de todas la ventanas dando click en “Aceptar” para que se cree nuestra nueva configuración.

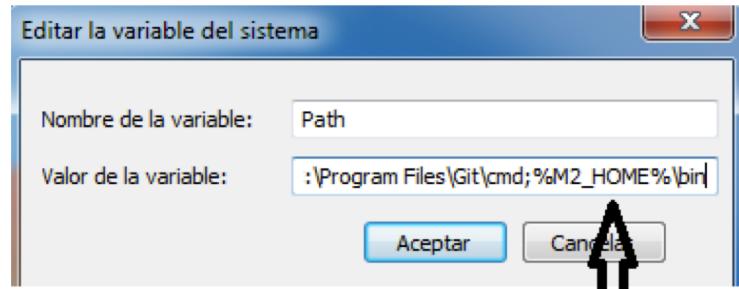
Modificando PATH

- ◆ Agregando Variable de entorno M2_HOME creada, a la variable PATH para que Windows la reconozca.
- ◆ Para editar la variable de entorno Path, Ingresamos a la misma ruta donde agregamos nuestra variable M2_HOME de Maven pero ahora en el apartado “Variable del sistema” buscamos la variable “Path” y damos click en Editar.



Modificando PATH

- ◆ Nos abrirá la ventana de edición de la variable, en donde agregaremos nuestra variable de entorno M2_HOME dentro de los signos de porcentaje % %, pero algo que destacar es que se agregara la ruta de la carpeta bin, quedando de la siguiente manera %M2_HOME%\bin , cuidando que se agregue el punto y coma para separarlo de las otras variables que están alojadas en Path.



A la variable M2_HOME, no se le agrego la ruta \\bin, por ello en esta parte la agregamos

Revisando la Instalación

- ◆ Cerramos las consolas de git bash abiertas, abrimos una nueva y probamos el comando:
 - ◆ mvn -v

Creando nuestro primer proyecto en Maven

- ◆ Lo primero que haremos será abrir una ventana de comandos, navegar hasta el directorio donde queremos crear el proyecto, luego ejecutar el siguiente comando:
- ◆ `mvn archetype:generate -DgroupId=es.indra
-DartifactId=demo-maven
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false`
- ◆ Lo que hemos hecho es crear un proyecto vacío, un proyecto con una estructura de directorios definido, en donde podemos ir agregando clases, recursos, etc.,

Creando nuestro primer proyecto maven

- ◆ Podemos ejecutar la tarea de empaquetado:
 - ◆ **`mvn package`**
- ◆ Si necesitamos instalar este artefacto en nuestro repositorio local, solo tenemos que ejecutar el build phase **install**
 - ◆ **`mvn install`**
- ◆ Abriendo el proyecto en eclipse...

J2EE



¿Qué es JEE?

Es una plataforma de programación –parte de la plataforma Java— para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java.

Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones



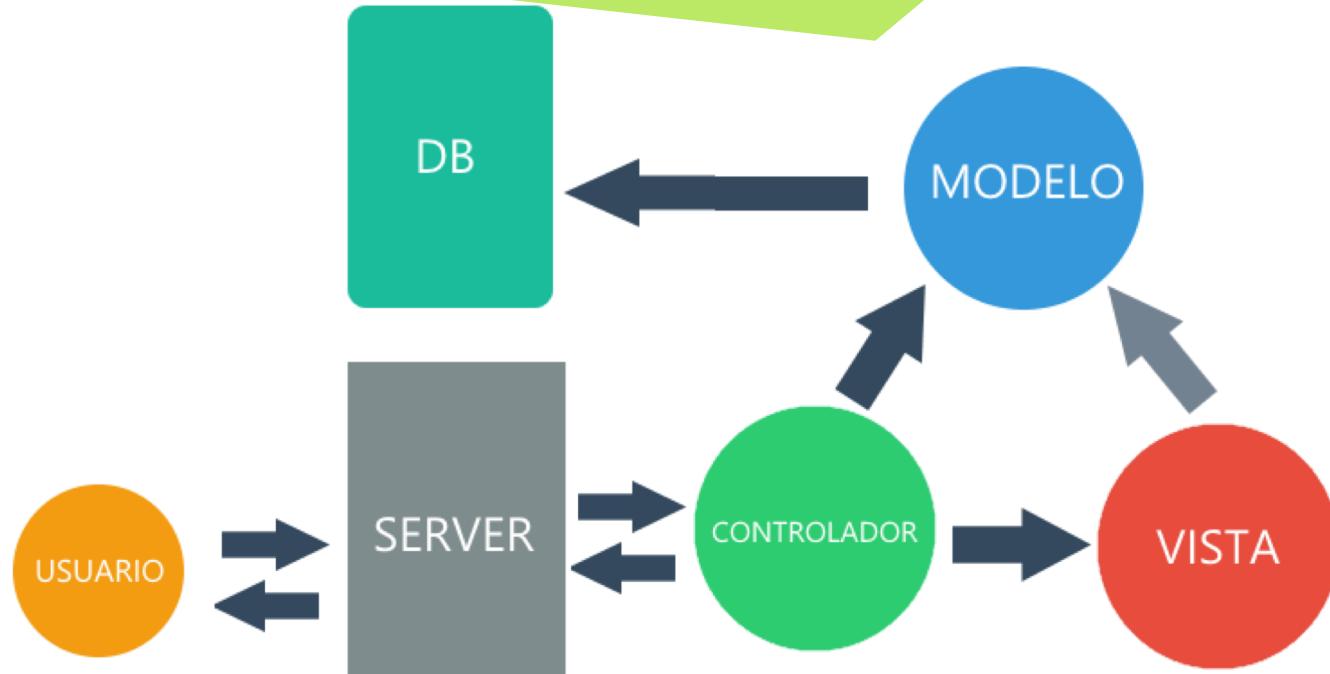
¿Qué es JEE?

Está compuesto por diferentes componentes tales como: Enterprise JavaBeans, Servlets, Portlets, JavaServer Pages y varias tecnologías de servicios web.

Java EE tiene varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc y define cómo coordinarlos



MVC



Arquitectura MVC

Patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones

- ◆ Es un patrón de diseño orientado a aplicaciones con interfaz de usuario.
- ◆ Su nombre nace de las iniciales de las palabras **Modelo Vista Controlador**.

Controlador.

- ◆ Este patrón esta orientado para aplicaciones web.

- ◆ **Modelo:** Es la capa que atiende a toda la lógica del negocio y los datos del mismo.
- ◆ **Vista:** Es la capa que controla y gestiona todo lo relativo a la presentación de los datos al usuario.
- ◆ **Controlador:** Es la capa que se encarga de gestionar la interacción entre la vista y el modelo.

¿Qué es un WAR?

- ◆ WAR es el acrónimo de Web ARchive.
- ◆ Es una especificación que nos dice cómo hay que empaquetar una Aplicación Web para que pueda ser ejecutada por los servidores de aplicaciones.
- ◆ Una Aplicación Web es el conjunto de los recursos que son necesarios para que el servidor de aplicaciones haga lo que queremos que haga.
 - ◆ En esta colección se incluyen las páginas HTML, JSP, imágenes, clases java, librerías, applets... todo lo que necesita nuestra aplicación.

Estructura del WAR

- ◆ Para que una Aplicación Web sea portable, necesitamos como mínimo que los servidores que la ejecuten encuentren todo lo necesario para que funcione, y esto sólo lo podemos conseguir si todos los servidores buscan las cosas en el mismo sitio. Hay cuatro tipos de archivos que los servidores van a buscar:
 - ◆ - Clases java
 - ◆ - Librerías
 - ◆ - Recursos html
 - ◆ - Un archivo especial, el descriptor del despliegue (Deployment Descriptor) de la aplicación

Estructura del WAR

- ◆ Cada uno de estos tipos de archivo tiene su sitio en el archivo WAR:
- ◆ /WEB-INF/classes contiene las clases java
- ◆ /WEB-INF/lib contiene las librerías java
- ◆ /WEB-INF/web.xml es el descriptor de despliegue
- ◆ ...y el resto de recursos (html, jsp, imágenes, etc) se ubican en el raíz. No hay una estructura definida para estos archivos.

Servidor Web

- ◆ Apache Tomcat, también conocido como servidor Tomcat, es un contenedor de servlets java de código abierto y proporciona un entorno web puro Java para ejecutar este código.
- ◆ Es muy famoso y ampliamente utilizado para ejecutar sitios web basados en Java en todo el mundo.
- ◆ Tomcat está disponible para varios sistemas operativos como Linux, Unix, Windows, etc



TOMCAT

- ◆ Tomcat es un contenedor web con soporte de servlets y JSPs.
- ◆ Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS.
- ◆ Incluye el compilador **Jasper**, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.
- ◆ Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

TOMCAT - Estructura de directorios

- ◆ **bin** - arranque, cierre, y otros scripts y ejecutables.
- ◆ **common** - clases comunes que pueden utilizar Catalina y las aplicaciones web.
- ◆ **conf** - ficheros XML para la configuración de Tomcat.
- ◆ **logs** - logs de Catalina y de las aplicaciones.
- ◆ **server** - clases utilizadas solamente por Catalina.
- ◆ **shared** - clases compartidas por todas las aplicaciones web.
- ◆ **webapps** - directorio que contiene las aplicaciones web.
- ◆ **work** - almacenamiento temporal de ficheros y directorios.

TOMCAT - Instalación

- ◆ Vamos a <http://tomcat.apache.org/download-80.cgi>
- ◆ Podemos descargar el instalador
- ◆ Instalamos



Tomcat

- ◆ Accedemos a <http://localhost:8080>
- ◆ Vamos a configurar el servidor en tomcat
 - ◆ Perspectiva J2EE
 - ◆ Vista Server

Creación de Proyectos Web en Eclipse

- ◆ Crear un Dynamic Web Project
- ◆ Configurar Runtime
- ◆ Configurar carpetas fuente:
 - ◆ 'src/main/java',
 - ◆ 'src/main/resources',
- ◆ Configurar carpeta web:
 - ◆ 'src/main/webapp',

Configuración como proyecto Maven

- ◆ Proyecto: click derecho
- ◆ Submenú Configure
 - ◆ Submenú **Convert to Maven Project**
- ◆ Configuramos el proyecto con los identificadores maven

Configuración como proyecto Maven

- ◆ Una vez que el proyecto se ha creado, podemos identificar la siguiente estructura:
 - ◆ Los subdirectorios 'src/main/java' y 'src/main/resources' que contendrá todos los recursos utilizados por la aplicación.
 - ◆ El subdirectorio 'src/main/webapp' que alojará todos los archivos que no sean código fuente Java, como archivos JSP y de configuración.
 - ◆ El directorio 'target' donde se generará el archivo WAR que usaremos para almacenar y desplegar rápidamente nuestra aplicación.
 - ◆ El fichero 'pom.xml' que contiene las dependencias Maven.

Primera página jsp

- ◆ En src/main/webapp creamos el archivo index.jsp con el contenido:

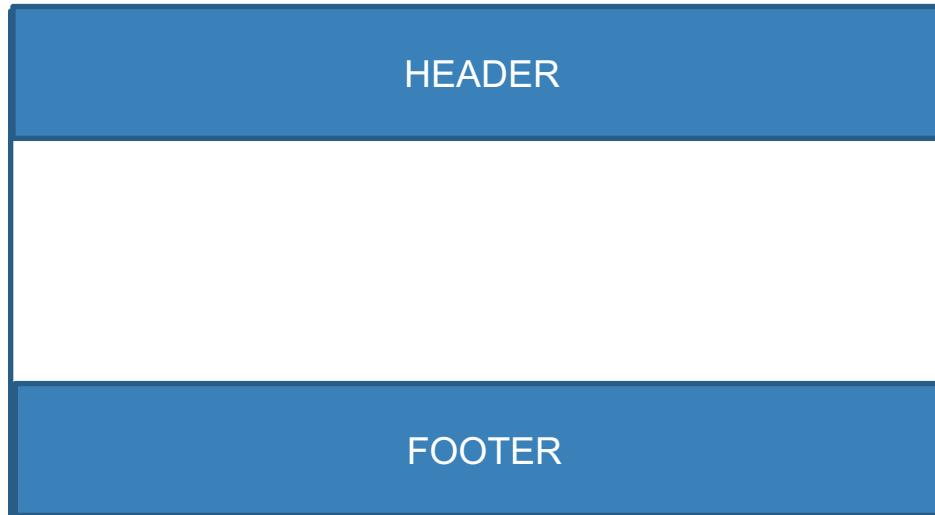
- ◆ <html>
 - ◆ <head>
 - ◆ <title> Ejemplo :: Mi primera página</title>
- ◆ </head>
- ◆ <body>
 - ◆ <h1>Ejemplo 1 – Probando contenido</h1>
 - ◆ <p>Esto es una prueba</p>
- ◆ </body>
- ◆ </html>

Desplegar la aplicación en el servidor

- ◆ Para compilar, construir y desplegar la aplicación automáticamente sólo es necesario seleccionar 'Run as > Run on Server' sobre el menú contextual que aparece cuando damos click con el botón derecho sobre el nombre del proyecto.
- ◆ Elegimos el servidor y....
- ◆ <http://localhost:8080/webdemo3/>

Práctica

- ◆ Crear una página html que tenga solo una cabecera y un pie de página:



API Rest

Antes que nada definamos conceptos:

- ◆ API: la interfaz de programación de aplicaciones, cuya abreviatura en inglés es API (Application Programming Interface),
- ◆ Es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

API Rest

Teniendo en cuenta lo anterior, definamos REST:

- ◆ REST: el término REST (Representational State Transfer) se originó en el año 2000.
- ◆ Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web respetando HTTP.

API Rest

- En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook...
- Hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible.
- Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.

API Rest

Las restricciones que definen a un sistema RESTful serían:

- ◆ **Cliente-servidor:** esta restricción mantiene al cliente y al servidor débilmente acoplados.
- ◆ Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente.

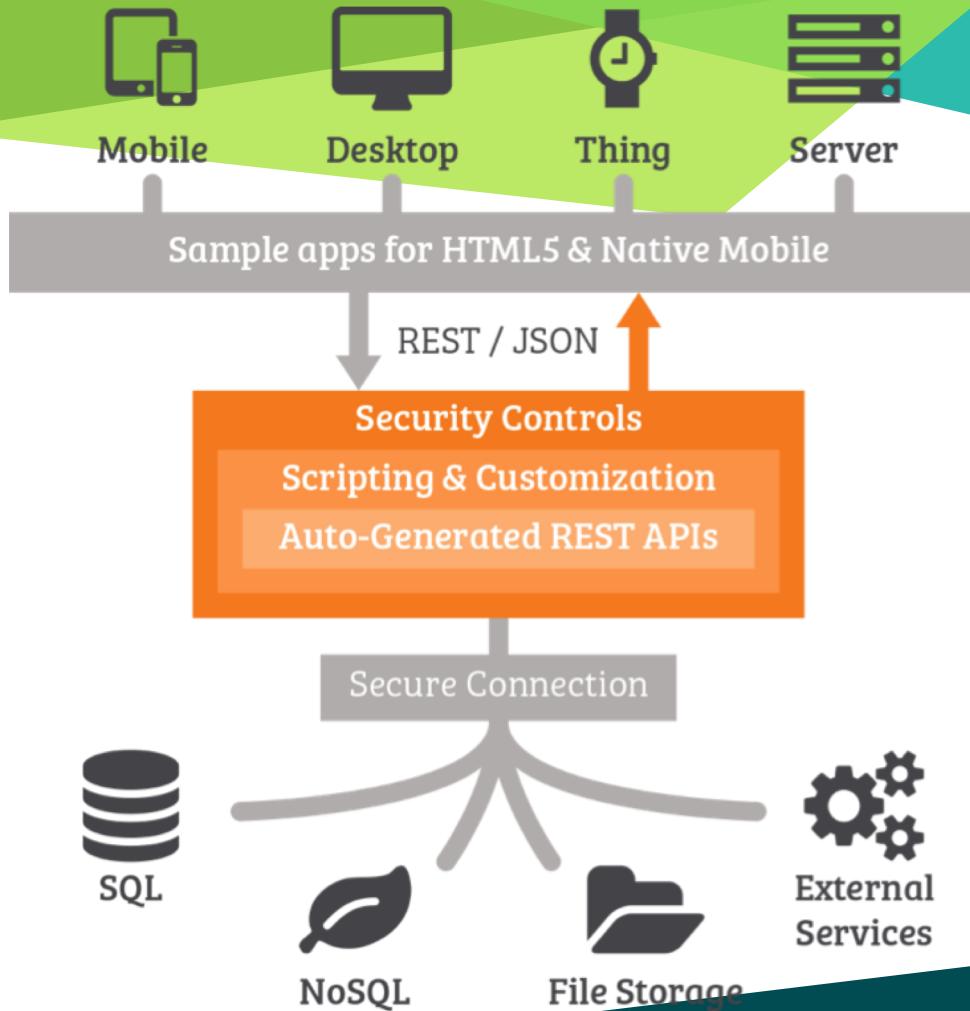
API Rest

Características de una API Rest:

- ◆ **Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar), PATCH (edición parcial) y DELETE (eliminar).**

API Rest

- ◆ Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.



API Rest

Ventajas que ofrece una API Rest al desarrollo:

- ◆ **Separación entre el cliente y el servidor:** el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

API Rest

- ◆ **Visibilidad, fiabilidad y escalabilidad.** La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

API Rest

- ◆ La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo.
- ◆ Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Terminología en un API REST



API Rest

Antes de comenzar, tengamos en cuenta algunos términos a tener presentes y bien identificados:

- ◆ **Resource o recurso:** abarca un objeto o la representación de una entidad compuesta por datos y, por tanto, manipulable a través de métodos o funciones. Por ejemplo, **usuarios** sería un resource y podemos realizar acciones sobre él como **registrar, editar o eliminar.**

API Rest

- ◆ **Colecciones:** son sets o grupos de resources, por ejemplo, podemos tener el recurso **usuario** dentro de una colección **usuarios**.
- ◆ **URL (Uniform Resource Locator):** es un path o dirección desde donde un recurso puede ser accedido y ciertas acciones aplicadas sobre él.

API Rest

Al definir y construir una API, definimos una serie de paths para acceder a propiedades y métodos que cambian recursos o entidades.

Como buena práctica, los paths o URLs desde donde accedemos a estos recursos solo deben contener el nombre del recurso, sin acciones ni verbos.

API Rest

El nombre en la URL indica a qué recurso queremos acceder, mientras que el método de la petición a esa URL indicará la acción que queremos realizar sobre el recurso. Por ejemplo:

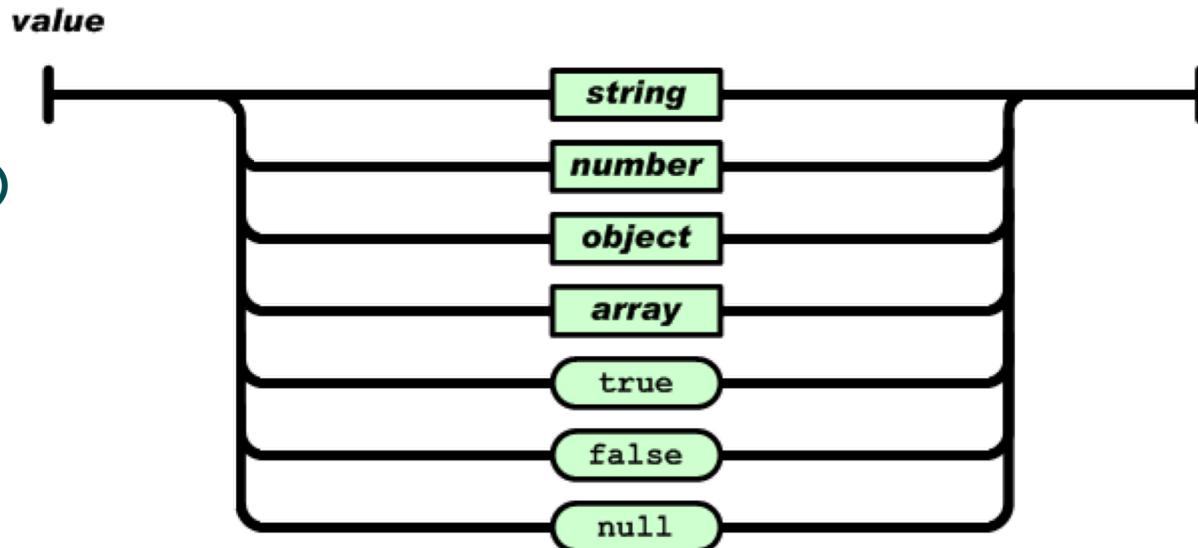
- ◆ El método **GET** sobre **/usuarios** debería listar todos los usuarios existentes

API Rest - JSON

- ◆ **JSON:** es el acrónimo para referirse a JavaScript Object Notation, o Notación de Objetos JavaScript. Un objeto en formato JSON se compone de pares clave - valor, por lo que podemos ver la relación y facilidad con la que puede representarse un modelo de datos con este formato.
Al igual que una variable, un objeto JSON puede contener propiedades de distintos tipos de datos, los tipos que admite un Objeto JavaScript (JSON) son:

API Rest - JSON

- ◆ String
- ◆ Number
- ◆ Objetos (otro JSON)
- ◆ Array
- ◆ Boolean
- ◆ Null



<http://json.org/>

API Rest - Métodos

Métodos HTTP

Teniendo en cuenta lo visto hasta ahora, podemos identificar los métodos HTTP como una parte vital en el proceso de desarrollo y construcción de una API Rest, pues son estos métodos quienes determinarán cómo actuará el servidor de acuerdo a nuestra petición.

Conozcamos los métodos HTTP principales que tenemos disponibles para nuestra API Rest.

API Rest - Métodos

◆ GET

Se usa para obtener información del servidor, en una API Rest, suele usarse para retornar datos en formato JSON. Este método solo debe usarse para obtener información del servidor de acuerdo a los estándares de HTTP. El método GET no debe cambiar el estado del servidor, es decir, no debe hacer ninguna modificación a registros, archivos, etc.

API Rest - Métodos

◆ POST

Es el encargado de crear un nuevo recurso o registro y, por consiguiente, modificar el estado del servidor. A nivel de una API Rest, en una petición POST enviamos datos en formato JSON, estos datos son leídos por el servidor y son almacenados, comúnmente, en una base de datos.

API Rest - Métodos

◆ POST

Cabe destacar que el formato JSON es estándar para la creación de API Rests, esto porque es más sencillo manipular, validar y gestionar los datos contenidos en un JSON que si estuviesen en un formato String.

Pero su ventaja no se limita al almacenamiento en el servidor, también se facilita su manipulación y muestra a nivel del cliente, pudiendo mostrar los datos de forma ordenada en pares de propiedades y valores al recibirlas, manteniendo un mínimo nivel de edición al realizar una petición.

API Rest - Métodos

◆ PUT

Se usa para actualizar por completo un registro o recurso, por lo que, al igual que el método POST, realiza cambios en el servidor, la diferencia radica en que trabaja sobre data previamente almacenada, cambiando sus propiedades, pudiendo crear un registro nuevo si el buscado para modificar no existe.

API Rest - Métodos

◆ PATCH

Se usa para modificar parcialmente un registro o recurso al igual que el método PUT, sin embargo, cuando queremos modificar solo un elemento de un registro o recurso PATCH es la opción correcta, si queremos modificar todo el registro o recurso, usaríamos PUT.

API Rest - Métodos

◆ **DELETE**

Es el método empleado para eliminar datos del servidor, por motivos de seguridad e integridad, se suele aplicar una eliminación lógica de datos usando el método PUT y una propiedad de control de estatus en vez de una eliminación física con DELETE.

De los 5, 4 métodos representan lo que se conoce como CRUD (Create, Read, Update, Delete)

Servlets

Java Servlet Technology

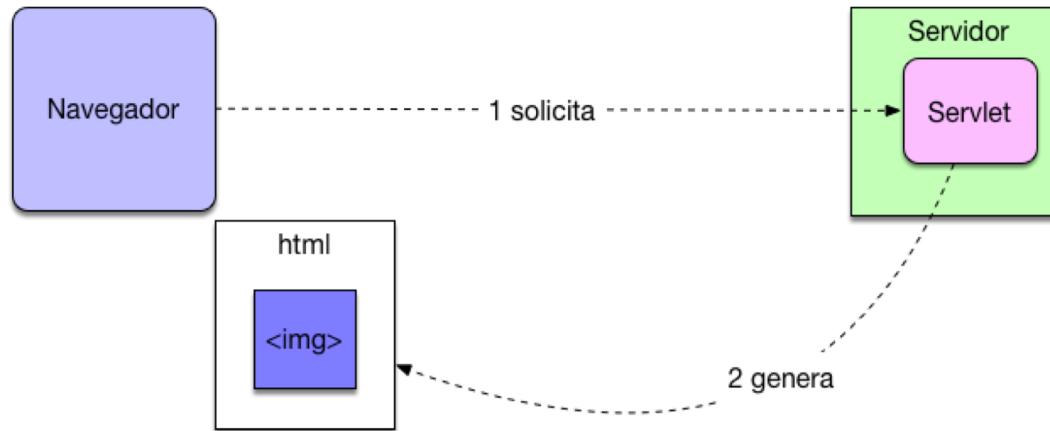
¿Qué son los Servlet?

Son componentes del estándar JEE que se ejecutan del lado del servidor tras ser invocados por un navegador web a través del protocolo HTTP. Estos componentes son capaces de recibir información a través de parámetros y retornar una salida que puede ser interpretada por un navegador.

Los servlets son el mecanismo ideal para crear aplicaciones web sencillas

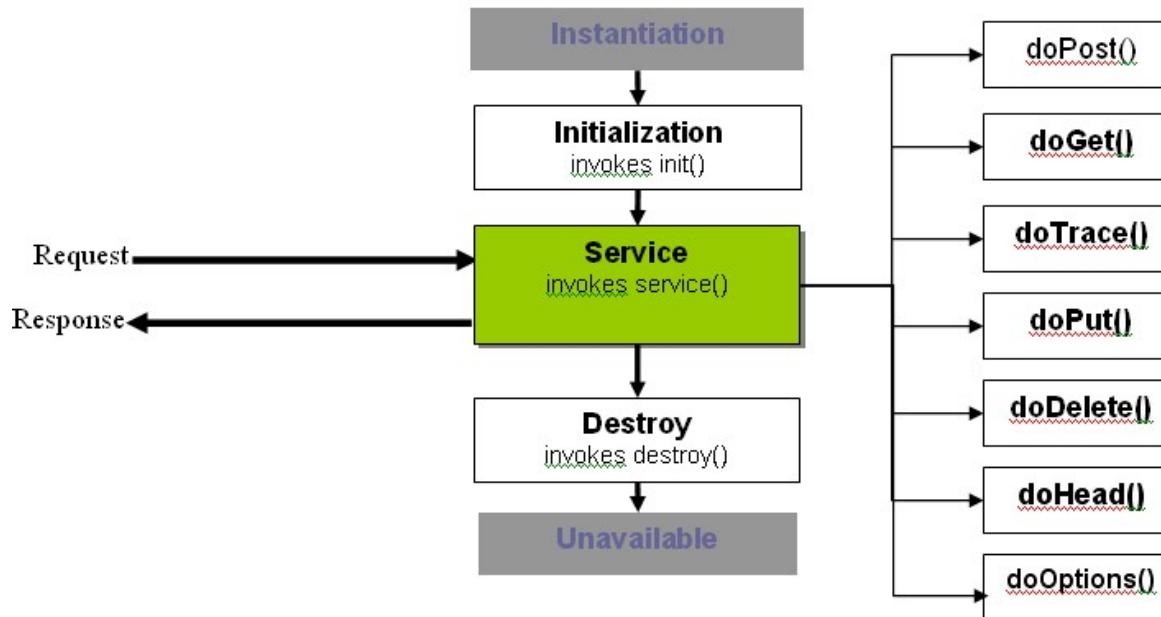
Java Servlet Technology

- ◆ Son pequeños programas escritos en Java que admiten peticiones a través del protocolo HTTP. Los servlets reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador, normalmente en HTML.



Java Servlet Technology

Ciclo de vida del Servlet



Java Servlet Technology

Ciclo de vida del Servlet

- ◆ **init:** este método se ejecuta cuando se ha creado la instancia de un servlet. Puede ejecutar tras el primer acceso a un servlet, o al enlazar su arranque con el arranque del servidor.
- ◆ **service:** este método se encarga de despachar una solicitud para ser atendida por un verbo HTTP determinado y crea un hilo de ejecución específico.
- ◆ **destroy:** este método se ejecuta en cuanto el servlet es terminado.

Métodos doGET y doPOST

- ◆ Tanto GET como POST son utilizados por el navegador para solicitar un único recurso del servidor. Cada recurso requiere una solicitud GET o POST por separado.
- ◆ El método GET es comúnmente utilizado por los navegadores (y es el método predeterminado) para recuperar información de los servidores. Al usar el método GET, la tercera sección del paquete de solicitud, que es el cuerpo de la solicitud, permanece vacía.
- ◆ El método GET se utiliza de una de estas dos maneras: cuando no se especifica ningún método, es cuando el navegador solicita un recurso simple, como una página HTML, una imagen, etc.
- ◆ Cuando se envía un formulario, y elegimos el método = GET en la etiqueta HTML.

Métodos doGET y doPOST

- ◆ Si el método GET se usa con un formulario HTML, los datos recopilados a través del formulario se envían al servidor al agregar un "?" hasta el final de la URL, y luego agregamos todos los pares nombre = valor (nombre del campo de formulario html y el valor ingresado en ese campo) separados por un "&"
- ◆ Ejemplo:
- ◆ GET /sultans/shop/form1.jsp?name= Sam% 20Sultan&iceCream = vanilla
- ◆ Los datos del formulario name = value se almacenarán en una variable de entorno llamada **QUERY_STRING**.

Métodos doGET y doPOST

- ◆ El método POST se usa cuando se crea un formulario HTML y se solicita method = POST como parte de la etiqueta.
- ◆ El método POST permite al cliente enviar datos de formulario al servidor en la sección de cuerpo de solicitud de la solicitud (como se discutió anteriormente).
- ◆ Los datos están codificados y tienen un formato similar al método GET, excepto que los datos se envían al programa a través de la entrada estándar.

Métodos doGET y doPOST

- ◆ Ventajas del método GET: Se pueden ingresar parámetros un poco más rápidos a través de un formulario o añadiéndolos después de que la página URL se pueda marcar con sus parámetros
- ◆ Desventajas del método GET: solo puede enviar 4K de datos. (No debe usarlo cuando use un campo de área de texto) Los parámetros son visibles al final de la URL
- ◆ Ventajas del método POST: los parámetros no son visibles al final de la URL. (Úselo para datos confidenciales) Puede enviar más de 4K de datos al servidor
- ◆ Desventajas del método POST: no se pueden visualizar los datos

Creación de Servlets

- ◆ Click derecho, New Servlet

Java Servlet Technology

Ejemplo

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

public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}
```

Java Servlet Technology

Mapping

```
<webapp>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
</webapp>
```

Creación de Formulario con Servlets

formulario.jsp

```
<FORM NAME="FORM1" METHOD="POST" ACTION="/miServlet">  
    Nombre: <INPUT TYPE=text NAME="NOM" SIZE="25">  
  
    <INPUT TYPE=SUBMIT VALUE="Enviar">  
</FORM>
```

Práctica

- ◆ Crear un nuevo proyecto web que contenta un formulario para la creación de Categorías para el proyecto del carrito de compra.
- ◆ Debe capturar todos los parámetros necesarios para la creación de la categoría
- ◆ Al recibir los datos, debe almacenarlos en la tabla de MySQL
- ◆ Debe retornar una vista html de la categoría creada y la data que se muestra debe ser lo que retorne de BD después de la creación

JSP



Java Server Page (JSP)

¿Qué es JSP?

Es una tecnología del stack web Java que nos permite crear páginas web dinámicas, es decir, una página con la capacidad cambiar su contenido de acuerdo a la interacción del usuario.

Podemos asemejar esta tecnología a PHP.



Java Server Page (JSP)

¿Qué es JSP?

- ◆ Es un fichero HTML o XHTML que embebe código Java que aporta generar contenido dinámico.
- ◆ Es más lento que un Servlet porque implica un proceso de transformación y conversión.
- ◆ En el patrón MVC, sería la capa de vista y los Servlets la capa controladora.



Java Server Page (JSP)

¿Cómo funcionan los JSP?

- ◆ Un usuario hace una solicitud específica a una página JSP con unos determinados parámetros.
- ◆ Si es la primera solicitud, el servidor transforma el JSP en un Servlet capaz de procesar los parámetros enviados y lo compila.
- ◆ Si el Servlet está compilado, se ejecuta el Servlet con los parámetros enviados.
- ◆ El Servlet devuelve al usuario la salida en formato HTML estático.

Java Server Page (JSP)

Variables implícitas

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

Java Server Page (JSP)

Directivas

Son instrucciones que nos permiten añadir archivos o componentes a la páginas, como también configurarla. Las directivas son las siguientes:

- ◆ **include**: permite incluir páginas (JSP o HTML) al JSP que utiliza la instrucción.

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

- ◆ Práctica: Crear nuevo jsp que incluye el header y footer creados anteriormente

Java Server Page (JSP)

Directivas

- ◆ **taglib**: permite incluir librerías que proporcionan lógica dinámica de etiquetas.

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

- ◆ **page**: permiten configurar ciertos atributos de la página.

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

Java Server Page (JSP)

Declaraciones y Scriptlets

Podemos hacer declaraciones de variables, funciones y funciones estaticas.

```
<%! int maxAlumnosClase = 30; %>
```

También podemos añadir lógica a través de los scriptlets

```
<% /** código Java */ %>
```

Java Server Page (JSP)

Expresiones

Podemos añadir expresiones que se calculen e impriman directamente en la página resultante.

```
<%= contador+1 %>
```

```
<%@ page import="java.io.* , java.util.*" %>
<% String title = "Demo de JSP"; %>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center> <h1><% out.print(title); %></h1> </center>
</body>
</html>
```

```
<%@ page import="java.io.* , java.util.*" %>
<% String title = "User Agent Example"; %>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center> <h1><% out.print(title); %></h1> </center>
<div align="center"> <p>$ {header["user-agent"]} </p> </div>
</body>
</html>
```

Parte 1

```
<%@ page errorPage="ShowError.jsp" %>
<html>
<head>
<title>Error Handling </title>
</head>
<body>
<%
int x = 1;
if (x == 1) {
    throw new RuntimeException("Error intencionado!!!!");
}
%>
</body> </html>
```

Parte 2

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>Show Error Page</title>
</head>
<body>
<h1>Opps...</h1>
<p>Sorry, an error occurred.</p>
<p>Here is the exception stack trace: </p>
<% exception.printStackTrace(response.getWriter()); %>
```

Practica 2

- ◆ Añadir a la práctica anterior el header y footer

JSP lenguaje de expresión

- ◆ EL (Expression Language) es un lenguaje utilizado en las páginas JSP para interactuar con los datos (JavaBeans) servidos (Servlet) por parte del servidor, sin importar el alcance de los atributos (request, session, application).
- ◆ Combinado con la librería JSTL Core (JavaServer Pages Standard Tag Library) nos permite construir toda la lógica de las páginas JSP de una forma mucho más versátil.
- ◆ Para utilizarse no es necesario hacer ninguna declaración especial en los JSP. El único requisito es que sea un archivo JSP válido (e.j. page.jsp).

JavaServer Pages Standard Tag Library

¿Qué son los JSTL?

Son unos componentes que extienden los JSP proporcionando cuatro librerías de etiquetas con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas.

Las cuatro librerías son las siguientes:

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

JavaServer Pages Standard Tag Library

¿Qué son los JSTL?

- ◆ **<c:out>**: Es utilizado para devolver los valores al cliente, escapando los tags HTML o XML.
- ◆ **<c:set>**: Se utiliza para evaluar una expresión que asignará un valor a una variable.
- ◆ **<c:remove>**: Se utiliza para eliminar un atributo del contexto que se indique.
- ◆ **<c: if>**: Se utiliza para evaluar una expresión condicional positiva.

JavaServer Pages Standard Tag Library

¿Qué son los JSTL?

- ◆ **<c:choose>**: Representa a la sentencia switch de Java.
- ◆ **<c:when>**: Representa a la sentencia case que van dentro de la sentencia switch.
- ◆ **<c:otherwise>**: Representa a la sentencia default que va dentro de la sentencia switch.
- ◆ **<c:catch>**: Captura las excepciones que se produzcan den el JSP.

JavaServer Pages Standard Tag Library

¿Qué son los JSTL?

- ◆ **<c:import>**: Permite importar el contenido de un fichero HTML o JSP, y comparte el contexto.
- ◆ **<c:forEach>**: Permite ejecutar un determinado finito de veces, el contenido que encierra entre esta etiqueta.
- ◆ **<c:forTokens>**: Permite iterar una cadena en que cada item esta separado por un token.
- ◆ **<c:param>**: Permite añadir parámetros a los tags **<c:url>** y **<c:redirect>**.

JavaServer Pages Standard Tag Library

¿Qué son los JSTL?

- ◆ **<c:url>**: Permite formatear una URL relativa y añadir los parámetros, y transformarla a una URL absoluta del contexto de la aplicación.
- ◆ **<c:redirect>**: Permite crear una URL para una redirección y le añade parámetros.

Sesiones

- ◆ Independientemente de la técnica utilizada, lo que está claro es que hay almacenar los datos de la sesión en algún sitio.
- ◆ Una buena opción es el objeto **HttpSession** que puede almacenar los datos de la sesión en el servidor.
- ◆ Para utilizar este objeto es necesario tener un objeto de la sesión, leerlo, escribirlo y eliminarlo cuando se cierre la sesión o después de un cierto tiempo si la sesión no se ha cerrado expresamente.

HttpSession

- ◆ El funcionamiento del sistema de sesiones es relativamente sencillo. **Cada vez que un usuario crea una session accediendo a una página (que la genere) se crea un objeto a nivel de Servidor con un HashMap vacío que nos permite almacenar la información que necesitamos relativa a este usuario.**
- ◆ Realizado este primer paso se envía al navegador del usuario una Cookie que sirve para identificarle y asociarle el HashMap que se acaba de construir para que pueda almacenar información en él.
- ◆ Este HashMap puede ser accedido desde cualquier otra página permitiéndonos compartir información.

Sesiones

- ◆ La persistencia del objeto es válida en el contexto de la aplicación Web, por lo que puede ser compartida por varios servlets.
- ◆ Un servlet puede acceder a objetos almacenados en otro servlet.
- ◆ Estos objetos, también llamados atributos, pueden estar disponibles para otros servlets dentro del ámbito de petición, sesión o aplicación.
- ◆ El objeto **HttpSession** almacena y accede a la información de la sesión que controla el servlet, por lo que no es necesario que esta información sea manipulada en código.

HttpSession

- ◆ Los métodos más útiles de esta clase son:
 - ◆ getId(), devuelve una cadena conteniendo el identificador único asignado a la sesión. Es el utilizado en la reescritura de URL para identificar a esa sesión.
 - ◆ isNew(), devuelve true si el cliente no ha establecido una sesión. Si el cliente tiene deshabilitadas las cookies, la sesión será nueva en cada petición.
 - ◆ setAttribute(), asigna un objeto a la sesión, utilizando un nombre determinado.

HttpSession

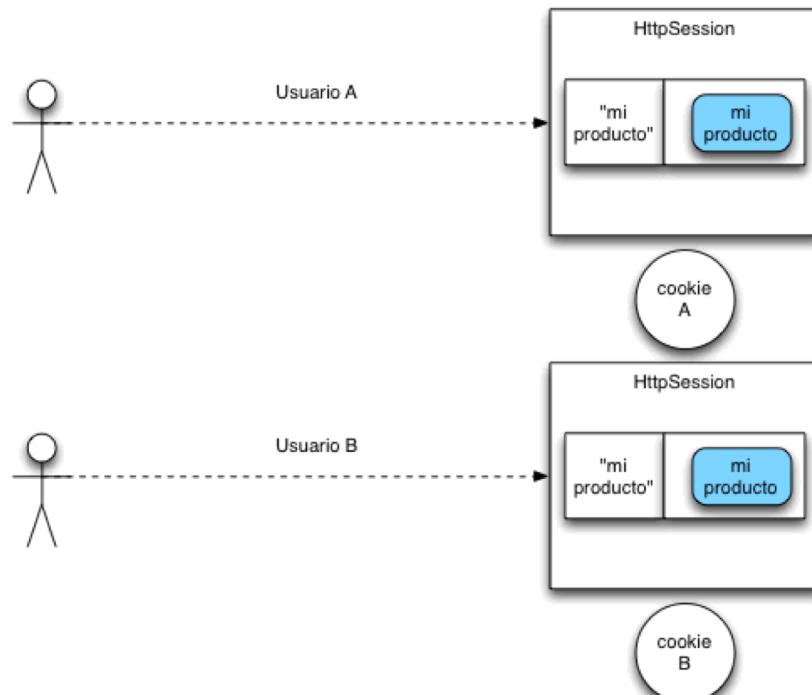
- ◆ Los métodos más útiles de esta clase son:
 - ◆ `getAttribute()`, devuelve el objeto asignado a la sesión, identificado por el nombre que le indique.
 - ◆ `setInactiveInterval()`, especifica el tiempo máximo que ha de pasar entre peticiones consecutivas del cliente para que la sesión se dé como no válida. Un número negativo como argumento hará que nunca se invalide la sesión.
 - ◆ `invalidate()`, elimina la sesión actual y libera todos los objetos asociados a ella.

HttpSession

- ◆ Tiene una estructura de **HashMap (Diccionario)** y permite almacenar cualquier tipo de objeto en ella de tal forma que pueda ser compartido por las diferentes páginas que como usuarios utilizamos.

Usuarios y Sessiones

- ◆ El concepto de Session es individual de cada usuario que se conecta a nuestra aplicación y la información no es compartida entre ellos. Así pues cada usuario dispondrá de su propio HashMap en donde almacenar la información que resulte útil entre páginas.



Práctica

- ◆ Crear una página con un formulario de login con los campos:
 - ◆ Login
 - ◆ Password
- ◆ Debe recibirlos un Servlet y redireccionar al home de creación de categorías y que muestre el nombre de usuario como saludo en el header
- ◆ Bienvenido: [usuario]

web.xml

- ◆ El descriptor de aplicaciones web proporciona al servidor de aplicaciones información sobre los recursos web de la aplicación. Este archivo siempre se denomina web.xml.
- ◆ El archivo web.xml debe residir en el directorio WEB-INF bajo el contexto de la jerarquía de directorios que existe para una aplicación web.

web.xml

- ◆ Este archivo contiene:
 - ◆ Parámetros de inicialización
 - ◆ Configuración de la sesión.
 - ◆ ~~Definiciones de Servlets/JavaServer Pages.~~
 - ◆ ~~Mapeado Servlets/JavaServer Pages.~~
 - ◆ Mapeado de tipos MIME.
 - ◆ Seguridad.
 - ◆ Páginas de error, de bienvenida,...

web.xml

```
◆ <!DOCTYPE web-app PUBLIC  
◆   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
◆   "http://java.sun.com/dtd/web-app_2_3.dtd" >  
◆  
  
◆ <web-app>  
◆   <display-name>Archetype Created Web Application</display-  
name>  
◆  
◆   <context-param>  
◆     <param-name>pais</param-name>  
◆     <param-value>España</param-value>  
◆   </context-param>  
◆  
◆ </web-app>
```

web.xml

- ◆ <error-page>
- ◆ <exception-type>java.lang.Exception</exception-type>
- ◆ <location>/error.jsp</location>
- ◆ </error-page>

Cookies

- ◆ Las cookies son pequeños bits de información textual, que un servidor web (o un contenedor de aplicaciones tal como Tomcat) envía a un navegador cliente para identificarlo
- ◆ El navegador luego devuelve en su petición esa cookie (o cookies) cuando visita nuevamente ese sitio web. De esta manera el servidor, a través de una página jsp o servlet, lee las cookies que le envió previamente en una conexión anterior.

Cookies

- ◆ Crear una cookie es bastante fácil cuando utilizamos la clase **Cookie** que encontramos en el paquete javax.servlet.http, simplemente hacemos una instancia de esta clase y la mandamos en la respuesta del servidor con el método addCookie
- ◆

```
private static final void doGet(HttpServletRequest request,
HttpServletResponse response) {
    Cookie galletaColor = new Cookie("color", "rojo");
    response.addCookie(galletaColor);
}
```

Cookies

- ◆ Una cookie puede tener los siguientes parámetros:
 - ◆ **Dominio:** Establece el nombre de domino (o dominios) para el cual una cookie es valida. Se establece a través del método `setDomain(String dominio)` de la clase `Cookie`
 - ◆ **Path:** Establece la uri especifica en la que la cookie será valida (e.g: `/admin`) es útil si no queremos exponer la cookie en toda la aplicación. Se establece como es de esperarse con el metodo `setPath(String path)` de la clase `Cookie`
 - ◆ **Expiración:** Indica cuanto debe durar una cookie en el navegador. Este valor se establece en segundos a través del método `setMaxAge(int segundos)` de la clase `Cookie`

Cookies

- ◆ **HostOnly:** Esta propiedad es muy importante en términos de seguridad, ya que establece si una cookie podrá ser leída por el cliente o solo por el servidor. Piensen que si alguien logra un ataque bien podrían robar las cookies de sus usuarios sin mayores dificultades. Por eso `HostOnly` debería ser `true` siempre que el dominio desde el cual se está intentando leer no sea igual al dominio de origen establecido.
- ◆ **Secure:** Otra parámetro muy importante que indica que la cookie en cuestión puede ser leída únicamente a través de un protocolo seguro HTTPS o SSL. Y se establece con el método `setSecure(boolean isSecure)` de la clase `Cookie`.
- ◆ **HttpOnly:** Es similar a `HostOnly` excepto que esta si se puede manipular. Funciona como una medida de seguridad extra a la especificación establecida

Cookies

```
Cookie miCookie = new Cookie("idCliente","dato") ;  
  
// hacemos que nuestra cookie tenga sentido durante un día  
miCookie.setMaxAge(60*60*24) ;  
  
response.addCookie(miCookie) ;
```

```
Cookie [] cookies = peticion.getCookies();

for(int i=0; i<cookies.length; i++)
{
    Cookie cookieActual = cookies[i];

    String identificador = cookieActual.getName();
    String valor = cookieActual.getValue();

    If(identificador.equals("idCliente"))
    {
        // alguna operacion
    }
}
```

Práctica

- ◆ Modificar la práctica anterior para que, además de los datos de login, muestre un desplegable para elegir el idioma (deben listarse los idiomas español e inglés).
- ◆ El idioma debe guardarse en una cookie y, en el footer debe mostrar el idioma seleccionado:
 - ◆ Idioma: [idioma]

Investigación:

- ◆ Investigar cómo se puede:
 - ◆ Buscar una cookie determinada
 - ◆ Modificar el valor de la cookie

Spring Core, AOP

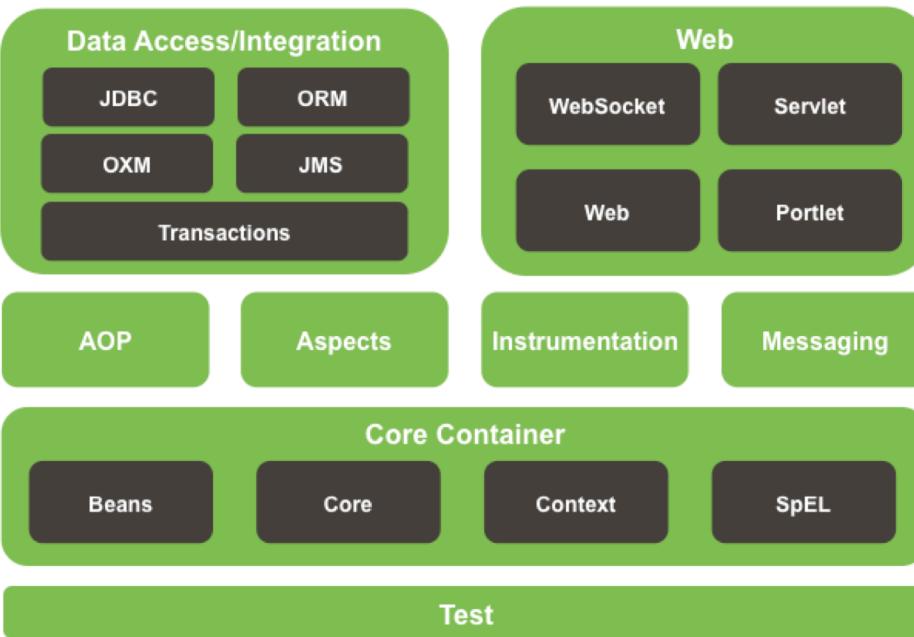


¿Qué es Spring Framework?

- ◆ Es una plataforma que proporciona un soporte integral de las diferentes infraestructuras que permiten el desarrollo de aplicaciones Java.
- ◆ Spring nos permite construir aplicaciones haciendo uso de objetos planos Java (POJO).
- ◆ Spring puede aplicarse para desarrollar aplicaciones bajo el modelo JSE o JEE.



Spring Framework Runtime



Arquitectura de Spring

- ◆ **Contenedor de IoC:** Permite la configuración de los componentes de la aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias.
- ◆ **Programación orientada a aspectos:** habilita la implementación de rutinas transversales.
- ◆ **Acceso a datos:** se trabaja con RDBMS usando JDBC o motores ORM. También se está proporcionando soporte ORM a base de datos No SQL.

Arquitectura de Spring

- ◆ **Gestión de transacciones:** unifica las distintas APIs de gestión de transacciones y coordina las transacciones de los objetos Java.
- ◆ **MVC:** es un framework basado en HTTP y servlets, que provee herramientas para el desarrollo de aplicaciones web y servicios REST.
- ◆ **Convención sobre Configuración:** el módulo Spring Roo ofrece una solución rápida para el desarrollo de aplicaciones basadas en Spring Framework, dando prioridad a la simplicidad sin perder flexibilidad.

Arquitectura de Spring

- ◆ **Procesamiento por lotes:** es un framework capaz de procesar grandes volúmenes de datos. Con capacidades para evaluar el rendimiento del proceso en lote y la administración de recursos.
- ◆ **Autenticación and Autorización:** procesos de seguridad configurables que soportan un rango de estándares, protocolos, herramientas y prácticas a través del subproyecto Spring Security

Arquitectura de Spring

- ◆ El contenedor Spring se puede configurar mediante XML, anotaciones o programáticamente.
- ◆ Para personalizar este contenedor, debemos:
 - ◆ Crear los beans.
 - ◆ Configurar los servicios de usara cada bean.

IoC: Inversion of Control

- ◆ Es un principio en el que el control de ejecución del programa deja de ser de si mismo y pasa a ser de la librería que implanta el IoC.
- ◆ ¿Cómo utilizamos IoC?
 - ◆ Construimos nuestra aplicación.
 - ◆ Configuramos cada componente de nuestra aplicación.
 - ◆ El motor Spring se encarga de instanciar y ejecutar los componentes de acuerdo a la configuración.

DI: Dependency Injection

- ◆ Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto.
- ◆ ¿Cómo utilizamos el DI?
 - ◆ Construimos los componentes de la aplicación.
 - ◆ Configuramos cada dependencia del componente.
 - ◆ El motor Spring instancia cada componente proporcionandole su respectiva dependencia.

DI: Dependency Injection

◆ Ventajas

- ◆ Reduce el código pegamento, es decir, minimiza la cantidad de código que usamos escribir para unir distintos componentes.
- ◆ Facilita la evolución de dependencias sin recompilar el código del proyecto.

DI: Dependency Injection

- ◆ Cada componentes debe declarar sus dependencias, pero no se encargara de conseguirlas.
- ◆ El contenedor Spring es el encargado de resolver todas las dependencias configuradas dentro de cada bean.

DI: Dependency Injection

- ◆ - Aunque algunas veces este código puede ser tan simple como usar el operador “new” para instanciar un nuevo objeto, otras puede ser más complejo, como realizar una búsqueda de dicha dependencia en un repositorio a través de JNDI, como en el caso de los recursos remotos.
- ◆ - Como es posible colocar la configuración de dependencias en archivos XML podemos realizar una reconfiguración fácilmente, sin necesidad de recompilar nuestro código. Gracias a esto es posible realizar el cambio de la implementación de una dependencia a otra (como en el ejemplo de Hibernate que mencioné antes)

ApplicationContext

- ◆ El ApplicationContext es una interfaz que proporciona toda la funcionalidad del contenedor Spring.
- ◆ Entre las implementaciones más utilizadas están:
 - ◆ **ClassPathXmlApplicationContext**
 - ◆ **FileSystemXmlApplicationContext**
 - ◆ **WebApplicationContext**

Bean Configuration

- ◆ La columna vertebral de todo proyecto Spring es la configuración de sus beans.
- ◆ Una de las formas de configuración disponibles en Spring es XML
- ◆ Otra es mediante anotaciones.

XML Bean Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- A simple bean definition -->
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- A bean definition with lazy init set on -->
    <bean id="..." class="..." lazy-init="true">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- A bean definition with initialization method -->
    <bean id="..." class="..." init-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- A bean definition with destruction method -->
    <bean id="..." class="..." destroy-method="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- more bean definitions go here -->
</beans>
```

Annotations

- ◆ El documento de configuración en XML suele ser complicado de hacer y también de mantener.
- ◆ Requiere conocer en gran detalle cuales son los XML Schemas de los módulos de Spring que deseamos utilizar.
- ◆ Si se modulariza la configuración es necesario controlar las importaciones de los documentos de configuración.
- ◆ La configuración por anotaciones suele añadir a las clases Java metainformación que aporta funcionalidad que el contexto de la aplicación utiliza como configuración.

Annotations

- ◆ **@Required:** esta anotación es usada para indicar que una propiedad debe tener un valor obligatorio.
- ◆ Esta anotación debe ser utilizada en métodos setter.

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    @Required  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
    // ...  
}
```

Annotations

- ◆ **@Autowired:** esta anotación inyecta dependencias a través de atributos, setter o constructores.
- ◆ Anotación @Autowired dentro de atributos.

```
import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {
    @Autowired
    private SpellChecker spellChecker;

    public TextEditor() {}

    public SpellChecker getSpellChecker(){
        return spellChecker;
    }
    public void spellCheck(){
        spellChecker.checkSpelling();
    }
}
```

◆ Anotación @Autowired en constructores.

```
import org.springframework.beans.factory.annotation.Autowired;  
  
public class TextEditor {  
  
    private SpellChecker spellChecker;  
  
    @Autowired  
    public TextEditor(SpellChecker spellChecker){  
        System.out.println("Inside TextEditor constructor." );  
        this.spellChecker = spellChecker;  
    }  
  
    public void spellCheck(){  
        spellChecker.checkSpelling();  
    }  
}
```

◆ Anotación @Autowired en setters.

```
import org.springframework.beans.factory.annotation.Autowired;  
  
public class TextEditor {  
  
    private SpellChecker spellChecker;  
  
    @Autowired  
    public void setSpellChecker( SpellChecker spellChecker ){  
        this.spellChecker = spellChecker;  
    }  
  
    public SpellChecker getSpellChecker( ) {  
        return spellChecker;  
    }  
  
    public void spellCheck() {  
        spellChecker.checkSpelling();  
    }  
}
```

Aspect Oriented Programming (AOP)

- ◆ Es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar la separación de conceptos.
- ◆ Un aspecto es la modularización de una preocupación que atraviesa múltiples clases. Los logs pueden ser un ejemplo de esta preocupación transversal.



Aspect Oriented Programming (AOP)

- ◆ Es, conceptualmente, algo que se utiliza repetidas veces y en lugares diferentes del código fuente de una aplicación, y que se encapsula y aísla del resto del código para, de esta forma, poder modificarlo sin que ello afecte al resto del código fuente.
- ◆ De esta forma, un aspecto puede ser llamado desde cualquier parte del código sin requerir el conocimiento de cómo funciona internamente por parte del programador.
- ◆ AOP también facilita el trabajo de un equipo de desarrolladores, ya que unos no deben necesariamente conocer los detalles de cómo funciona internamente el código fuente creado por los demás.

Entendiendo AOP

- ◆ Para entender la AOP pensemos en una clase con una lógica de negocios simple:
- ◆

```
public class Aerolinea {  
    public Vuelos[] consultarVuelos(String o, String d) {  
        // aquí el Código de obtener vuelos  
    }  
}
```
- ◆ Podemos querer tener un aspecto que realice un log de mensajes siempre que consultarVuelos termine:

- ◆ public class consultarVuelosAfterReturnAspect {
 - ◆
 - ◆ public void after() throws Throwable {
 - ◆ System.out.println("al retornar de la operacion");
 - ◆ }
 - ◆ }
- ◆ Acabamos de definir una simple clase Java que tiene un método llamado after que toma un argumento del tipo Object y simplemente registra ese valor.

Términos comunes

Concepto	Descripción
Aspect	Es la funcionalidad que se cruza a lo largo de la aplicación que se va a implementar de forma modular y separada del resto de sistema.
Jointpoint	Es un punto de ejecución dentro del sistema donde el aspecto es conectado.
Advice	Un advice/consejo es una acción tomada por un aspecto en un Joinpoint en particular. Los diferentes tipos de asesoramiento incluyen "around", "before" y "after". En spring, un advice es modelado como un interceptor
Pointcut	Define los advice que se aplicaran a cada punto de cruce.

Tipos de aspectos

Concepto	Descripción
Before	El aspecto se ejecuta antes de ejecutarse el método que lo provoca.
After	El aspecto se ejecuta después de ejecutarse el método que lo provoca, independiente del éxito o fracaso del método.
After-running	El aspecto se ejecuta después de que el método que lo provoca retorne un resultado.
After-throwing	El aspecto se ejecuta después de devolver una excepción el método que lo provoca.
Around	Este aspecto puede realizar acciones antes y después de invocar el método interceptado.

Implementando Aspectos

- ◆ Para trabajar con aspectos debemos añadir la dependencia en el pom.xml
 - ◆ <dependency>
 - ◆ <groupId>org.springframework.boot</groupId>
 - ◆ <artifactId>spring-boot-starter-aop</artifactId>
 - ◆ </dependency>

Clase de Configuración

- ◆ Necesitamos crear una nueva clase para configurar el uso de aspectos.
 - ◆ Creamos es.indra.demo.aspects y dentro la clase del aspecto LogServiceAspect
 - ◆ Esta clase se decora con las anotaciones Aspect y Configuration
-
- ◆ @Aspect
 - ◆ @Configuration
 - ◆ **public class LogServiceAspect {**

Método con el Advice

- ◆ Vamos a crear un método que imprima un log siempre que se ejecute algún método del paquete es.indra.demo.service
- ◆ @Before("execution(* es.indra.demo.service.impl.*.*(..))")
- ◆ **public void** before(JoinPoint joinPoint) {
- ◆ //Advice
- ◆ System.out.println("log de la operación: " + joinPoint);
- ◆ }

Práctica

- ◆ Parte 1: Crear un aspecto de tipo around para los métodos de la clase controller que tome el tiempo que tarda en ejecutarse la petición rest
- ◆ Parte 2: mejorar el ejercicio anterior para tener una anotación propia que tome el tiempo de ejecución de todo método que la tenga

Spring MVC



Spring MVC

- ◆ **Spring** tiene sus propios módulos que conforman un framework para el desarrollo de aplicaciones Java basadas en web, Este framework sigue el patrón de diseño MVC. A este conjunto de módulos se le conoce con el nombre de **Spring MVC**.
- ◆ **Spring MVC** es un framework de capa de presentación, basado o dirigido por peticiones. Esto quiere decir que los programadores estructuramos el diseño de la aplicación web en términos de **métodos que manejan peticiones HTTP individuales**

- ◆ El framework define una serie de interfaces que siguen el patrón de diseño Strategy para todas las responsabilidades que deben ser manejadas por el framework. El objetivo de cada interface es ser simple y clara, para que sea fácil para los usuarios de **Spring MVC** (o sea, nosotros) crear nuestras propias implementaciones.
- ◆ La pieza central de **Spring MVC** es un componente llamado el "**DispatcherServlet**", el cual sigue el patrón de diseño front controller, este envía las peticiones a los componentes designados para manejarlas.

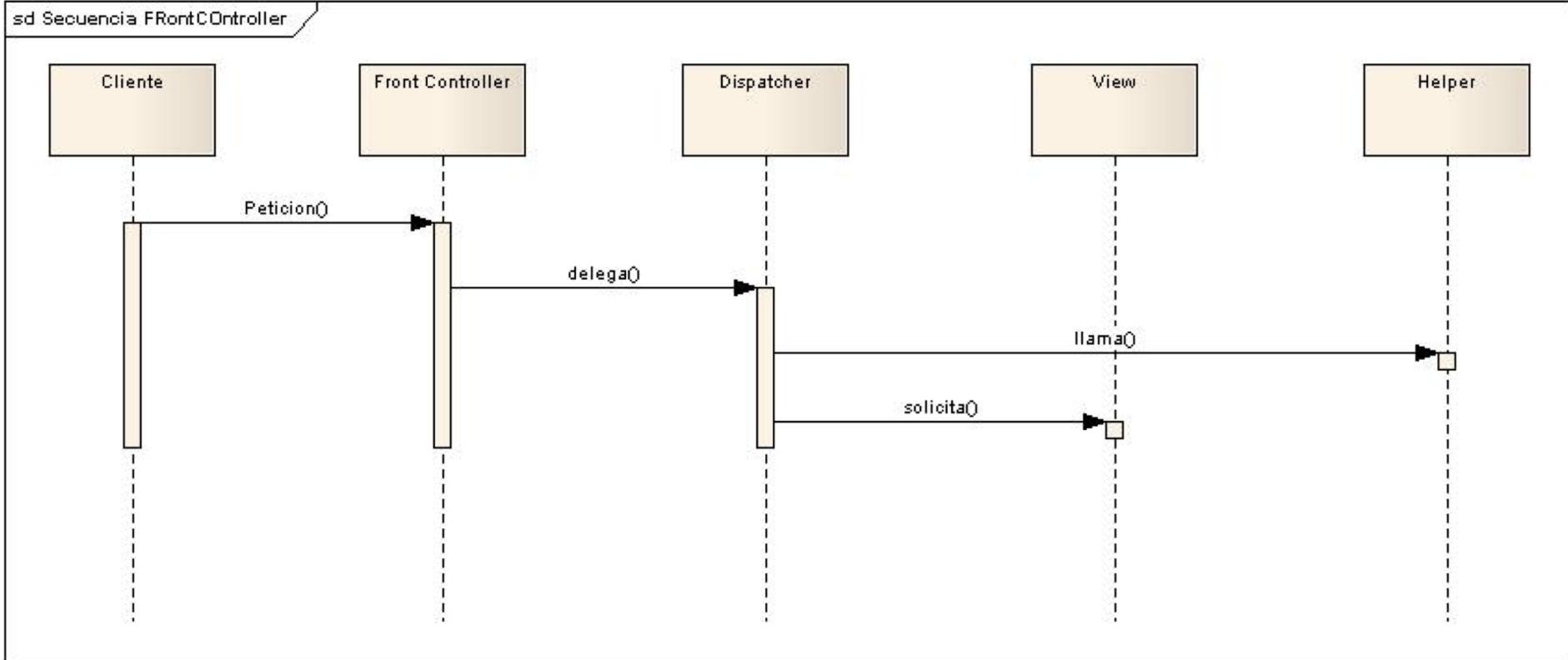
Front Controller

- ◆ Es un patrón de diseño que se basa en usar un controlador como punto inicial para la gestión de las peticiones. El controlador gestiona estas peticiones, y realiza algunas funciones como: comprobación de restricciones de seguridad, manejo de errores, mapear y delegación de las peticiones a otros componentes de la aplicación que se encargarán de generar la vista adecuada para el usuario.
- ◆ Se divide en 2 diferentes objetos el Front Controller y el Dispatcher. En ese caso, El Front Controller acepta todos los requerimientos de un cliente y realiza la autenticación, y el Dispatcher direcciona los requerimientos a manejadores apropiada.

Front Controller – Ventajas y Desventajas

- ◆ Tenemos centralizado en un único punto la gestión de las peticiones.
 - ◆ Aumentamos la reusabilidad de código.
 - ◆ Mejoramos la gestión de la seguridad.
-
- ◆ La velocidad de respuesta disminuye al tener que ser procesadas las peticiones primero por el controlador.

Front Controller



Front Controller

◆ Controlador

- ◆ El controlador es el punto de contacto inicial para manejar todos los pedidos en el sistema. El controlador puede delegar a un asistente para completar la autenticación y autorización de un usuario o para iniciar la recuperación de contacto.

◆ Dispatcher

- ◆ Un despachador es responsable de la gestión de vista y la navegación, controlando la elección de la siguiente vista a presentar al usuario, y proporcionar el mecanismo para el control a este recurso.

Front Controller

◆ Helpers

- ◆ Un ayudante se encarga de ayudar a la vista o al controlador a completar su procesamiento. Por lo tanto, los ayudantes tienen numerosas responsabilidades, incluyendo la recopilación de datos requeridos por la vista y el almacenamiento en el modelo intermedio, en cuyo caso la ayuda se conoce como un value bean.
- ◆ Además, los helpers podrán adaptar este modelo de datos para el uso de la vista.
- ◆ Los helpers pueden servir peticiones de datos desde la vista simplemente proporcionando acceso a los datos en bruto o formato a los datos como el contenido Web.

Spring MVC

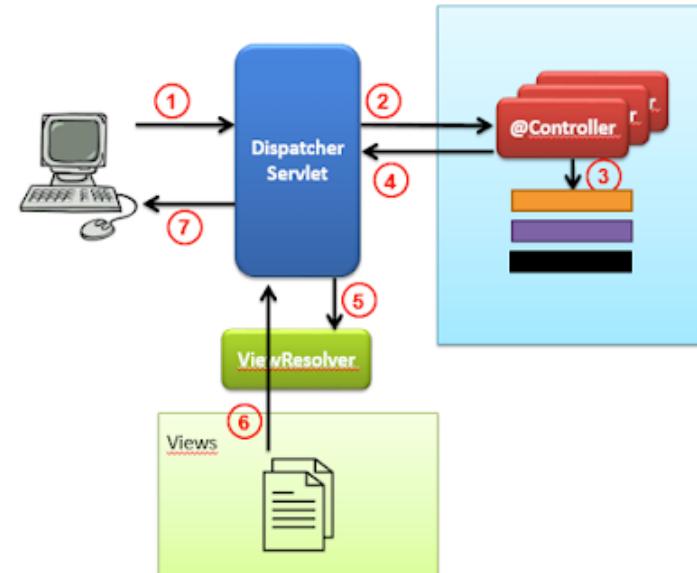
- ◆ Un principio clave en **Spring MVC** es el principio de diseño "**Abierto / Cerrado**" o, por su nombre en inglés, "**Open for extension, closed for modification**".
- ◆ Algunos métodos en las clases core de **Spring MVC** están marcados como **final** para que los desarrolladores no podamos sobre-escribirlos y proporcionar nuestro propio comportamiento, de esta forma el framework garantiza que su comportamiento básico no puede ser modificado.

DispatcherServlet

- ◆ es la pieza central de **Spring MVC**. **Spring MVC** está, como muchos otros frameworks orientado a peticiones, diseñado alrededor de un **Servlet** central que despacha las peticiones a los controladores y ofrece otra funcionalidad que facilita el desarrollo de aplicaciones web.
- ◆ Sin embargo, el "**DispatcherServlet**" hace más que sólo eso, se encuentra completamente integrado con el contenedor de **IoC** de **Spring**, lo cual nos permite usar el resto de características de **Spring**.

DispatcherServlet

1. El cliente hace una petición a la aplicación web, esta petición llega al **DispatcherServlet**.
2. El **DispatcherServlet** determina qué componente debe atender la petición y la envía a este.
3. El **Controller** implementa la lógica específica para responder la petición, para lo que puede hacer uso de cualquier recurso que esté al alcance de cualquier aplicación Java (incluyendo conexiones con servicios web o con base de datos).
4. Una vez que el **Controller** termina su proceso, regresa la petición al **DispatcherServlet**, estableciendo los datos adecuados del modelo e indicando el nombre lógico de la vista que debe regresarse al cliente y un modelo lleno con los nombres y valores de los atributos que se usarán para generar la vista final.
5. En base al nombre lógico regresado por el **Controller**, el **DispatcherServlet** usa un **ViewResolver** para determinar qué recurso debe utilizar para generar la vista final que se mostrará al usuario, este recurso puede ser una **JSP**, una página **HTML**, un template de **Velocity**, un archivo de **Excel**, un **PDF**, etc.
6. El **DispatcherServlet** obtiene la vista que será regresada al cliente.
7. El **DispatcherServlet** finalmente regresa la vista adecuada al cliente.



Hibernate



ORM

- ◆ ORM, del inglés **Object-Relational Mapper** o "Mapeador" de relacional a objetos (y viceversa). Un ORM es una biblioteca especializada en acceso a datos que genera por ti todo lo necesario para conseguir esa abstracción de la que hemos hablado.
- ◆ Gracias a un ORM ya no necesitamos utilizar SQL directamente, ni pensar en tablas ni en claves foráneas. Pensamos en objetos
- ◆ El ORM puede **generar clases a partir de las tablas** de una base de datos y sus relaciones, **o hacer justo lo contrario**: partiendo de una jerarquía de clases crear de manera transparente las entidades necesarias en una base de datos, ocupándose de todo

ORMs

- ◆ En el mundo **Java** el ORM más conocido y utilizado es **Hibernate** (<http://hibernate.org/>) que pertenece a Red Hat aunque es gratuito y Open Source. Hay muchos otros como **Jooq**, **ActiveJDBC** que trata de emular los Active Records de Ruby On Rails, o **QueryDSL**, pero en realidad ninguno llega ni por asomo al nivel de uso de Hibernate.

Ventajas e inconvenientes de un ORM

- ◆ Nos permiten aumentar la reutilización del código y mejorar el mantenimiento del mismo, ya que tenemos un único modelo en un único lugar, que podemos reutilizar en toda la aplicación, y sin mezclar consultas con código o mantener sincronizados los cambios de la base de datos con el modelo y viceversa.
- ◆ Mayor seguridad, ya que se ocupan automáticamente de higienizar los datos que llegan, evitando posibles ataques de inyección SQL y similares.
- ◆ Hacen muchas cosas por nosotros: desde el acceso a los datos hasta la conversión de tipos o la internacionalización del modelo de datos.

Ventajas e inconvenientes de un ORM

- ◆ No son ligeros por regla general: añaden una capa de complejidad a la aplicación que puede hacer que empeore su rendimiento, especialmente si no los conocemos a fondo.
- ◆ La configuración inicial que requieren se puede complicar dependiendo de la cantidad de entidades que se manejen y su complejidad, del gestor de datos, etc...
- ◆ El hecho de que aísle de la base de datos y no tengamo casi ni que pensar en ella es un arma de doble filo. Si no sabemos bien lo que hacemos y las implicaciones que tiene en el modelo relacional podemos construir modelos que generen consultas monstruosas y muy poco óptimas contra la base de datos, agravando el problema del rendimiento y la eficiencia.

Hibernate

- ◆ Hibernate es un framework ORM
- ◆ El fichero de configuración de hibernate hibernate.cfg.xml se debe crear directamente dentro de la carpeta src del proyecto y el propio framework *Hibernate* será el encargado de leerlo para obtener las sesiones que permitan conectar con la Base de Datos con el código que se implementa en una clase de configuración que debemos crear

```
◆  <?xml version="1.0" encoding="UTF-8"?>
◆  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
◆  <hibernate-configuration>
◆  <session-factory>
◆    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
◆    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
◆    <property
      name="hibernate.connection.url">jdbc:mysql://sql7.freemysqlhosting.net:3306/sql7264745</prop
      erty>
◆      <property name="hibernate.connection.username">sql7264745</property>
◆      <property name="hibernate.connection.password">Zb1XGdwrD6</property>
◆      <b><mapping class="net.impactotecnologico.demohibernate.pojo.Categoria" /></b>
◆    </session-factory>
◆  </hibernate-configuration>
```

Mapeo de entidades/relaciones con clases/atributos

Java

- ◆ En el caso de las entidades, se deben anotar tanto la propia clase como cada uno de los atributos (se puede hacer en el atributo o en su getter/setter) para indicar con qué tabla mapearla y cómo mapear los atributos con los campos que corresponda, respectivamente.
- ◆ Las anotaciones que nos podemos encontrar para anotar una clase Java que debe ser mapeada con una tabla son:
 - ◆ @Entity Indica que la clase es una tabla en la base de datos
 - ◆ @Table(name = “nombre_tabla”) Indica el nombre de la tabla

Mapeo de entidades/relaciones con clases/atributos

Java

- ◆ En el caso de los atributos simples que deben ser mapeados con los campos de la tabla correspondiente:
 - ◆ @Id Indica que un atributo es la clave
 - ◆ @GeneratedValue(strategy = GenerationType.IDENTITY) Indica que es un valor autonumérico (PRIMARY KEY en MySQL, por ejemplo)
 - ◆ @Column(name = “nombre_columna”) Se utiliza para indicar el nombre de la columna en la tabla donde debe ser mapeado el atributo

Mapeo de entidades/relaciones con clases/atributos Java

```
public class Categoría implements java.io.Serializable {
```

```
    private Integer id;  
    private String nombre;
```

```
    @Id  
    @GeneratedValue(strategy = IDENTITY)  
    @Column(name = "id", unique = true, nullable = false)  
    public Integer getId() {  
        return this.id;  
    }
```

Mapeo de entidades/relaciones con clases/atributos

Java

- ◆ Para el mapeo de las relaciones, además de crear el objeto que permita mantener la relación entre las clases (de forma bidireccional), éstos atributos deben ser mapeados según convenga: se indicará el tipo de relación visto desde el lado correspondiente
 - ◆ @OneToOne Indica que el objeto es parte de una relación 1-1
 - ◆ @ManyToOne Indica que el objeto es parte de una relación N-1. En este caso el atributo sería el lado 1
 - ◆ @OneToMany Indica que el objeto es parte de una relación 1-N. En este caso el atributo sería el lado N
 - ◆ @ManyToMany Indica que el objeto es parte de una relación N-M. En este caso se indica la tabla que mantiene la referencia entre las tablas y los campos que hacen el papel de claves ajenas en la base de datos

- ◆ `@Entity`
- ◆ `@Table(name = "productos")`
- ◆ `public class Producto {`
- ◆ `...`

- ◆ `@ManyToOne`
- ◆ `@JoinColumn(name="id_categoria")`
- ◆ `private Categoria categoria; ...`

- ◆ `... }`

- ◆ `@Entity`
- ◆ `@Table(name = "categorias")`
- ◆ `public class Categoria {`
- ◆ `...`
- ◆ `@OneToMany(mappedBy = "categoria", cascade = CascadeType.ALL)`
- ◆ `private List<Producto> productos; ...`

Registrar un objeto

- ◆ Para registrar un nuevo objeto en la Base de Datos necesitamos haber creado previamente la clase y haberla mapeado correctamente con la tabla que le corresponda. Entonces, utilizando la clase **HibernateConnector** podremos obtener una sesión (conexión con la Base de Datos) para registrar ese objeto directamente en la Base de Datos de la siguiente forma:

Registrar un objeto

- ◆ Hay que tener en cuenta que entre el inicio y cierre de la transacción podemos realizar más de una operación y éstas se ejecutarán como tal.
- ◆ Es la forma correcta en el caso de que queramos registrar más de un objeto cuando éstos estén relacionados de alguna forma y dependan entre ellos.
- ◆ Un caso muy claro sería el del registro de un pedido junto con todas sus líneas de detalle puesto que no tendría sentido registrarlo sin los detalles, por lo que la forma más segura sería darlos de alta dentro de una misma transacción.

```
Session sesion = HibernateConnector.getCurrentSession();
sesion.beginTransaction();
sesion.save(unPedido);
for (DetallePedido detallePedido : detallesDelPedido)
    sesion.save(detallePedido);
sesion.getTransaction().commit();
sesion.close();
```



Gracias!

Alguna pregunta?

Pueden contactarnos mediante
info@impactotecnologico.net y apuntarse a
algunos de nuestros cursos



