

Tu carrera digital ~

# PROGRAMACIÓN WEB FULL STACK

# Índice

<b>Módulo 1 Fundamentos del desarrollo web</b>	
Introducción	3
Breve introducción a la línea de comandos	26
Introducción a la pila de protocolos TCP/IP	33
Git	50
<b>Módulo 2 Diseño de páginas interactivas frontend (HTML)</b>	
HTML5 (I)	61
Nuevas etiquetas	72
Diseño de páginas interactivas frontend (CSS)	
CSS básico	83
<b>Diseño de páginas interactivas frontend (JavaScript)</b>	
Fundamentos	99
Variables	113
Control de flujo	131
Funciones	138
<b>Diseño de páginas interactivas frontend (TypeScript)</b>	
Fundamentos	154
<b>Diseño de páginas interactivas frontend (Angular)</b>	
Fundamentos (I)	165
Fundamentos (II)	176
<b>Módulo 3 Java básico</b>	
Introducción a Java	198
Fundamentos	221
Control del flujo	245
Orientación a objetos	270
<b>Módulo 4 Bases de datos (SQL)</b>	
Introducción las bases de datos, MariaDB y SQL	310
JDBC	350
<b>Módulo 5 Spring e Hibernate</b>	
Introducción a Spring	370
Inversión de control	386
Inyección de dependencias	394
Spring MVC con Hibernate	409

Tu carrera digital ~

# Módulo 1

# Fundamentos del

# desarrollo web

Introducción



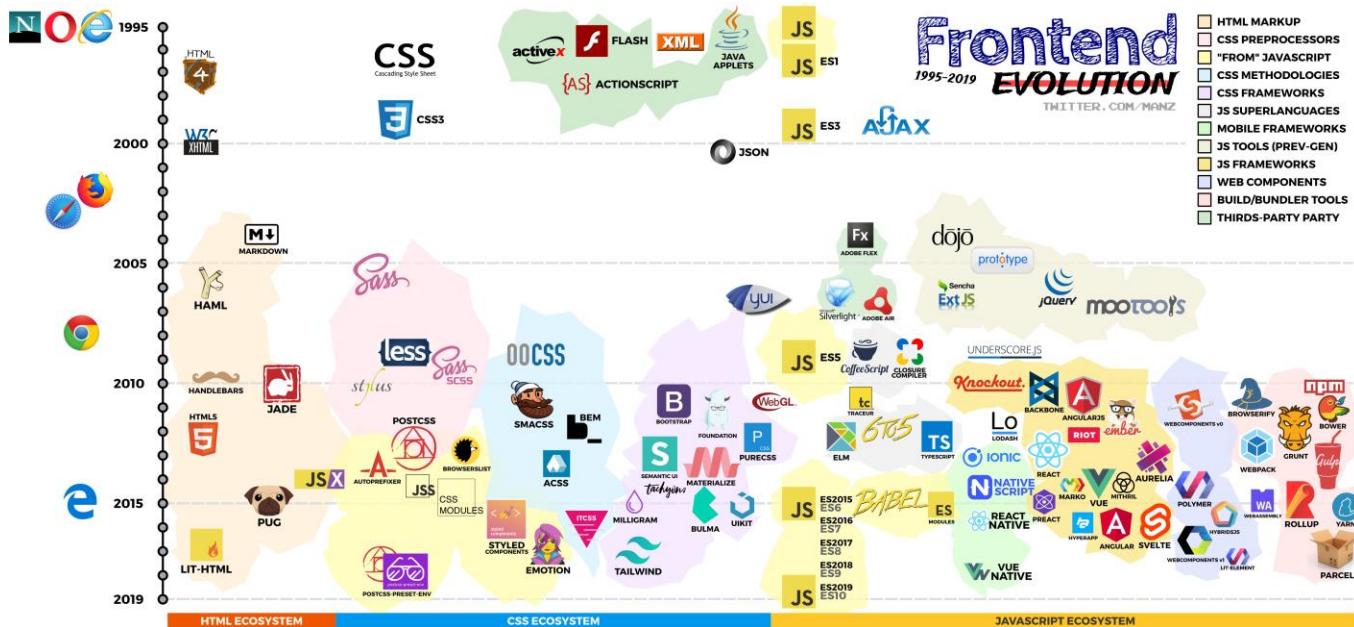
# Introducción

---

- ◆ Introducción básica a Internet
- ◆ Navegadores web
  - Nestcape
  - Internet Explorer - Edge
  - Mozilla
  - Firefox
  - Safari
  - Chrome
  - Opera
- ◆ Diferencias entre Frontend y Backend
- ◆ Estado actual del Frontend, Backend y Devops en la industria tecnológica
- ◆ Tendencias futuras
  - Nuevas WebAPI
  - Web3
- ◆ Enlaces de interés

## Introducción básica a Internet

- ◆ Internet es un conjunto descentralizado de redes interconectadas entre sí que se constituyen globalmente, mediante protocolos de comunicación, como una red única de alcance mundial.
- ◆ Por otra parte, la *World Wide Web* (WWW), o simplemente Web, representa un universo de información organizado a través de recursos o páginas web interconectadas mediante Internet.
- ◆ El funcionamiento de la Web es posible gracias a la existencia de una serie de componentes software (programas) y hardware (componentes físicos), entre los que se incluye principalmente los dispositivos de comunicación (*hubs*, repetidores, puentes, pasarelas, encaminadores o *router*) y también protocolos de comunicación (TCP, IP, HTTP, FTP, SMTP, ...).



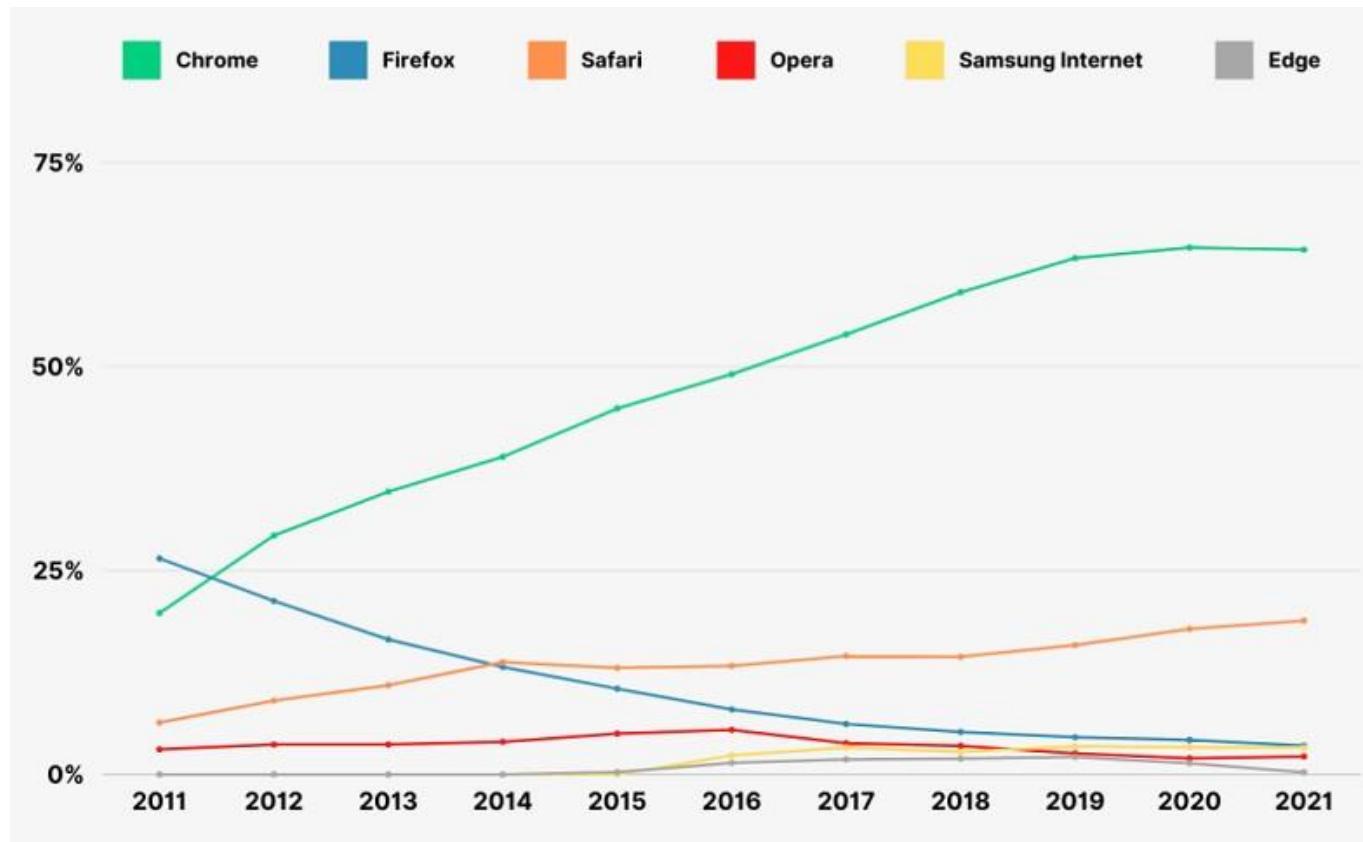
- El funcionamiento básico de la web se fundamenta en el modelo de comunicación cliente/servidor, siendo el cliente quien representa al componente consumidor del servicio, mientras que el servidor es quien provee el servicio.
- En esta comunicación cliente/servidor, el cliente es habitualmente quien inicia el intercambio de información mediante una solicitud al servidor, que responde enviando una respuesta. En este punto, la comunicación finaliza.
- Entre las características más importantes en el modelo de comunicación cliente/servidor se encuentran:
  - El cliente puede conectarse a varios servidores al mismo tiempo.
  - El servidor puede aceptar conexiones de un gran número de clientes.
  - El servidor no posee la capacidad de establecer una comunicación con el cliente. Es el cliente el que siempre inicia la comunicación.
  - Tanto el cliente como el servidor pueden ubicarse en el mismo computador.
- El cliente generalmente representa al usuario final, que utiliza algún tipo de dispositivo de electrónica de consumo (teléfono, ordenador, televisión, reloj inteligente), mientras que el servidor es un computador de medias o altas prestaciones que opera sin intervención humana.
- Las funcionalidades en los entornos clientes/servidor suelen agruparse en diferentes niveles o capas. Cada capa se centra en la descripción de un aspecto específico de la comunicación web entre cliente y servidor.



- ◆ Generalmente se identifican tres tipos de capas fundamentales: capa de presentación, capa de la lógica de negocio o reglas de negocio y capa de datos. La ubicación de cada una de estas capas depende del entorno de ejecución, lenguaje de programación, tecnología o tendencia web aplicada:
  - Capa de presentación: es la parte visible de la web, es decir, la interfaz gráfica con la que interactúa el usuario final. La programación de esta capa se centra en dar formato a la información enviada por el servidor para que sea legible y atractiva para el usuario. También se encarga de capturar, manejar y responder a las acciones realizadas por el usuario a través de la interfaz gráfica. Generalmente la lógica de esta capa de presentación se ejecuta en una aplicación denominada navegador web, que se encuentra en el cliente.
  - Capa de reglas de negocio: recibe las peticiones desde la capa de presentación, las procesa y devuelve la respuesta. Generalmente esta lógica de negocio se ubica en el servidor.
  - Capa de datos: es el lugar donde residen los datos y también la encargada de acceder a los mismos. Normalmente está formada por uno o más sistemas de gestión de bases de datos que se encargan de gestionar los datos, proporcionando a la capa de negocio de diferentes operaciones de almacenamiento, filtrado, búsqueda, recuperación, entre otras.

## Navegadores web

- ◆ Los navegadores web son programas que permite acceder a la web. Desde su aparición en 1990, los navegadores han evolucionado conforme avanzaba también la propia web.



- El primer navegador web fue WorldWideWeb, creado por Tim Berners Lee en 1990. Poco después surgió MidasWWW, Cello y Mosaic (que se hizo el más popular).
- Los navegadores web se fundamentan en los siguientes principios:
  - Funcionan con el modelo cliente/servidor. El navegador web es el cliente, mientras que el servidor es una máquina remota que almacenan el recurso de información
  - El navegador web (cliente) accede al servidor a través de un Localizador de Recursos Uniforme (URL).
  - El navegador web (cliente) y el servidor se comunican mediante un protocolo común que ambos entienden. Este protocolo es conocido como Protocolo de Transferencia de Hipertexto (HTTP).
  - El servidor devuelve datos en un formato de modelado de información o marcas que permite al navegador web representar datos al usuario de forma estructurada. Este lenguaje de marcas se denomina (Lenguaje de Marcas de Hipertexto).
- Entre los navegadores más importantes se encuentran los siguientes.



## Nestcape

- ◆ Fue creado por Marc Andreessen (uno de los programadores de Mosaic) en 1994. Hasta 1997 fue el navegador más popular por varios motivos:
  - Siempre existieron versiones gratuitas.
  - Se publicaban versiones continuamente con nuevas funcionalidades.
  - Microsoft no incluyó en Windows un navegador web hasta 1996. Netscape aprovechó esta situación para situarse como única aplicación para el acceso a la Web.
- ◆ Windows incluyó Internet Explorer en Windows 95 OSR2 y la cuota de mercado de Netscape comenzó a decaer. Durante estos años se libró la llamada guerra de navegadores web entre Microsoft y Nestcape.
- ◆ Netscape se rindió en 1998 y antes de abandonar el mercado fundó la fundación sin ánimo de lucro Mozilla, para crear un navegador de software libre basado en Nestcape. En 1999 Netscape fue comprada por AOL (reconvertida ya en proveedor de Internet), que a su vez se fusionó con Time Warner en 2000. Aunque se siguieron publicando versiones de Netscape basadas en Mozilla hasta 2008, Netscape fue irrelevante desde el 2000..
- ◆ La última versión de Nestcape se liberó en 2008.

## Netscape



## Internet Explorer - Edge

- ◆ Microsoft presentó Internet Explorer (IE) en agosto de 1995, basándose en una versión de Mosaic.
- ◆ Posteriormente se publicaron versiones casi cada año: Internet Explorer 2 (1995), Internet Explorer 3 (1996), Internet Explorer 4 (1997), Internet Explorer 5 (1999), Internet Explorer 5.5 (2000) e Internet Explorer 6 (2001).
- ◆ Microsoft anunció en 2003 que únicamente publicarían nuevas versiones de IE cada nueva versión de Windows, pero esta decisión cambió en 2005 con la llegada de Firefox.
- ◆ Microsoft comenzaría a respetar los estándares de la web a partir de Internet Explorer 7, facilitando la labor a los programadores web, que hasta entonces debían programar las páginas web en dos versiones: una para Internet y otra para el resto de navegadores.

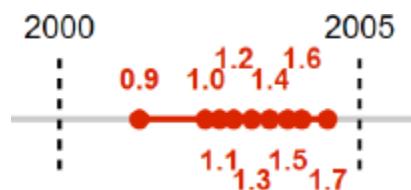
- ◆ Microsoft publicó Edge 12 en 2015 para Windows 10. De esta forma, Internet Explorer es abandonado en la versión 11, al igual que su motor de renderizado Trident. EdgeHTML pasó a convertirse en el motor de renderizado de Edge.
- ◆ En diciembre de 2018, Microsoft anunció que las futuras versiones de Edge estarían basadas en Chromium, el motor de Google Chrome, y que este navegador estaría disponible para todas las versiones de Windows. Antes de Edge Chromium, Microsoft Edge sólo se podía instalar en Windows 10, pero a partir de Edge 79 (publicado en enero de 2020), Edge se puede instalar en Windows 7, 8, 8.1 o 10.
- ◆ La versión más moderna de Internet Explorer que puede instalarse para cada versión de Windows son:
  - Windows XP: Internet Explorer 8.
  - Windows 7, Vista y 8: Internet Explorer 11 y Edge Chromium
  - Windows 10 y 11: Edge y Edge Chromium.



## Mozilla

- ◆ Mozilla era el nombre interno con el que era llamado el navegador Netscape.
- ◆ Netscape anunció en 1998 la liberación del código fuente de su navegador, recibiendo el nombre de Mozilla. La idea era proporcionar una suite que incluyera no solo el navegador, sino también un cliente de correo electrónico, un programa de chat y un editor de código. En junio de 2002 se publicó Mozilla 1.0.
- ◆ Durante esos años, la financiación del proyecto provenía de AOL, que utilizaba Mozilla como base para las versiones de Netscape que siguieron publicándose durante unos años. AOL alcanzó un acuerdo con Microsoft en 2003 para poner fin a las demandas por abuso de posición dominante. Como consecuencia, Microsoft pagó a AOL 750 millones de dólares y, a cambio, AOL pasó a utilizar Internet Explorer en vez de Netscape. De esta forma, AOL dejó de financiar el desarrollo de Mozilla.
- ◆ En 2004 se crea la Fundación Mozilla para poder continuar el desarrollo de Mozilla. Se trata de una fundación sin ánimo de lucro que recibe la mayor parte de sus ingresos de Google.
- ◆ De 2002 a 2004 todavía se siguieron publicando numerosas versiones de Mozilla, pero se decidió separar (seguramente por influencia de Google, entre otros factores) los componentes de la suite de Mozilla y publicarlos como programas separados (el navegador Firefox, el cliente de correo electrónico Firebird, etc).

- La última versión de Mozilla se publicó en junio de 2004 y en ese mismo año se publica la primera versión de Firefox.
- En 2005 el desarrollo de Mozilla se dio por terminado.



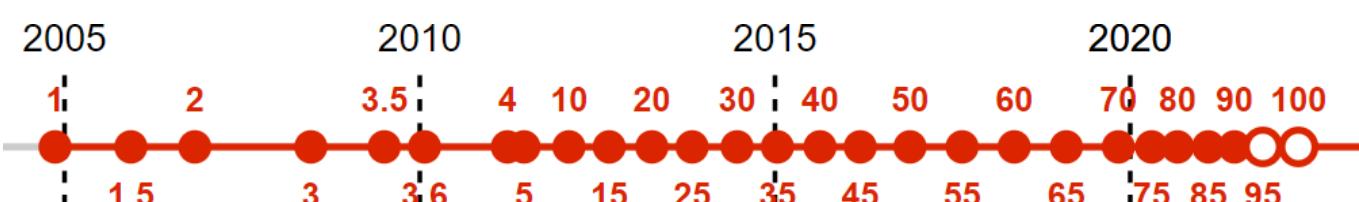
## Firefox

- Firefox es el navegador creado por la Fundación Mozilla y es continuación del navegador Mozilla, que a su vez es continuación del navegador Netscape. Su motor de renderizado es el que fue originalmente desarrollado para Nestcape: [Gecko](#).
- Además de cumplir las recomendaciones del [W3C](#) (no solamente respecto al HTML y a

- CSS, sino también otros estándares importantes SVG o MathML), Firefox pone el énfasis en la usabilidad (pestañas, interfaz, etc), facilitando además la personalización y ampliación a través de extensiones.

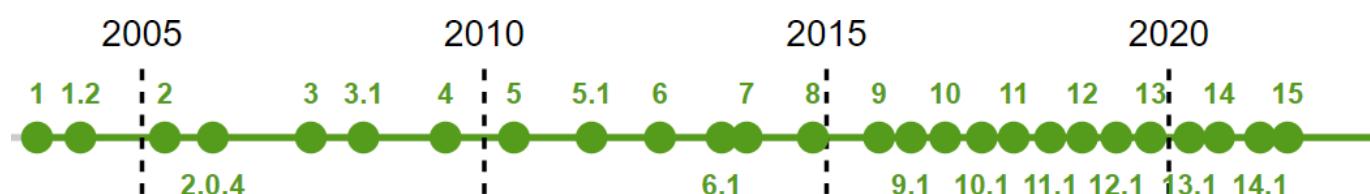
- El desarrollo de Firefox está financiado principalmente por Google, a través de donaciones a la Fundación Mozilla. A cambio, la página de inicio de Firefox es la página web del buscador de Google. Cuando Google comenzó a publicar en 2008 su propio navegador (Google Chrome), surgieron dudas sobre la continuidad de esas donaciones, pero el acuerdo se siguió renovando hasta la actualidad.
- Firefox se convirtió en el navegador alternativo a Internet Explorer en 2005 y su uso creció hasta casi el 25% a principios de 2009. Sin embargo, la aparición de Google Chrome por esas fechas detuvo su crecimiento.

- A partir de la versión 57, Firefox ya no admite extensiones basadas en [XUL](#), el lenguaje de interfaces de Mozilla. Actualmente Firefox sólo admite las extensiones que utilizan [WebExtensions](#), un conjunto de librerías de programación admitido por todos los navegadores (con pequeñas variantes). La ventaja para los creadores de extensiones es la compatibilidad para todos los navegadores. La desventaja es que muchas de las antiguas extensiones de Firefox han dejado de funcionar.
- Otras [iniciativas para incluir publicidad](#) en el navegador tampoco han ayudado a Firefox a tratar de recuperar cuota de mercado.



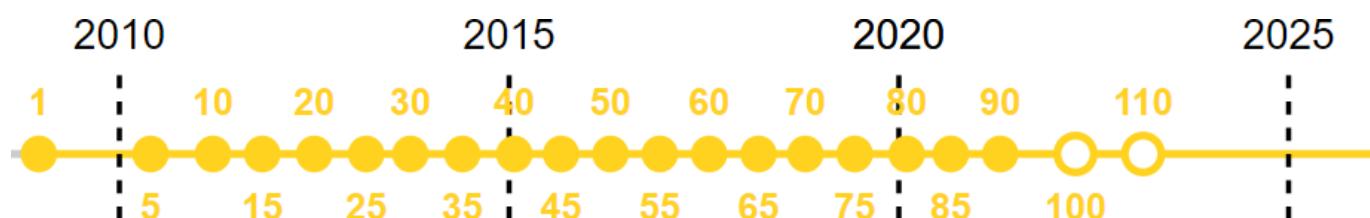
## Safari

- Hasta 2003 el sistema operativo Mac de Apple no disponía de su propio navegador web, por lo que incluía Netscape o Internet Explorer. Sin embargo, en junio de 2003 Apple publicó Safari 1.0 para Mac OS X.
- Safari utiliza el motor de renderizado WebKit, desarrollado por Apple a partir del motor de renderizado KHTML del proyecto de software libre KDE.
- El éxito de los teléfonos de Apple impulsó el uso de Safari, aunque su uso está limitado a dispositivos de la propia Apple. Entre 2007 y 2012 se llegaron a publicar versiones de Safari para Windows, pero sin ningún éxito.



## Chrome

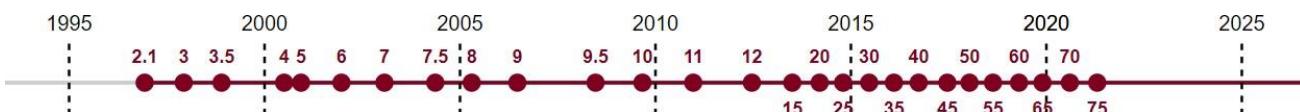
- Chrome es un navegador creado en 2008 por Google a partir de WebKit, el motor de renderizado del navegador Safari. WebKit fue sustituido por Blink a partir de la versión 28 (2013).
- Chrome ha destacado siempre por su interfaz minimalista y por la velocidad de ejecución del código Javascript.
- Los lanzamientos de Chrome son obtenidos a partir de Chromium, el proyecto de software libre que también sirve de base para el sistema operativo Chrome OS.12 Chromium y también para otros navegadores web como Brave.



## Opera

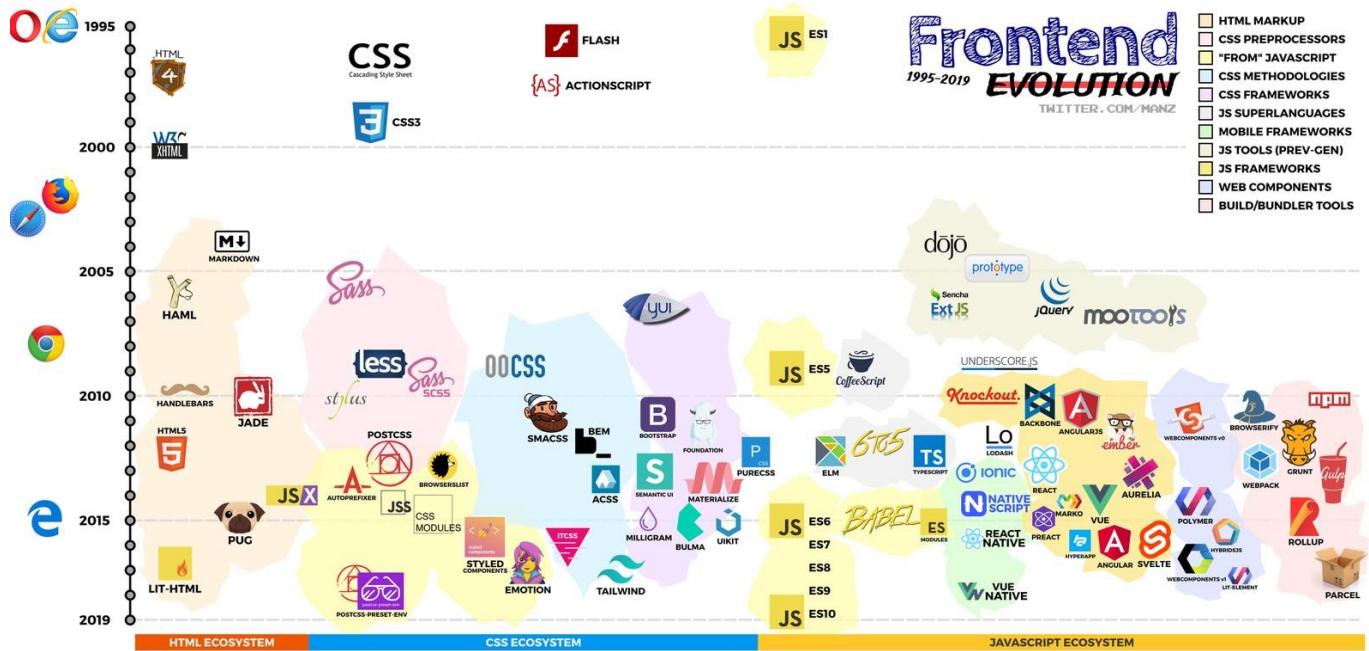
- Opera comenzó su andadura en 1994 como un proyecto de investigación de Telenor, una compañía telefónica Noruega. A partir de 1995 se constituye como Opera Software.
- La primera versión, Opera 2.1, se publicó en diciembre de 1996 y desde entonces ha ido publicando versiones tanto para PCs como para dispositivos móviles.

- Su principal característica ha sido siempre el cumplimiento de las recomendaciones del W3C, posiblemente debido a que el Director de Tecnología Håkon Wium Lie fue inventor de las hojas de estilos en cascada (CSS).
- El motor de renderizado de Opera fue Presto desde sus inicios y hasta 2013, momento en el que fue reemplazado por WebKit.
- Opera nunca ha tenido una gran cuota de mercado, excepto en dispositivos móviles donde ha perdido usuarios debido a la competencia con Safari, Android y otros navegadores web móviles.



## Diferencias entre Frontend y Backend

- Frontend y backend son conceptos muy populares en el contexto del desarrollo web. Cada uno representa diferentes estilos de programación o tecnologías.
- El Frontend (o lado del cliente) representa al conjunto de tecnologías alrededor del Lenguaje de Marcado de Hipertexto (HTML), las hojas de estilo de cascada (CSS) y el lenguaje JavaScript.
  - HTML es un lenguaje de modelado de la información o lenguajes de marcado (etiquetas), que estructura y organiza el contenido web para ser mostrado en un navegador.
  - CSS es un lenguaje de estilos que acompaña al HTML y establece debe mostrarse visualmente el contenido web en términos de colores, tipografía, disposición en columnas/filas, etc.
  - JavaScript es un lenguaje de programación utilizado para proporcionar interactividad al contenido web mediante menús desplegables, animaciones complejas, notificaciones, eventos, etc.
- Alrededor del conjunto de tecnologías HTML/CSS/JavaScript existe un sinfín de herramientas software (librerías, frameworks, preprocesadores, etc.) para simplificar y agilizar el trabajo a los programadores.



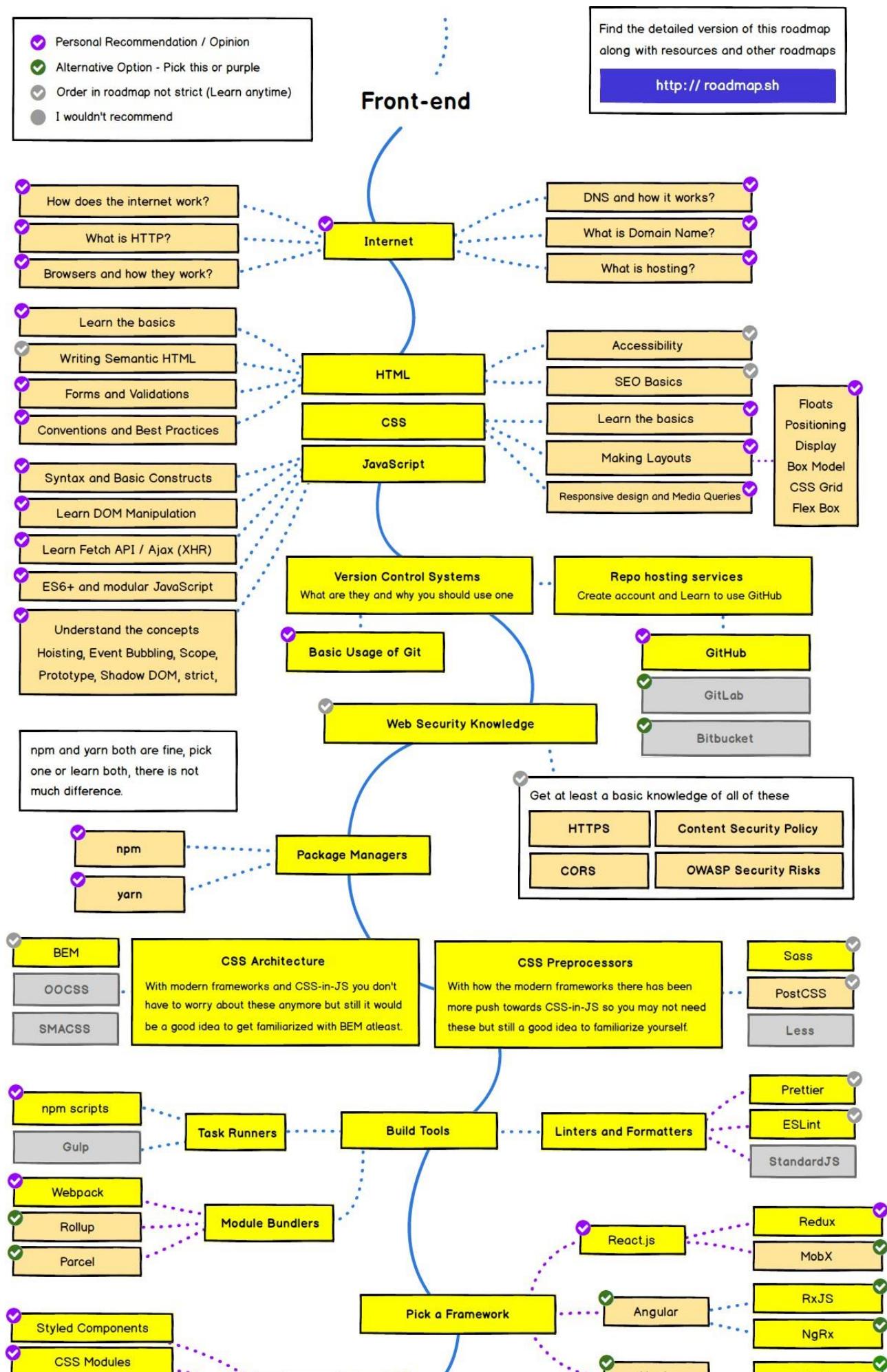
- HTML/CSS/JavaScript evolucionan con el tiempo con la inclusión de nuevas funcionalidades. La adopción de estas mejoras es más o menos rápida en función del navegador web.
- El Backend (o lado del servidor) está compuesto por tres grandes bloques:
  - Los datos: almacenados en un sistema de almacenamiento que generalmente lo provee un Sistema de Gestión de Bases de Datos (SGBD), o también llamado simplemente base de datos. Existen dos tipos de base de datos: las que se basan en el Lenguaje de Consulta Estructurado (SQL) y las que no (NoSQL).
  - El propio Backend: se trata de una aplicación escrita en algún lenguaje de programación (Java, Go, Node, Python, Ruby, etc), que se encarga de:
    1. Escuchar y atender peticiones entrantes basadas en el protocolo HTTP.
    2. Realizar consultas a la base de datos en base a la petición recibida.
    3. Devolver una respuesta (generalmente HTML/CSS/JavaScript, o datos en algún formato de modelado de la información, como JSON o XML).
  - La infraestructura: es el conjunto de elementos hardware y software que hacen posible el despliegue automático de la aplicación, su mantenimiento, escalabilidad y monitorización, así como otras tareas relacionadas con la fortificación de los sistemas (seguridad).

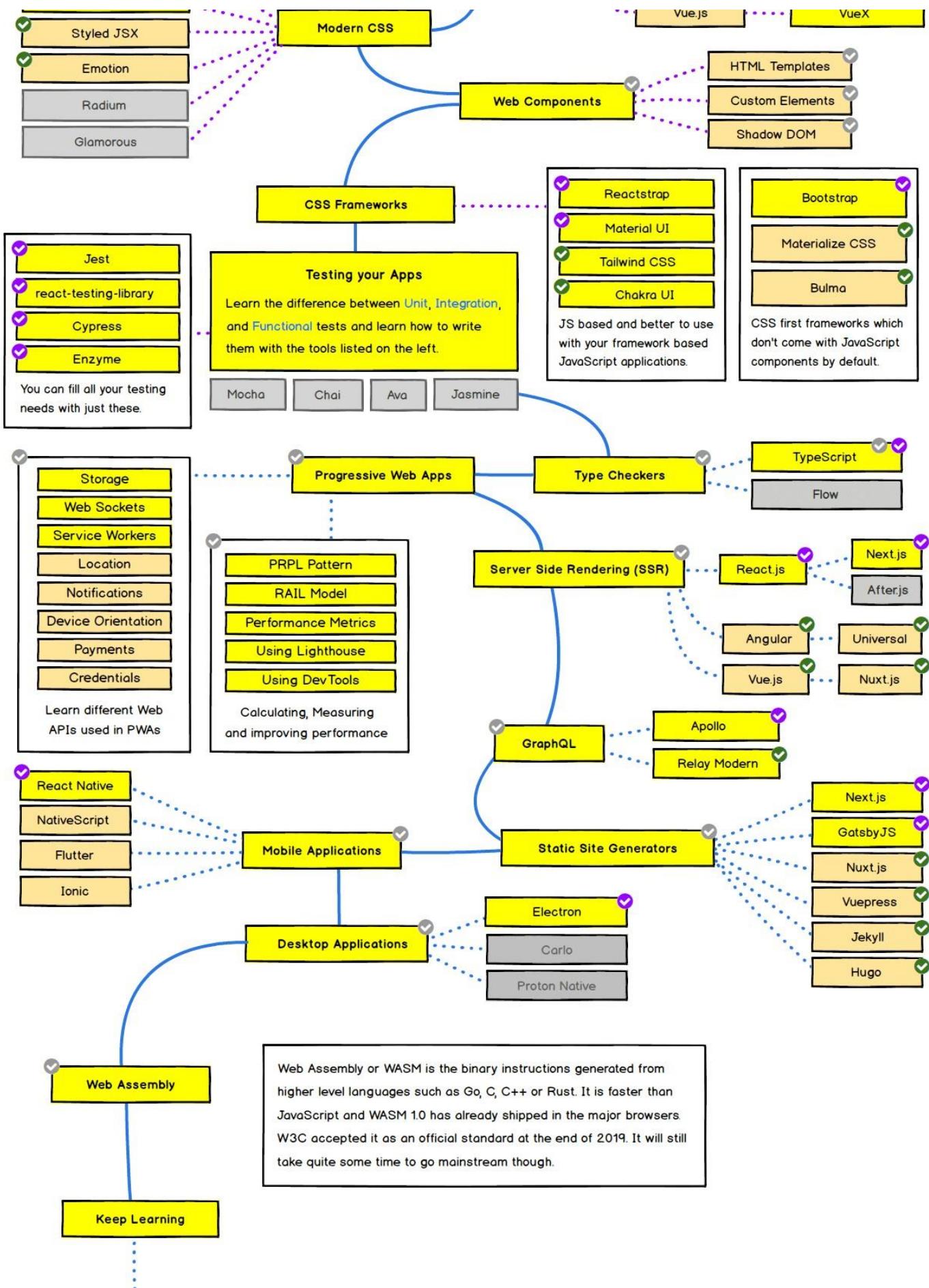


- La infraestructura del backend comprende multitud de tareas de alta complejidad y es habitual que el programador backend no se encarga directamente de ellas. Para ello existe un perfil técnico específico llamado Administrador de Sistemas o Devops.

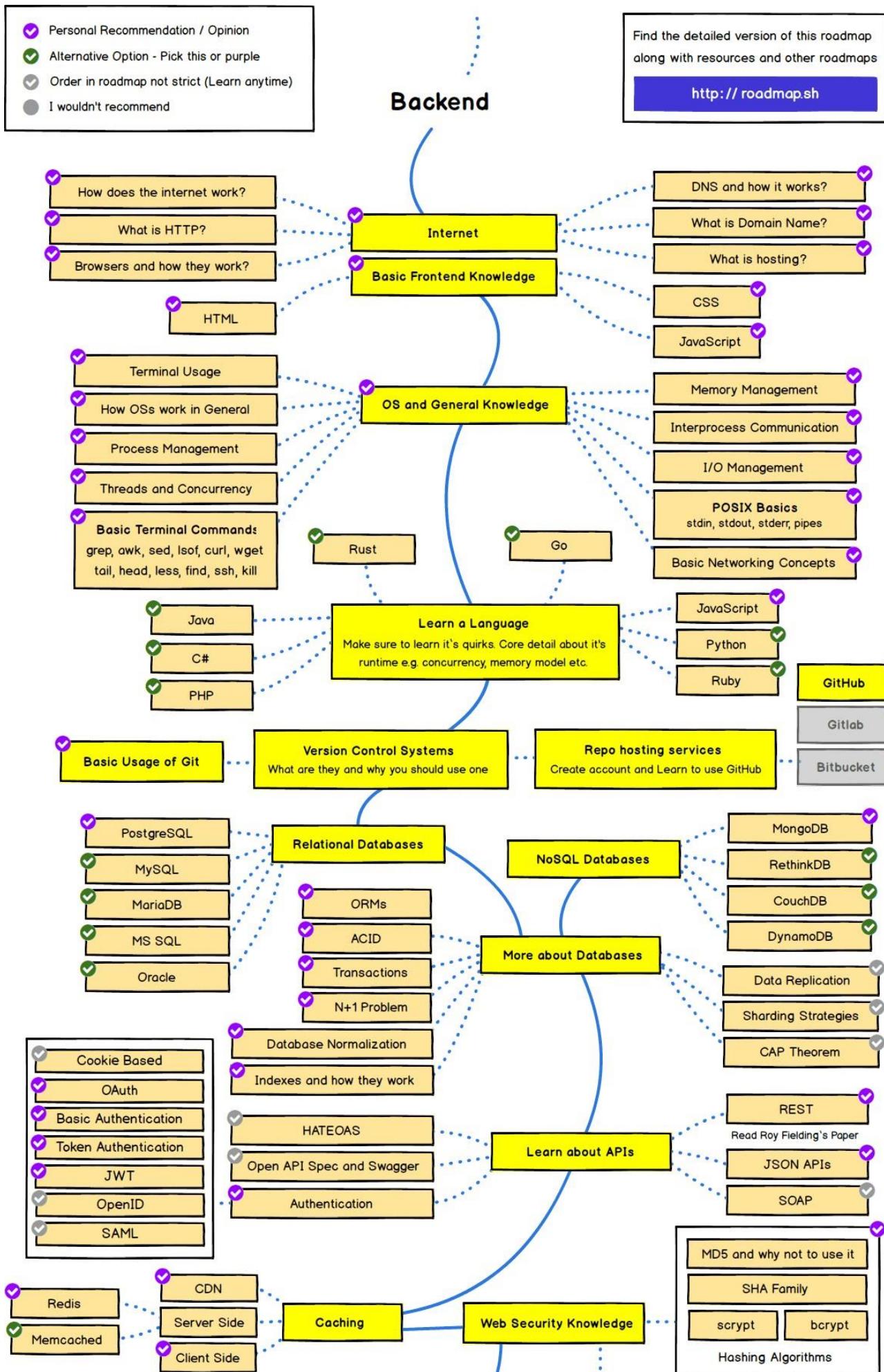
## Estado actual del Frontend, Backend y Devops en la industria tecnológica

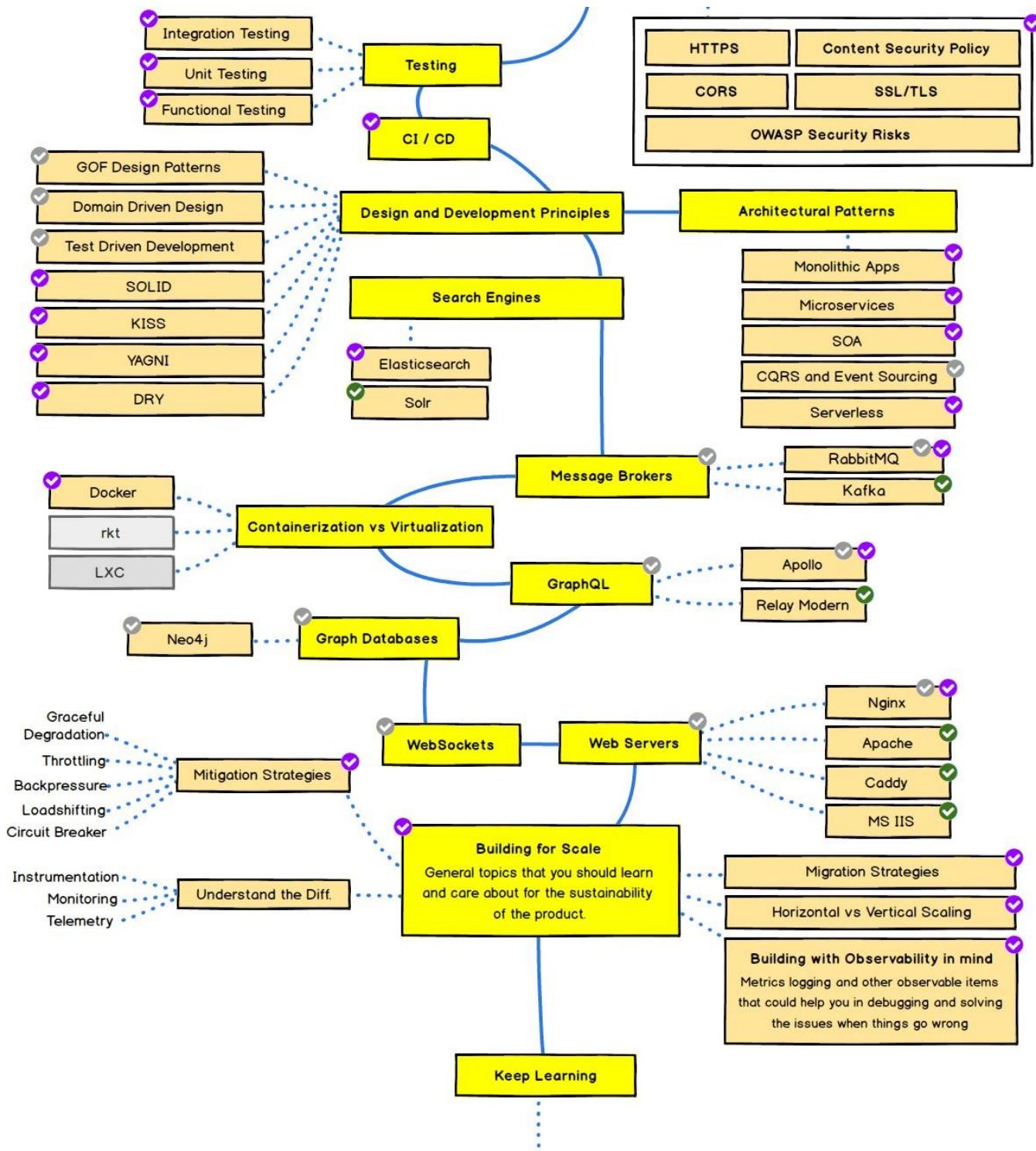
- Las rutas de aprendizaje del Frontend, Backend y Devops han ido variando a lo largo del tiempo en función de las necesidades de la industria tecnológica. Existen también importantes diferencias entre países.
- Sin embargo, hay un conjunto de tecnologías que han sido indiscutibles desde hace muchos años y actualmente se mantienen muy consolidadas.
- Un buen resumen de las diferentes rutas de aprendizaje puede encontrarse en [roadmap.sh](#)
- Entre los conceptos más importantes en la ruta de aprendizaje del programador Frontend se encuentran:
  - Internet.
  - Terminal de comandos.
  - Sistemas de control de versiones: Git y GitHub.
  - Seguridad de la información y los sistemas.
  - Protocolo HTTP.
  - HTML/CSS/JavaScript.
  - Gestores de paquetes.
  - TypeScript.
  - Frameworks frontend.
  - Frameworks de CSS.
  - Servicios Web.
  - Testing





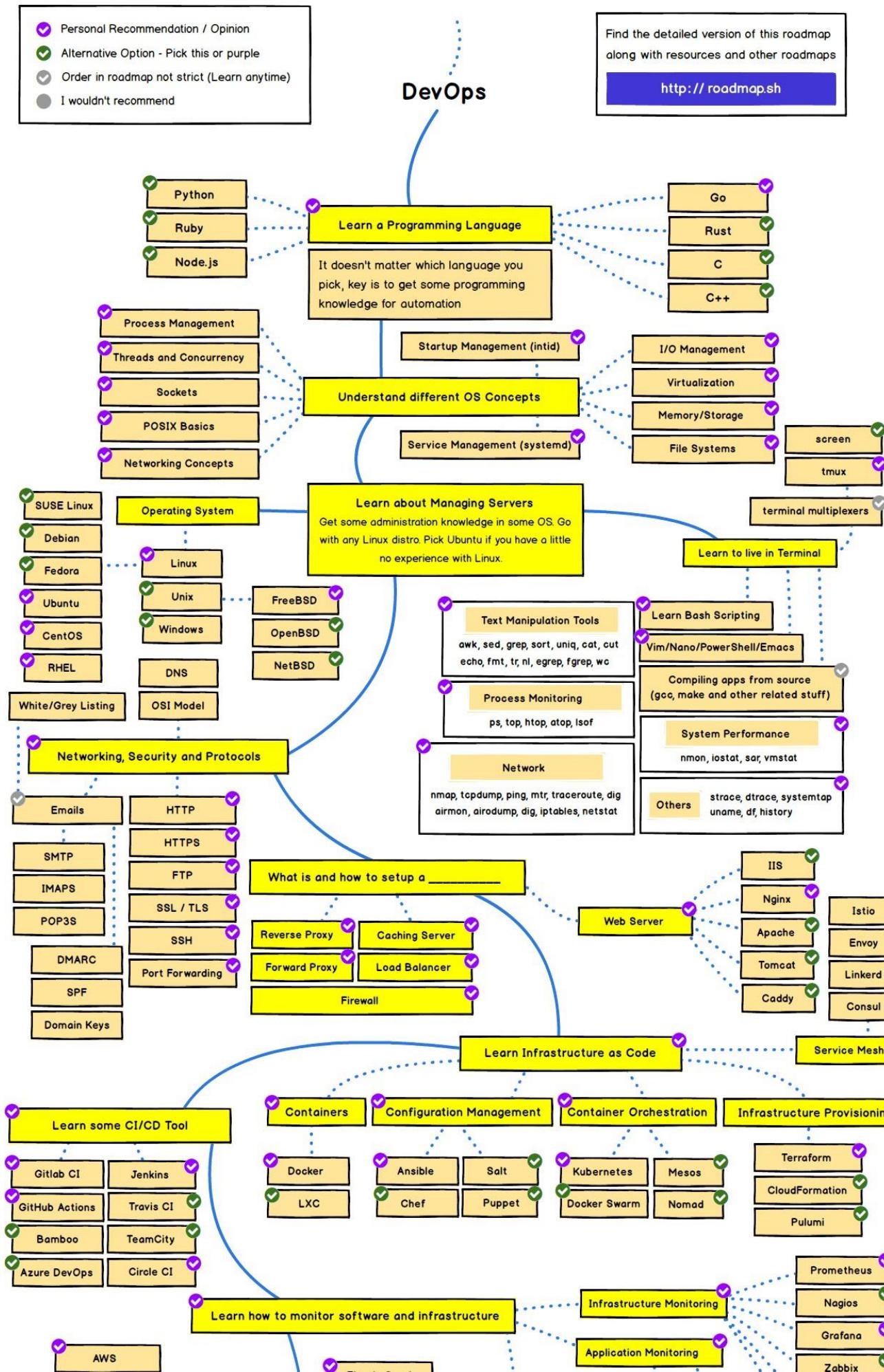
- ◆ Entre los conceptos más importantes en la ruta de aprendizaje del programador Backend se encuentran:
  - Internet.
  - Terminal de comandos.
  - Sistemas de control de versiones: Git y GitHub.
  - Seguridad de la información y los sistemas.
  - Protocolo HTTP.
  - Lenguajes de programación Backend: PHP, Java, Python, Node, Go, etc.
  - Bases de datos: SQL y NoSQL.
  - Servicios web.
  - Sistemas de caché.
  - Escalabilidad.
  - Testing.

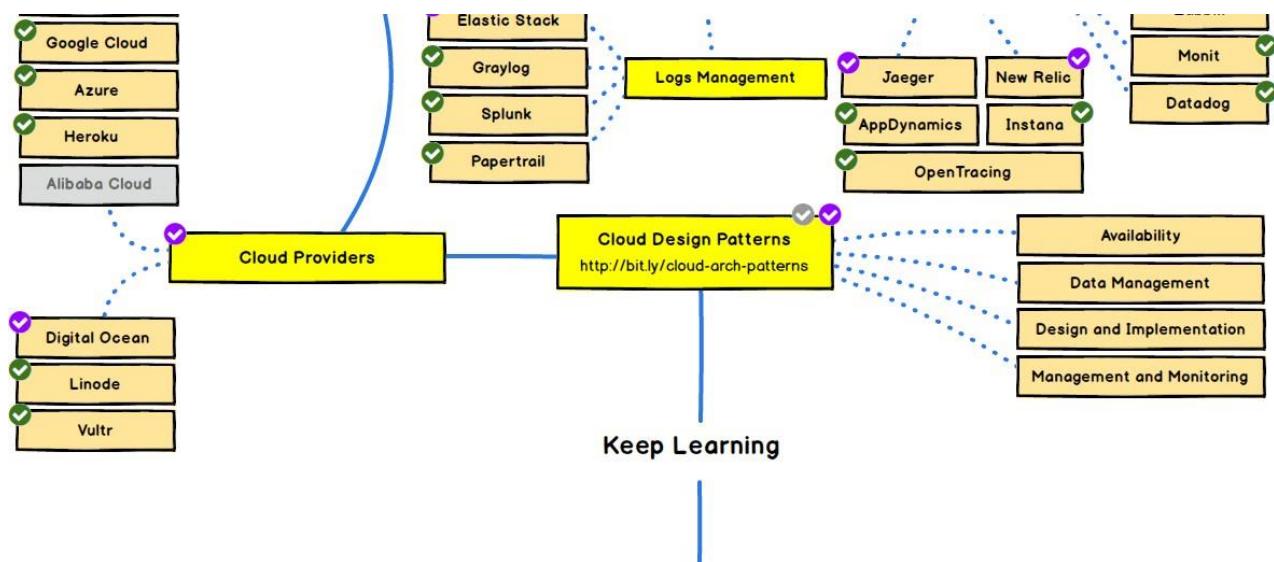




- Entre los conceptos más importantes en la ruta de aprendizaje del Devops se encuentran:
    - Internet.
    - Terminal de comandos.
    - Sistemas de control de versiones: Git y GitHub.
    - Seguridad de la información y los sistemas.
    - Pila de protocolos TCP/IP
    - Lenguajes de programación para administración de sistemas: Python, Bash, C++, Rust, etc.
    - Sistemas operativos.
    - Servidores web

- Bases de datos: SQL y NoSQL.
- Fortificación de sistemas
- Sistemas de monitorización y auditoría.
- Contenedores.
- Gestión de la configuración.
- Escalabilidad.
- Sistemas distribuidos.





## Tendencias futuras

- Existen multitud de esfuerzos en la actualidad que trabajan en la dirección de construir un futuro mejor en el contexto del desarrollo de aplicaciones web.
- A continuación se presentan dos enfoques bien distintos en tendencias futuras en el desarrollo web: WebAPI y Web3

### Nuevas WebAPI

- El World Wide Web Consortium (W3C) trabaja continuamente en el desarrollo de estándares y tecnologías que se integrarán en el futuro en todos los navegadores. Entre los más prometedores se encuentran:
  - WebGPU: es una API que expone las capacidades del hardware de las tarjetas gráficas para la Web.
    - La API está diseñada desde cero para gestionar de manera eficiente las funcionalidades nativas de las GPU (Unidad de procesamiento gráfico de las tarjetas gráficas).
    - WebGPU no está relacionado de manera alguna con la antigua WebAPI  WebGL.
    - WebGPU permitirá el desarrollo de juegos con gráficos intensos que se ejecuten en el navegador web directamente, aprovechando todo el poder de las tarjetas gráficas de última generación.
    - Su compatibilidad con los actuales navegadores web puede consultarse [aquí](#).
  - Generic Sensor API: se trata de una nueva API para la gestión de cualquier tipo de sensor (acelerómetro, giroscopio, termómetro, sensor de proximidad, sensores biométricos, etc.) alojado en un dispositivo con conectividad a la web.
    - Uno de los aspectos fundamentales de esta WebAPI es el control de la seguridad y privacidad de estos sensores por parte del usuario.
    - Algunos ejemplos pueden encontrarse en la [web](#).
    - Su compatibilidad con los actuales navegadores web puede consultarse [aquí](#).

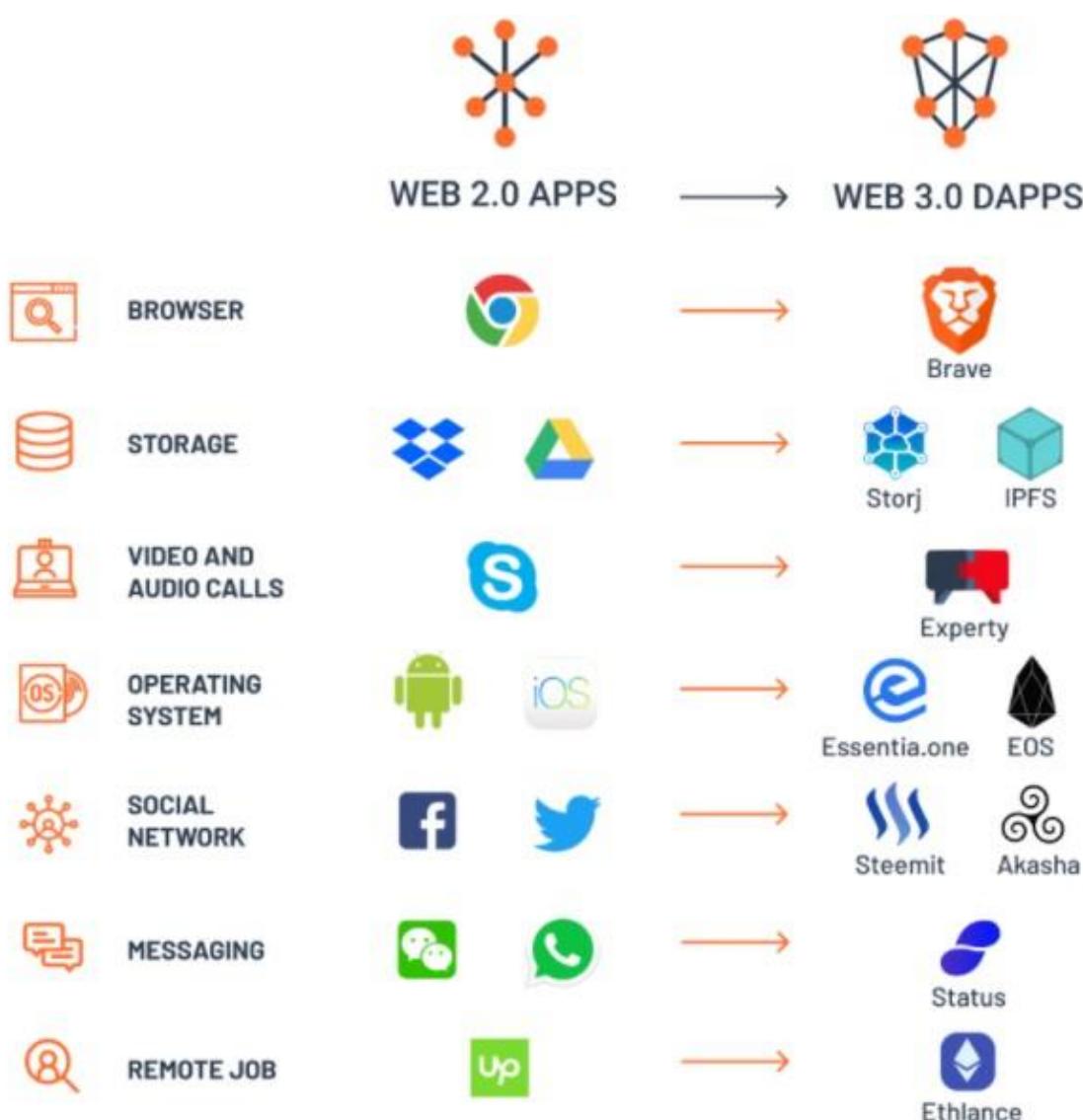
- Web Neural Network API: define una capa de abstracción independiente del hardware para hacer uso de las capacidades de aprendizaje automático y aprendizaje profundo.
  - La capa de abstracción aborda los requisitos de los marcos clave de JavaScript de aprendizaje automático y también permite a los desarrolladores web familiarizados con el dominio de la inteligencia artificial escribir código personalizado sin la ayuda de librerías externas.
  - Algunos casos de uso pueden ser la clasificación de imágenes, la detección de personas y objetos, el reconocimiento facial, el análisis de las emociones, la traducción automáticas en diferentes idiomas, el reconocimiento de habla, detección de noticias fake, etc.
- WebXR Device API: proporciona las interfaces necesarias para que los desarrolladores puedan crear aplicaciones inmersivas basadas en Realidad Virtual (VR) y Realidad Aumentada (AR).
  - Su compatibilidad con los actuales navegadores web puede consultarse [aquí](#).
  - Algunos experimentos de esta WebAPI han sido realizados por [Google](#).

## Web3

- Web3 se refiere a un ecosistema de tecnologías basadas en Blockchain, que tienen como finalizar la descentralización de Internet.
- Web3 está en contraposición con la Web 2.0, donde la gran mayoría de los datos y el contenido que se genera está centralizado en un grupo pequeño de empresas.
- Las tendencias actuales en el desarrollo de Web3 difieren, pero todas se basan en gran medida en tecnologías blockchain, incluyendo criptomonedas y tokens no fungibles (NFT), organizaciones autónomas descentralizadas (DAO), finanzas descentralizadas (DeFi) y/o contratos inteligentes.
- Nuevos lenguajes de programación como Solidity o Vyper estas surgiendo como alternativa al desarrollo de aplicaciones descentralizadas (DApp).
- Las DApps son un movimiento creciente de aplicaciones basadas en tres capas:
  - Frontend: es la interfaz que los usuarios utilizan para interactuar con la aplicación. En este caso, las DApp funcionan como las aplicaciones tradicionales, incluyendo interfaces web con HTML y Frameworks Frontend o librerías gráficas como Qt o GTK para el desarrollo de aplicaciones de escritorio.
  - Backend: se implementa como un contrato inteligente que se ejecuta sobre una blockchain (por ejemplo, Ethereum). El contrato inteligente está diseñado para garantizar el funcionamiento de la DApp y, además, son visibles y públicos, lo que permite un alto nivel de transparencia y seguridad. De esta forma, los usuarios pueden estar seguros que la DApp hace exactamente lo que se espera de ella.

Redes como Ethereum también permiten controlar la interacción con las capas de almacenamiento o la autenticación.

- Almacenamiento de datos: es totalmente descentralizado. Cada usuario de la DApp almacena un historial completo de las acciones que se realizan. Adicionalmente a esto, las interacciones son almacenadas en la blockchain de forma criptográficamente segura, impidiendo acceso no autorizados por terceras personas. De esta manera, si el dispositivo de un usuario se daña, bastaría con usar la DApp en un nuevo dispositivo para recuperar toda su información hasta ese preciso momento. El nivel de redundancia y seguridad de los datos aumenta a medida que haya más usuarios haciendo uso de la DApp.



- Las búsquedas de Web3 en Google han aumentado sustancialmente, así como las ofertas de trabajo en este sector.

## Enlaces de interés

- ◆ [Recursos para el desarrollador Frontend 1](#)
- ◆ [Recursos para el desarrollador Frontend 2](#)
- ◆ [Recursos para el desarrollador Frontend 3](#)
- ◆ [Recursos para el desarrollador Frontend 4](#)
- ◆ [Recursos para el desarrollador Frontend 5](#)
- ◆ [Recursos para el desarrollador Backend 1](#)
- ◆ [Recursos para el desarrollador Backend 2](#)
- ◆ [Recursos para el desarrollador](#)
- ◆ [Recursos relacionados con ciencias de computación](#)
- ◆ [Recursos para JavaScript](#)
- ◆ [Recursos para TypeScript 1](#)
- ◆ [Recursos para TypeScript 2](#)
- ◆ [Recursos para TypeScript 3](#)
- ◆ [Recursos para Java](#)
- ◆ [Recursos para Python](#)
- ◆ [Recursos sobre Git](#)
- ◆ [Recursos sobre Seguridad](#)
- ◆ [Buenas prácticas para Node.js](#)
- ◆ [Listado de aplicaciones para Windows](#)
- ◆ [Rutas de aprendizaje](#)
- ◆ [Listado de recursos de todo tipo](#)

# Módulo 1

# Fundamentos del

# desarrollo web

Breve introducción a la línea de comandos



# Breve introducción a la línea de comandos

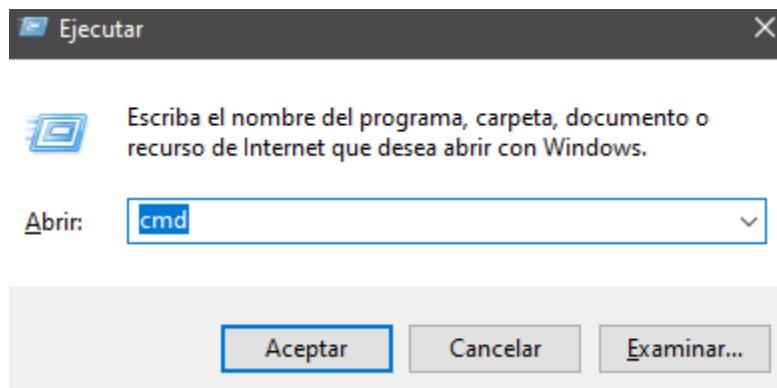
- ◆ Abrir una terminal de comandos
- ◆ Comando *cd*
- ◆ Comando *mkdir*
- ◆ Comando *rmdir*
- ◆ Comando *dir*
- ◆ Otros comandos

## Abrir una terminal de comandos

- ◆ La terminal de comandos de Windows puede abrirse de diferentes formas.
- ◆ La más simple es con la siguiente combinación de teclas:



- ◆ Y, a continuación, ejecutar cmd en la ventana de Ejecutar.

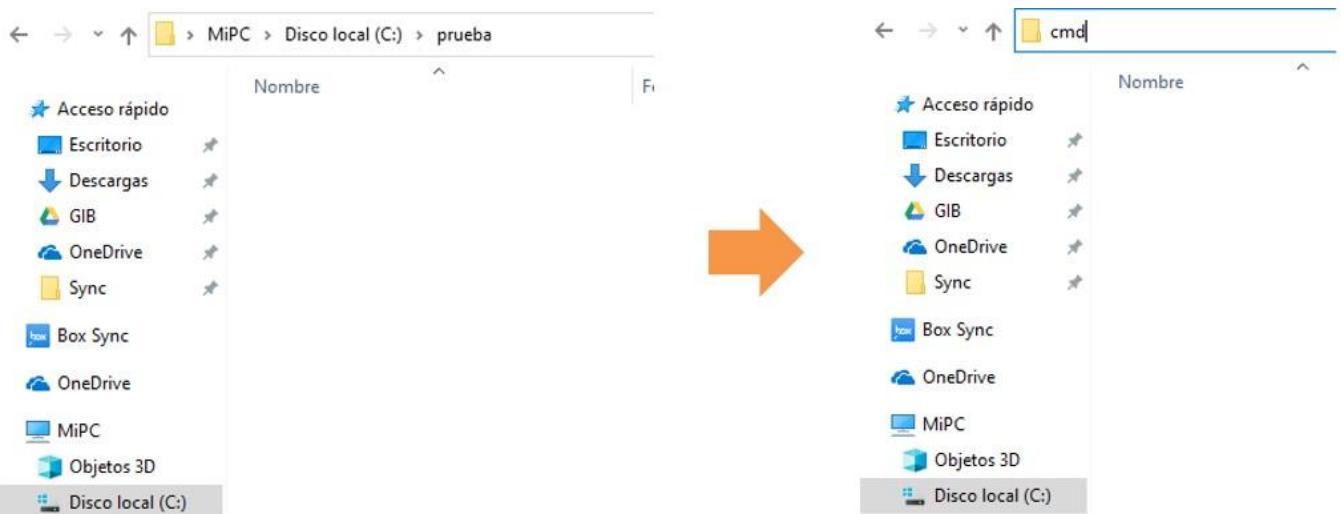


- ◆ La terminal de comandos se abre por defecto en la ruta c:\Users\USUARIO

```
C:\ C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Alejandro>
```

- Para abrir la terminal de comandos en otra ruta distinta puede escribirse *cmd* en la barra de direcciones del explorador de Windows, accediendo antes al directorio que se desea abrir.



- Ahora la terminal de comandos se abre en la ruta deseada.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>
```

## Comando cd

- El comando *cd* permite cambiar entre directorios.
- Escribir *cd* sin parámetros adicionales devuelve el directorio actual.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>cd
C:\prueba

C:\prueba>
```

- Añadir dos puntos al comando *cd* permite acceder al directorio superior.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>cd..
C:\>
```

- Para acceder de nuevo a un directorio inferior, se añade el nombre del directorio a continuación del comando *cd*.

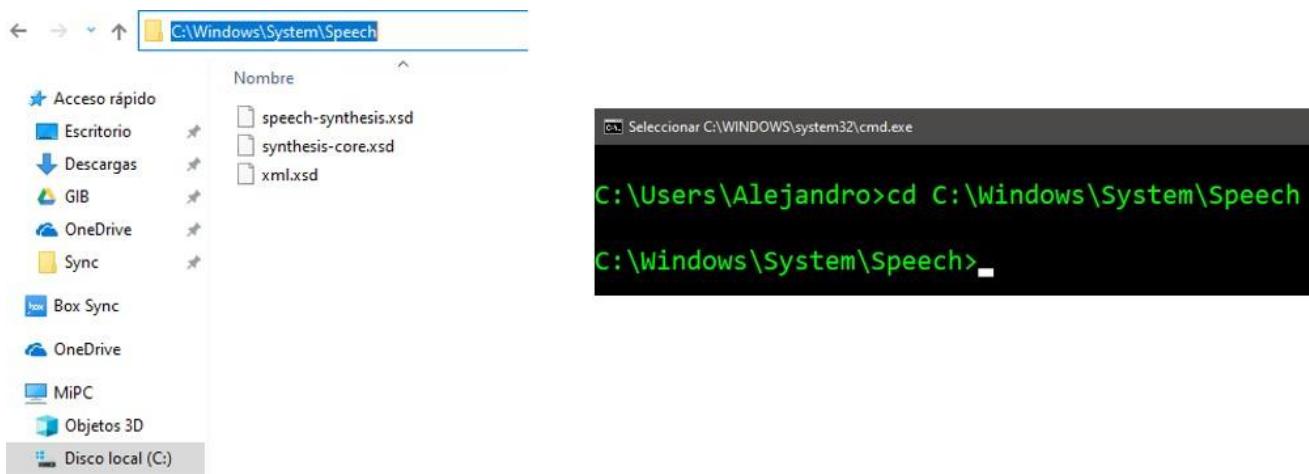
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>cd..
C:\>cd prueba
C:\prueba>cd
C:\prueba

C:\prueba>
```

- Es habitual tener que acceder a un directorio que se encuentra en un nivel muy inferior.

- Para acceder directorio a directorio hasta la ruta completa puede escribirse la misma después del comando `cd`.



## Comando `mkdir`

- El comando `mkdir` permite crear un directorio.
- Para ello se escribe del comando `mkdir` seguido del nombre del nuevo directorio.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>mkdir prueba2
C:\prueba>cd prueba2
C:\prueba\prueba2>cd..
C:\prueba>
```

## Comando `rmdir`

- El comando `rmdir` permite eliminar un directorio.
- Para ello se escribe del comando `rmdir` seguido del nombre del directorio a eliminar.
- El directorio solamente será eliminado si se encuentra vacío.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\prueba>rmdir prueba2

C:\prueba>
```

## Comando *dir*

- El comando dir permite visualizar el contenido (archivos y directorios) del directorio actual.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos re

C:\Windows>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 700E-C2E8

Directorio de C:\Windows

11/07/2020  11:17    <DIR>          .
11/07/2020  11:17    <DIR>          ..
19/03/2019   06:52    <DIR>        addins
19/07/2019   08:45    <DIR>        appcompat
10/06/2020  15:21    <DIR>        apppatch
16/06/2020  18:56    <DIR>        AppReadiness
```

## Otros comandos

- El comando *c/s* permite borrar todo el contenido de la ventana de la terminal de comandos.

```
C:\Windows\System32\cmd.exe
19/03/2019  06:46      64.512 twain_32.dll
19/03/2019  06:52    <DIR>      Vss
19/03/2019  06:52    <DIR>      WaaS
19/03/2019  06:52    <DIR>      Web
19/07/2019  01:36    <DIR>      WebManagement
18/03/2017  23:01          92 win.ini
11/07/2020  21:48        276 WindowsUpdate.log
19/03/2019  06:46        11.776 winhelp32.exe
11/07/2020  20:09    <DIR>      WinSxS
19/03/2019  14:02        316.640 WMSysPr9.prx
19/03/2019  06:45        11.264 write.exe
              33 archivos     81.544.997 bytes
              85 dirs   127.887.536.128 bytes libres

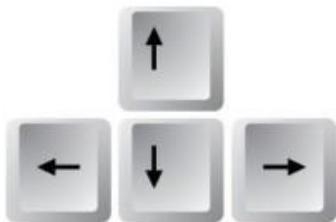
C:\Windows>cls
```

```
C:\Windows>
```

- También existen algunas teclas importantes del teclado que es importante conocer.



### Autocompletado



Las teclas arriba y abajo permiten acceder al historial de comandos

- El comando *exit* cierra la terminal de comandos

```
C:\Windows\System32\cmd.exe
C:\Windows>exit
```

Tu carrera digital ~

# Módulo 1

# Fundamentos del

# desarrollo web

Introducción a la pila de protocolos TCP/IP

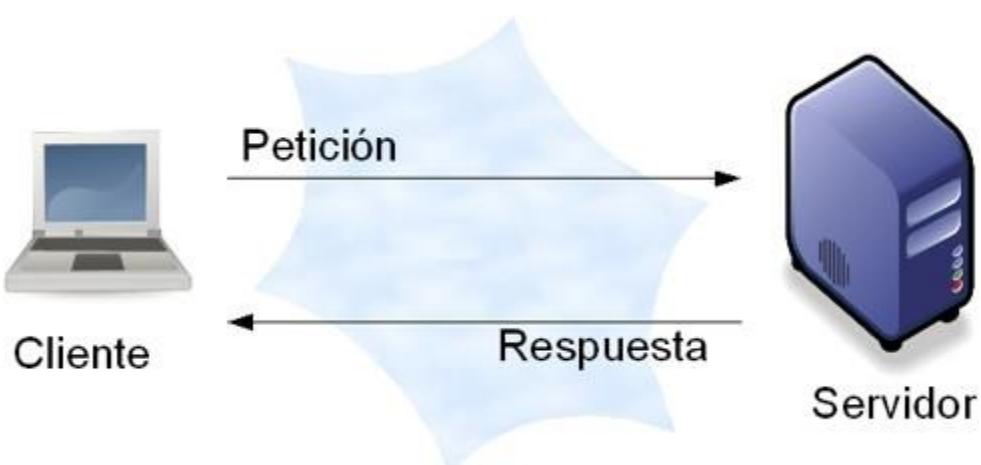


# Introducción a la pila de protocolos TCP/IP

- ◆ Arquitectura de comunicación cliente/servidor
- ◆ Arquitectura de comunicación P2P
- ◆ Arquitectura de comunicación publicador/suscriptor
- ◆ Modelo de capas TCP/IP
  - Nivel físico
  - Nivel de enlace de datos
  - Nivel de red
  - Nivel de transporte
  - Nivel de aplicación
- ◆ El protocolo DNS
- ◆ El protocolo HTTP
- ◆ Flujo de comunicación en tecnología web

## Arquitectura de comunicación cliente/servidor

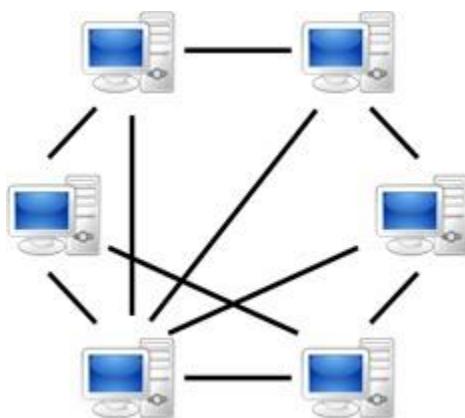
- ◆ La comunicación en Internet se basa en una arquitectura muy simple denominada cliente/servidor.
- ◆ El cliente inicia la comunicación mediante una petición.
- ◆ El servidor se encuentra "escuchando" peticiones y responde cuando recibe una.
- ◆ La comunicación finaliza cuando el servidor responde.



- ◆ Características importantes de la arquitectura cliente/servidor.
  - El cliente puede conectarse a varios servidores al mismo tiempo.
  - El servidor puede aceptar conexiones de un gran número de clientes.
  - El servidor no posee la capacidad de establecer una comunicación con el cliente.
  - Es el cliente el que siempre inicia la comunicación.
  - Tanto el cliente como el servidor pueden ubicarse en el mismo computador.

## Arquitectura de comunicación P2P

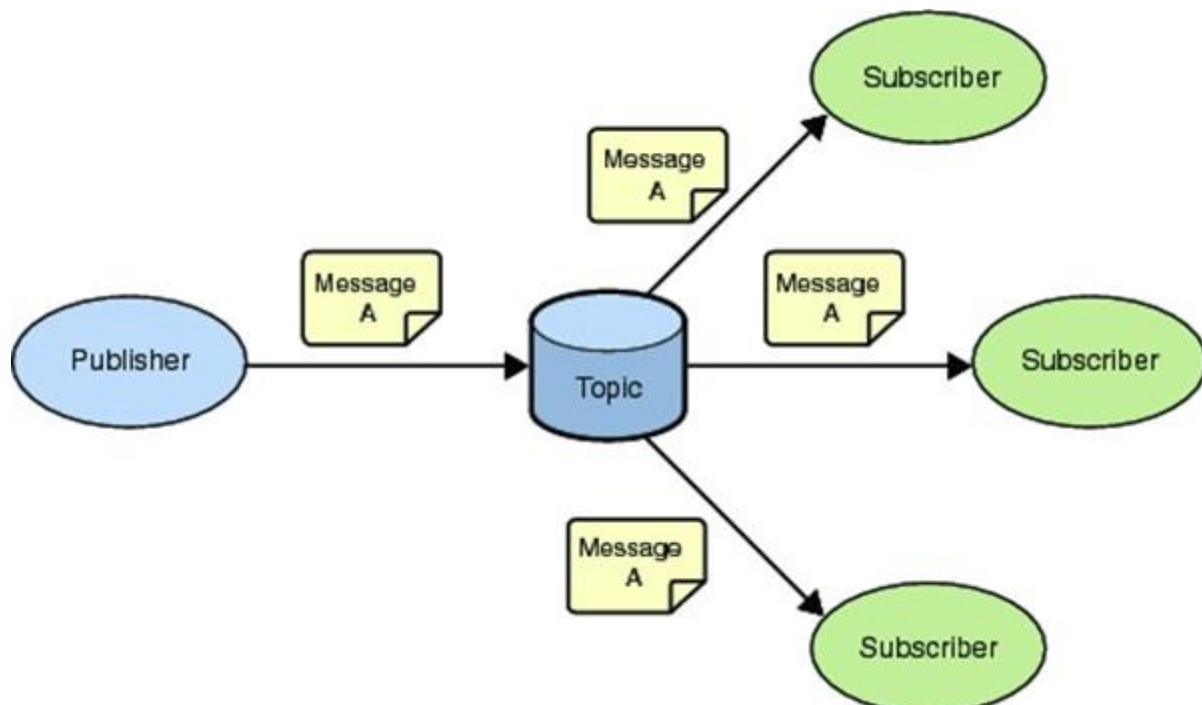
- Arquitecturas más complejas son construidas a partir del cliente/servidor.
- Por ejemplo, en las redes por pares (Peer-to-peer o P2P), todos los nodos de comunicación ejercen de cliente y servidor al mismo tiempo.



- En Java se utiliza la librería JXTA para crear redes basadas en arquitecturas P2P.

## Arquitectura de comunicación publicador/suscriptor

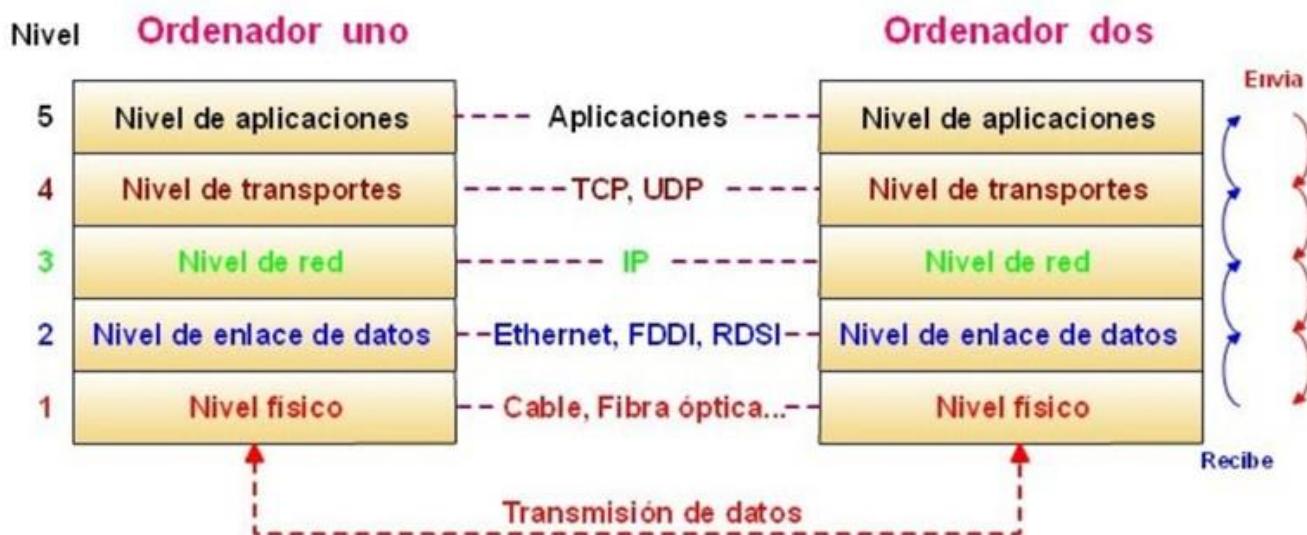
- Otra arquitectura importante que ha tenido gran auge en los últimos años es la Publicador/Suscriptor.



- En Java se utiliza la librería JMS para crear redes basadas en arquitecturas Publicador/Suscriptor.

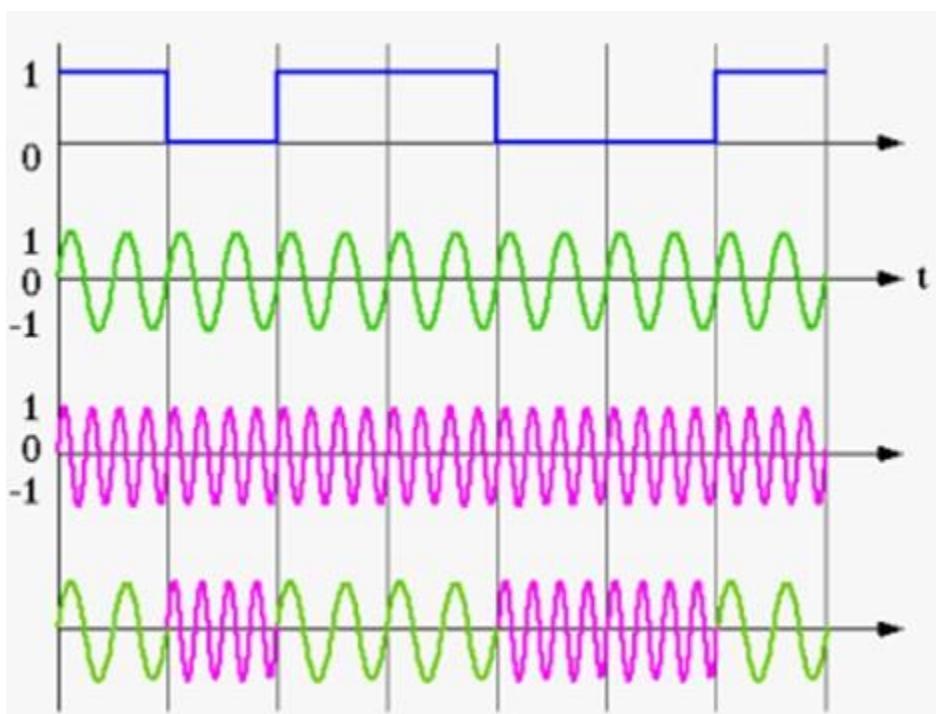
## Modelo de capas TCP/IP

- Independientemente de la arquitectura utilizada, la comunicación en Internet entre dos máquinas está gobernada por una serie de protocolos y estándares de comunicación agrupados en distintos niveles. Es el modelo de capas o pila de protocolos TCP/IP.



### Nivel físico

- Define la manera en la que los datos (ceros y unos) se convierten físicamente en señales digitales en los medios de comunicación (pulsos eléctricos, haz de luz, ondas electromagnéticas, etc.).

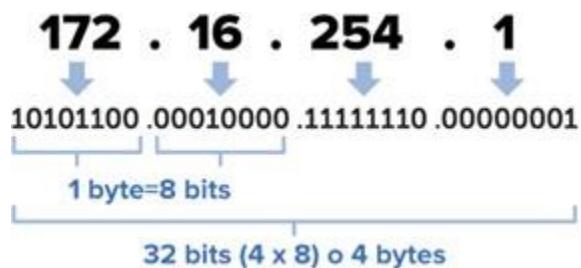


## Nivel de enlace de datos

- Define el modo de acceso al medio y cómo se transmiten los datos (frecuencia de operación, procedimiento ante colisiones, ancho de banda, potencia de emisión, etc).
  - El nivel de enlace es controlado por la interfaz de red.
  - Una interfaces de red es un dispositivo físico que permite establecer una comunicación:
    - Interfaz de red inalámbrico: Bluetooth, WiFi, LTE (4G), etc.
    - Interfaz de red alámbrico: tarjetas de red Ethernet, de fibra óptica, etc.
  - Cada interfaz de red posee una dirección única (dirección MAC) de 48 bits que la identifica a nivel mundial.
  - Las direcciones MAC se utilizan para establecer una comunicación entre dos máquinas que se encuentran conectadas físicamente de forma directa (por ejemplo, en una red de área local).
  - En Windows puede listarse toda la información sobre las interfaces de red (inalámbricas, alámbricas, privadas virtuales y virtuales escribiendo `ipconfig /all`

Nivel de red

- Se encarga de conducir los datos (enrutar) hacia el destino correspondiente en Internet a través de nodos intermedios.
  - El nivel de red también es el responsable de desensamblar los datos en pequeños fragmentos en la máquina origen y en ensamblarlos en la máquina destino.
  - Para que el enrutamiento se produzca es necesario que las interfaces de red que se comunican a través de Internet dispongan de una dirección IP (Internet Protocol address).
  - Las direcciones IP están compuestas por cuatro números (32 bits) separados por puntos y cuyos valores pueden variar entre 0 y 255.



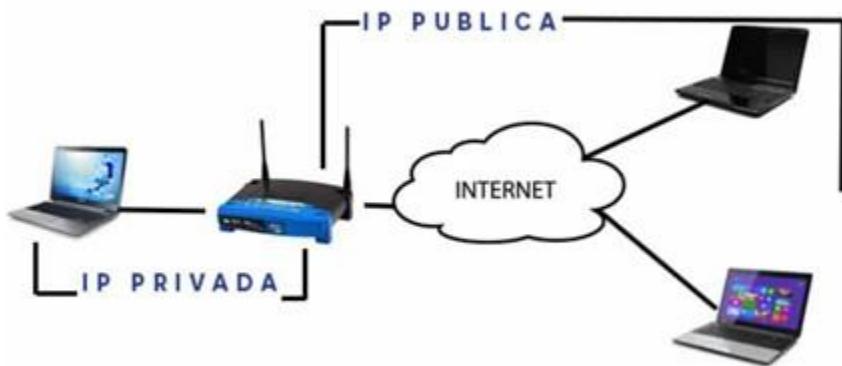
- El rango completo de direcciones IP abarca desde la dirección IP 0.0.0.0 a la 255.255.255.255.
- Existen direcciones especiales como la 127.0.0.1 (dirección de loopback o retorno), utilizada para realizar comunicaciones hacia el propio equipo.
- La dirección de loopback se utiliza principalmente para:
  - Probar el funcionamiento de la pila TCP/IP en el dispositivo (por ejemplo, mediante el comando ping).
  - Acceder a servicios de red de la propia máquina (por ejemplo, una aplicación web).

```
C:\Users\Alejandro>ping 127.0.0.1

Haciendo ping a 127.0.0.1 con 32 bytes de datos:
Respuesta desde 127.0.0.1: bytes=32 tiempo<1m TTL=128

Estadísticas de ping para 127.0.0.1:
  Paquetes: enviados = 4, recibidos = 4, perdidos = 0
              (0% perdidos),
  Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 0ms, Máximo = 0ms, Media = 0ms
```

- Las direcciones IP pueden clasificarse en públicas o privadas:
  - Una dirección IP privada identifica a una interfaz de red de un dispositivo que se encuentra dentro de una LAN (Local Area Network).
  - Una dirección IP pública es aquella que ofrece el proveedor de acceso telefónico para acceder a Internet.



- ◆ El rango de direcciones IP privadas son:
  - Para redes LAN grandes: desde la dirección 10.0.0.0 a 10.255.255.255
  - Para redes LAN medianas: desde la dirección 172.16.0.0. a 172.31.255.255
  - Para redes LAN pequeñas: desde la dirección 192.168.0.0 a 192.168.255.255
- ◆ El resto de direcciones IP son públicas (excepto aquellas que son especiales como 127.0.0.1).
- ◆ Habitualmente son los routers los que disponen de una dirección IP pública y se encargan de ofrecer conectividad a Internet a máquinas con direcciones IP privadas.
- ◆ Los routers van encaminando los datos hacia el destino correspondiente en Internet mediante protocolos de enrutamiento.

## Dirección IP de www.corenetworks.es

```
C:\Users\Alejandro>tracert 51.77.242.180
```

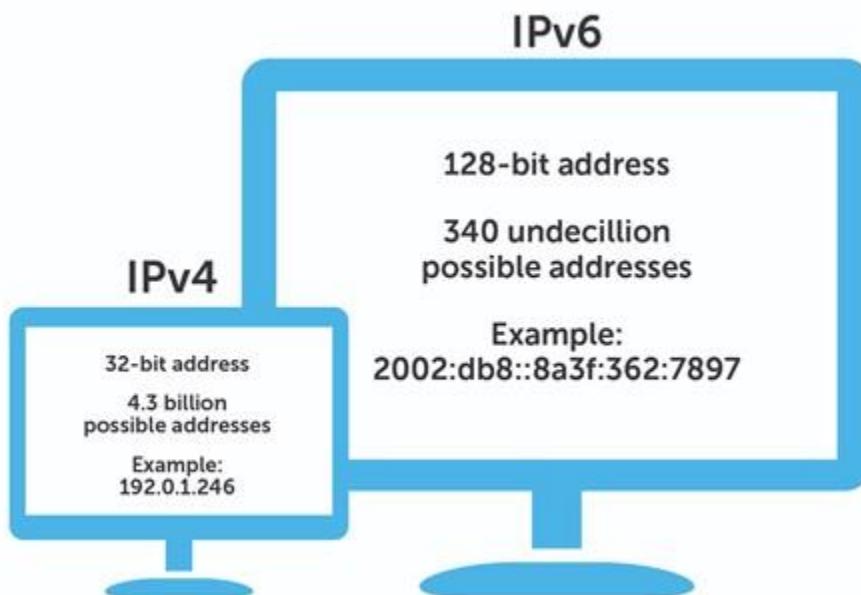
Traza a la dirección ip180.ip-51-77-242.eu [51.77.242.180]  
sobre un máximo de 30 saltos:

```

1    24 ms      5 ms      6 ms  LIVEBOXPLUS [192.168.1.1]
2    *          *          *      Tiempo de espera agotado para esta solicitud.
3  1393 ms    1246 ms    917 ms  10.255.140.186
4   809 ms     399 ms     96 ms  10.255.140.177
5  1445 ms    1229 ms   1137 ms  10.34.194.133
6   79 ms      18 ms     27 ms  10.34.35.186
7    *          *          *      Tiempo de espera agotado para esta solicitud.
8  1113 ms    1193 ms    726 ms  be100-1157.gsw-1-a9.fr.eu [91.121.131.153]
9   956 ms    1112 ms   1000 ms  be100.sbg-g2-nc5.fr.eu [91.121.215.218]
10   *          *          *      Tiempo de espera agotado para esta solicitud.
11   117 ms     240 ms    265 ms  be7.sbg-z2g2-a75.fr.eu [37.187.232.55]
12   283 ms      60 ms     46 ms  po7.sbg-z2b5-a70.fr.eu [92.222.57.118]
13    *          117 ms     54 ms  92.222.57.17
14    *          *          *      Tiempo de espera agotado para esta solicitud.
15  1118 ms    976 ms    994 ms  ip180.ip-51-77-242.eu [51.77.242.180]
```

Traza completa.

- Actualmente el número de direcciones IPv4 se ha agotado, aunque las subredes o NAT (Network Address Translation) han conseguido atrasar la adopción de IPv6. Las grandes empresas, multinacionales y proveedores de servicios sí han comenzado a migrar a direcciones IPv6.



- Pueden consultarse las direcciones IPv4 y IPv6 de todas las interfaces de red mediante ipconfig en Windows.

**Adaptador de Ethernet VMware Network Adapter VMnet1:**

Sufijo DNS específico para la conexión. . . :  
Vínculo: dirección IPv6 local. . . : fe80::1027:3820:3870:2d05%46  
Dirección IPv4. . . . . : 192.168.179.1  
Máscara de subred . . . . . : 255.255.255.0  
Puerta de enlace predeterminada . . . . :

**Adaptador de Ethernet VMware Network Adapter VMnet8:**

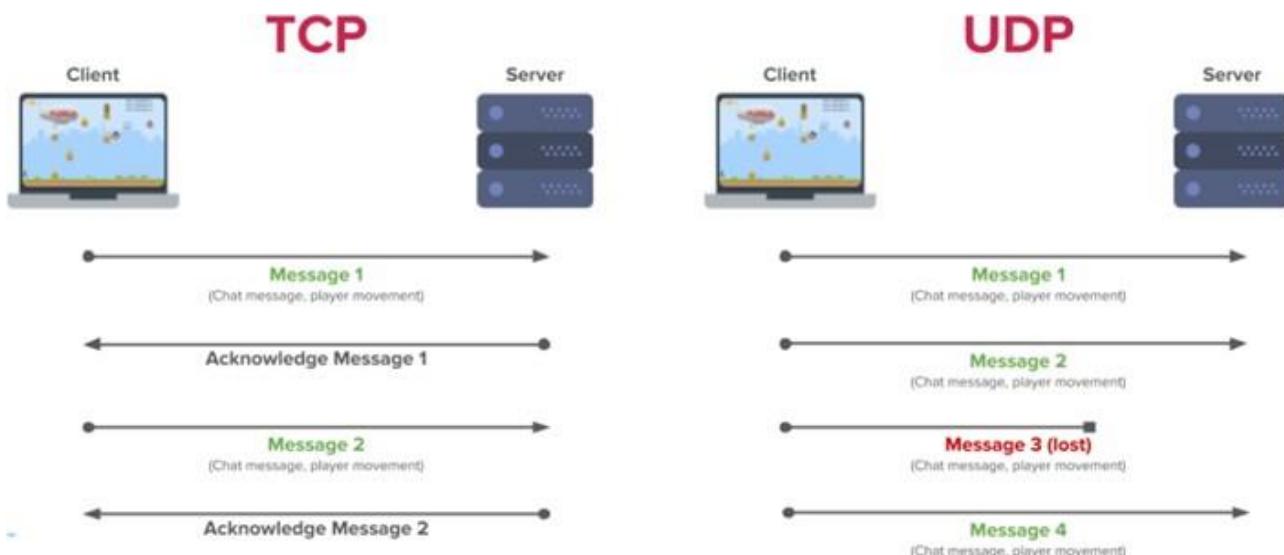
Sufijo DNS específico para la conexión. . . :  
Vínculo: dirección IPv6 local. . . : fe80::691e:16f6:96ea:e27%15  
Dirección IPv4. . . . . : 192.168.178.1  
Máscara de subred . . . . . : 255.255.255.0  
Puerta de enlace predeterminada . . . . :

**Adaptador de LAN inalámbrica Wi-Fi:**

Sufijo DNS específico para la conexión. . . :  
Vínculo: dirección IPv6 local. . . : fe80::87e:7cb1:7766:54fa%47  
Dirección IPv4. . . . . : 192.168.1.123  
Máscara de subred . . . . . : 255.255.255.0  
Puerta de enlace predeterminada . . . . : 192.168.1.1

**Nivel de transporte**

- ◆ Garantiza que los datos lleguen sin errores y de forma ordenada. Utiliza dos protocolos: TCP (orientado a conexión) y UDP (no orientado a conexión).
  - ◆ Al ser orientado a conexión, en TCP existe:
    - Establecimiento de la conexión.
    - Transferencia de datos con acuse de recibo.
    - Liberación de la conexión.
  - ◆ En cambio, en UDP los datos son enviados directamente sin establecer establecimiento, acuses de recibo o liberación de la comunicación.
  - ◆ UDP es más rápido, pero no posee mecanismos para detectar la pérdida de datos. UDP es utilizado en aplicaciones donde la pérdida de datos no supone un problema grave y es más importante la inmediatez (videojuegos, videoconferencia, VoIP o streaming).



- La capa de transporte también se encarga de que los datos se envíen a la aplicación adecuada dentro de la máquina que espera peticiones (servidor). El cliente también abre un puerto cuando se comunica con un servidor.

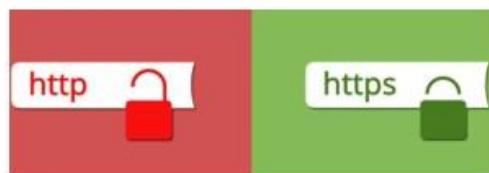
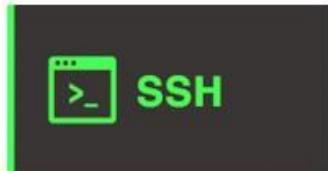


- Un puerto es un número comprendido entre 0 y 65535.
- Cualquier aplicación puede abrir un puerto TCP o UDP y esperar las peticiones entrantes de clientes.
- El rango de puertos puede dividirse en tres grupos.
  - Puertos bien conocidos (0-1023):
    - 21 (TCP): FTP.
    - 23 (TCP): Telnet.
    - 25 (TCP): SMTP.
    - 53 (UDP): DNS.
    - 69 (UDP): TFTP.
    - 80 (TCP): HTTP.
    - 110 (TCP): POP3.
    - 443 (TCP): HTTPS.
    - 520 (UDP): RIP.
    - 1812 (UDP): RADIUS.
    - 5060 (UDP): SIP.
  - Puertos registrados (1024-49151).

- Puertos privados o dinámicos (49152-65535).

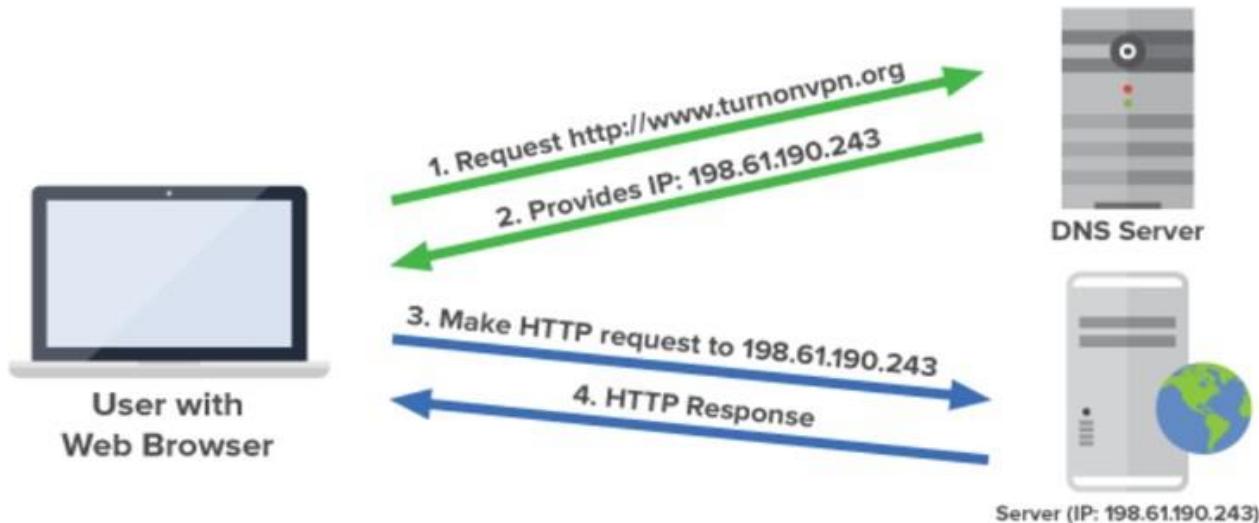
## Nivel de aplicación

- Nivel de aplicación: define los protocolos que utilizan las aplicaciones para intercambiar datos: correo electrónico (POP, IMAP y SMTP), gestores de bases de datos, gestores de transferencia de archivos (FTP), navegadores web (HTTP), resolución de nombres de dominio (DNS), etc.

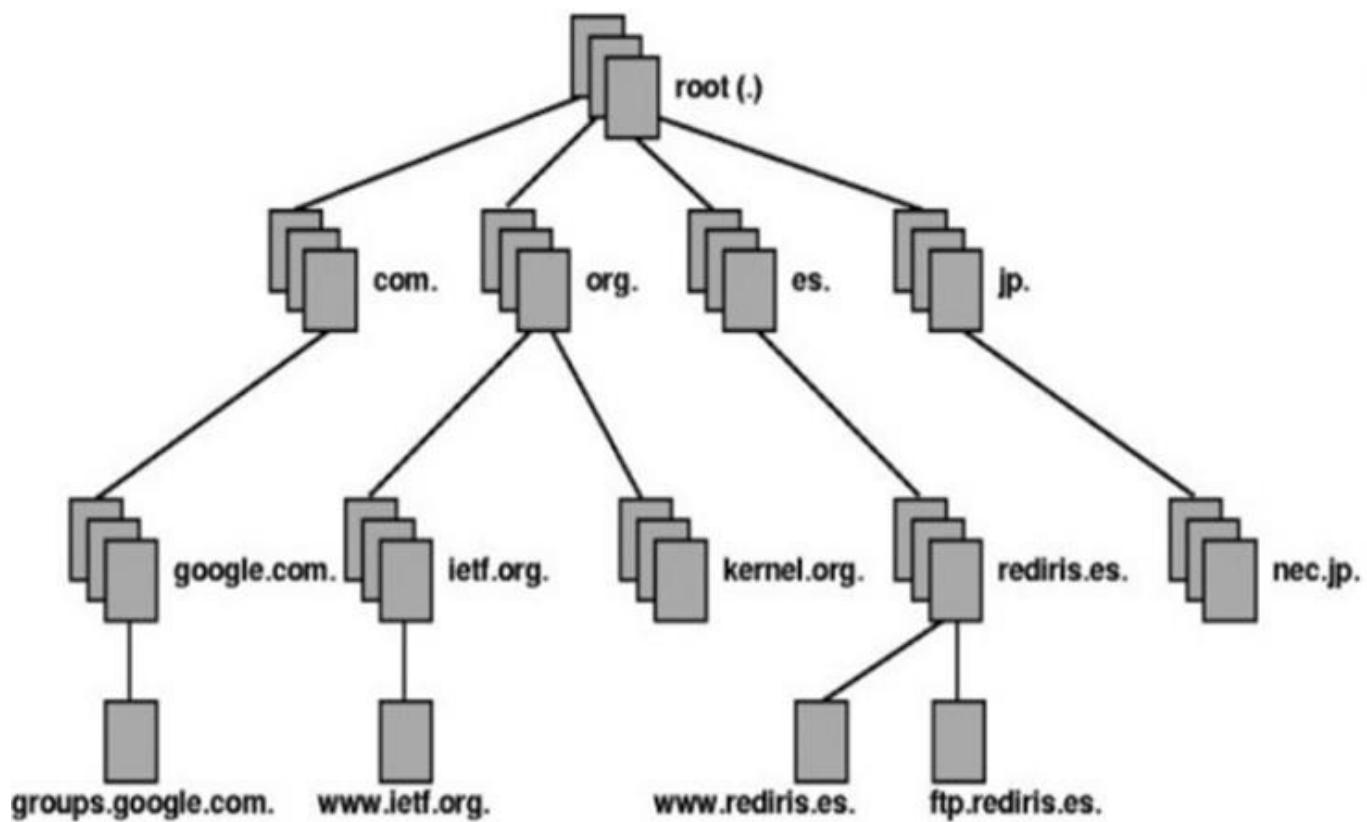


## El protocolo DNS

- El protocolo DNS se utiliza para asociar un nombre de dominio a un dirección IP. De esta forma se facilita a los usuarios el acceso a páginas mediante nombres de dominio y sin necesidad de recordar direcciones IP.



- Los nombres de dominio se clasifican en base una estructura jerárquica de subdominios.



- Herramientas como Spyse permiten obtener los subdominios de un determinado dominio.

**Spyse**

Tools API About Blog

Subdomains us.es

Filters	Results – 8,307	Download	General	DNS re
ASN – 6	acae.us.es	IP: 31.200.242.174	IP: 31.200.242.174	Title:
198096 (7,671)	Spain 60494	Geo: Spain	Geo: Spain	// There is no <title> tag on the site
766 (495)	Unelink Telecom, S.A.	AS Number: 60494	AS Organization: Unelink Telecom, S.A.	
+ more		Subdomains: 1	Subdomains: 1	
IP – 30+	gestioneventos.us.es - Eventos - Universidad de Sevi...			Meta description:
150.214.230.47 (55)	IP: 87.253.228.167 Spain 48846			// There is no meta-description on the site
150.214.9.141 (15)	Informatica El Corte Ingles SA			
+ more				
CIDR – 30+	dircom9.us.es			
150.214.130.0/24 (2)	IP: 150.214.9.1 Spain 198096			
150.214.143.0/24 (2)	Junta de Andalucía			
+ more				
Organization – 6	vpncclient9.us.es			
	IP: 150.214.9.5 Spain 198096			
	Junta de Andalucía			

Subdomains

Total – 1

Search   Show only with IP

- Para obtener la dirección IP de un determinado nombre de dominio o subdominio puede utilizarse el comando *nslookup* o mediante *ping*.

```
C:\Users\Alejandro>nslookup corenetworks.es
Servidor: dns.google
Address: 8.8.8.8
```

Respuesta no autoritativa:  
 DNS request timed out.  
 timeout was 2 seconds.  
 Nombre: corenetworks.es  
 Address: 91.142.213.206

```
C:\Users\Alejandro>nslookup www.corenetworks.es
Servidor: dns.google
Address: 8.8.8.8
```

Respuesta no autoritativa:  
 Nombre: www.corenetworks.es  
 Address: 51.77.242.180

```
C:\Users\Alejandro>ping corenetworks.es
```

Haciendo ping a corenetworks.es [91.142.213.206] con 32 bytes de datos:  
 Tiempo de espera agotado para esta solicitud.  
 Tiempo de espera agotado para esta solicitud.  
 Tiempo de espera agotado para esta solicitud.  
 Tiempo de espera agotado para esta solicitud.

Estadísticas de ping para 91.142.213.206:  
 Paquetes: enviados = 4, recibidos = 0, perdidos = 4  
 (100% perdidos),

```
C:\Users\Alejandro>ping www.corenetworks.es
```

Haciendo ping a www.corenetworks.es [51.77.242.180] con 32 bytes de datos:  
 Respuesta desde 51.77.242.180: bytes=32 tiempo=38ms TTL=48  
 Respuesta desde 51.77.242.180: bytes=32 tiempo=38ms TTL=48  
 Respuesta desde 51.77.242.180: bytes=32 tiempo=40ms TTL=48  
 Respuesta desde 51.77.242.180: bytes=32 tiempo=39ms TTL=48

Estadísticas de ping para 51.77.242.180:  
 Paquetes: enviados = 4, recibidos = 4, perdidos = 0  
 (0% perdidos),  
 Tiempos aproximados de ida y vuelta en milisegundos:  
 Minimo = 38ms, Máximo = 40ms, Media = 38ms

## El protocolo HTTP

- HTTP es un protocolo basado en TCP para la transferencia de datos en la Web.
- Utiliza diferentes métodos de petición o verbos HTTP: GET, POST, PUT, DELETE,...
- Además de los datos, el servidor devuelve un código de respuesta: 200 (OK), 404 (No encontrado), 501 (Error interno en el servidor), etc.



- Entre las características de HTTP se encuentran:
  - En HTTP El cliente es quien solicita los recursos al servidor.
  - HTTP fue concebido inicialmente para transferencia de páginas web pequeñas.
  - HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.
  - HTTP realiza múltiples conexiones para recibir datos. Esto ralentiza la comunicación porque TCP es lento.
  - HTTP es un protocolo inseguro.
- Todas las peticiones HTTP están compuestas por:

- La URI (Identificador de Recursos Uniforme) que identifica la ubicación al recurso web.
- El método HTTP: GET o POST son los más importantes.
- Cabeceras HTTP: permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. Una cabecera está compuesta por su nombre seguido de dos puntos (:) y, a continuación, su valor.
- El cuerpo de la respuesta: donde generalmente se encuentran los datos

```
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; ... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,...,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345
```

Request headers (red background)

General headers (green background)

Entity headers (blue background)

- El método GET de HTTP permite solicitar información de un recurso a un servidor a partir de un URI (Identificador de Recursos Uniforme).
- Un URI es una cadena de caracteres que identifica a un recurso de una red de forma única. Está formado por las siguientes partes:
  - Protocolo a nivel de aplicación: http:, https:, mailto:, ftp:, etc.
  - Autoridad: identifica el nombre de dominio o dirección IP del servidor: Ejemplo: //servidor.com
  - Puerto: punto de entrada al recurso. Ejemplo: 80
  - Ruta: identifica la ubicación del recurso dentro del servidor. Ejemplo: /madrid/usuarios/
  - Consulta: permite parametrizar la petición mediante pares clave/valor. El comienzo de este componente se indica mediante el carácter del símbolo de interrogación (?) y pueden enviarse varios parámetros concatenando mediante el carácter &. Ejemplo: ?x=5&y=2
- El método GET de HTTP se utiliza cuando se accede a una página web a partir de la barra de direcciones de un navegador web.



<https://www.marca.com/>

<http://servicioweb.com/v1/usuarios?usuario=juan>

<http://servicioweb.com/v1/temperatura?fecha=06-03-2015>

- En las peticiones HTTP con el método GET, la URI tiene una limitación de 2048 caracteres como máximo. Para enviar información de mayor tamaño se utiliza el método POST.
- Con POST se pueden insertar los datos que se envían dentro del cuerpo del paquete HTTP (sin limitaciones de tamaño). Con GET también se puede enviar datos de esta forma, pero está desaconsejado.
- Los URI no se cifran nunca cuando se envían al servidor, por lo que para el envío de datos sensibles debería utilizarse POST.
- Las peticiones tipo POST no pueden realizarse directamente a través de la barra de direcciones de un navegador web, sino que es necesario utilizar programación o clientes HTTP específicos como [Postman](#).

```
GET /academy/courses?id=java HTTP/1.1
Host: www.telerik.com
Accept: /*
Accept-Language: bg
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
```

```

POST /webmail/login.phtml HTTP/1.1
Host: www.abv.bg
Accept: /*
Accept-Language: bg
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 59
Body:
<CRLF>
LOGIN_USER=mente
DOMAIN_NAME=abv.bg
LOGIN_PASS=top*secret!
<CRLF>

```

The screenshot shows the Postman interface with the following details:

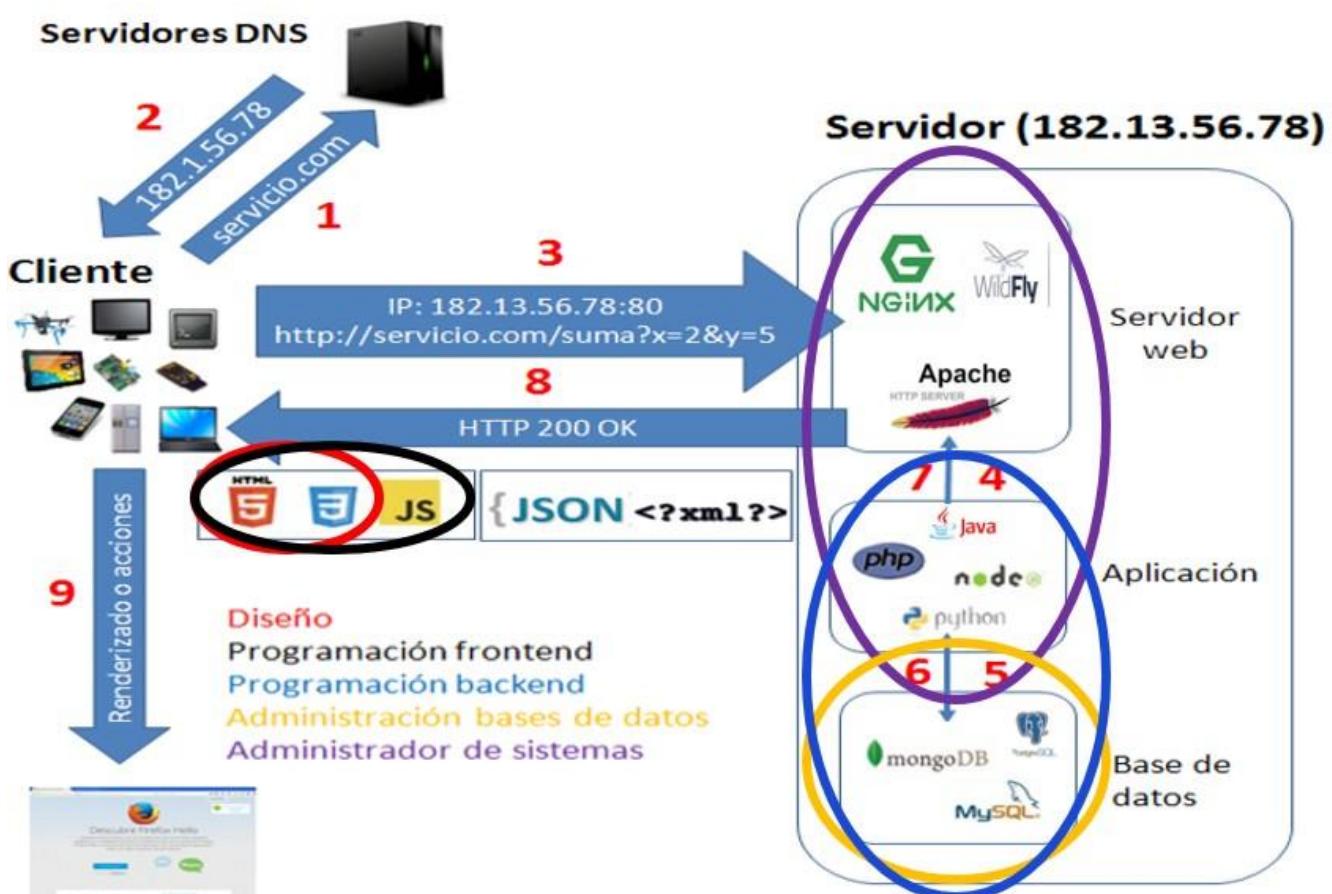
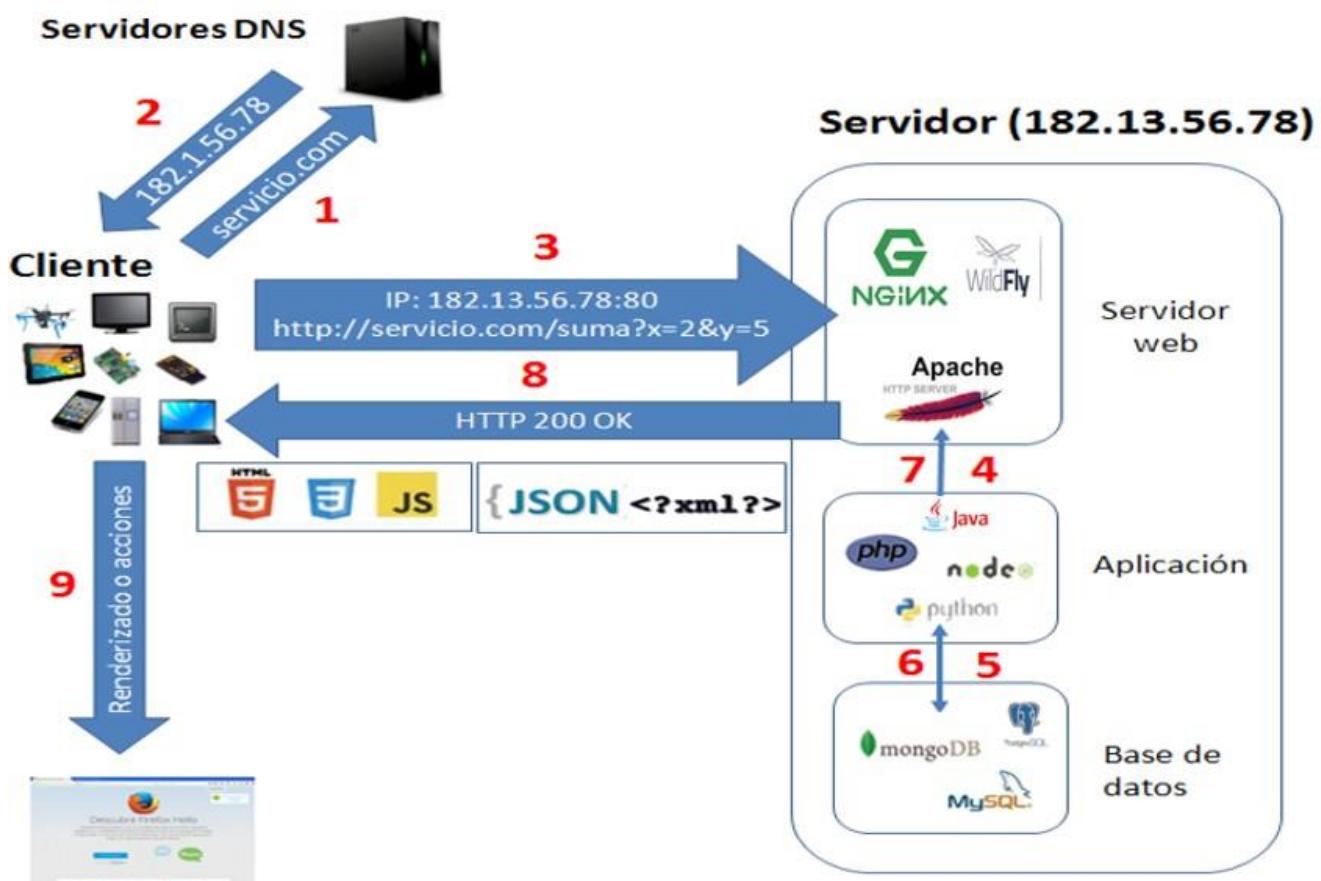
- Request Method:** GET
- URL:** https://swapi.co/api/films/2/
- Params:** a: b
- Body:** (Pretty) JSON response content:

```

1  {
2   "title": "The Empire Strikes Back",
3   "episode_id": 5,
4   "opening_crawl": "It is a dark time for the\nRebellion. Although the Death\nStar has been destroyed,\nImperial troops have driven the\nRebel forces from their\nhidden\nbase and pursued them across\nthe galaxy.\n\nInvasive the dreaded Imperial\nStarfleet, a group of freedom\nfighters led by Luke\nSkywalker\nhas established a new secret\nbase on the remote ice world\nof Hoth.\n\nThe evil lord Darth Vader,\nobessed with finding\nyoung\nSkywalker, has dispatched\nthousands of remote probes into\nthe far reaches of space....",
5   "director": "Irvin Kershner",
6   "producer": "Gary Kurtz, Rick McCallum",
7   "release_date": "1980-05-17",
8   "characters": [
9     "https://swapi.co/api/people/1/",
10    "https://swapi.co/api/people/2/",
11    "https://swapi.co/api/people/3/",
12    "https://swapi.co/api/people/4/",
13    "https://swapi.co/api/people/5/",
14    "https://swapi.co/api/people/10/"
  ]

```
- Status:** 200 OK
- Time:** 2.13s
- Size:** 2.7 kB

## Flujo de comunicación en tecnología web



Tu carrera digital ~

# Módulo 1

# Fundamentos del

# desarrollo web

Git



# Básico

- ◆ [Introducción](#)
- ◆ [Instalación](#)
- ◆ [Iniciando con git](#)
- ◆ [Comprobar el estado del repositorio](#)
- ◆ [Moverse entre commits](#)
- ◆ [Diferencias entre git rm y git reset](#)
- ◆ [Clonar un repositorio](#)
- ◆ [Interfaces gráficas](#)

## Introducción

- ◆ Los sistemas de control de versiones (VCS) son herramientas que permiten registrar el versionado de código de un proyecto software.
- ◆ También facilita la colaboración con otros desarrolladores en un proyecto de software sin peligro de sobreescribir el trabajo de los demás.
- ◆ Entre las ventajas de los VCS se encuentran:
  - Un completo historial de los cambios de todos los archivos del proyecto en todo el tiempo, incluyendo creación, modificación y eliminación.
  - Creación de ramas y fusiones: permite generar flujos de trabajo independientes que no interfieren y que pueden fusionarse con posterioridad.
  - Trazabilidad: quién, cuándo y qué cambios se hicieron.
- ◆ El sistema de control de versiones más popular actualmente es Git, creado por Linus Torvalds en el 2005.
- ◆ Git se considera un VCS distribuido (DVCS) porque la copia del código de desarrollo no solamente se encuentra en un repositorio remoto, sino también en el propio equipo del programador.
- ◆ Por otro lado, GitHub es una web de desarrollo colaborativo que permite alojar proyectos utilizando Git, además de proveer otras utilidades adicionales que ayudan a los programadores.

## Instalación

- ◆ Git puede descargarse desde su [página web oficial](#)
- ◆ En la instalación de Git, también se instala una consola especial denominada Git Bash (similar a la terminal de Linux, pero adaptada al sistema operativo Windows) y también Git GUI para facilitar el trabajo a los programadores.

- ◆ Durante la instalación se ofrece multitud de opciones. Generalmente las establecidas por defecto suelen ser las recomendadas. Las más importantes son:
  - *Git Bash Here* y *Git GUI Here*: añade estas opciones al menú contextual (botón derecho) del explorador en Window. Por defecto estas opciones se encuentran activadas.
  - *Check daily for Git for Windows Update*: permite comprobar actualizaciones de Git desde Windows Update. Por defecto esta opción se encuentra desactivada.
  - *Add a Git Bash Profile to Windows Terminal*: agrega el perfil de Git Bash a Windows Terminal (wt). Por defecto esta opción se encuentra desactivada.
  - *Choosing the default editor used by Git*: editor por defecto para manipular archivos de texto. Por defecto el editor de texto es *vim*
  - *Adjusting the name of the initial branch in new repositories*: nombre de la rama principal de los nuevos repositorios en Git. Por defecto el valor es *Let Git decide*, siendo *master* el valor tomado. Sin embargo, es previsible que este nombre cambie en un futuro por temas de inclusividad. Posibles elecciones: *main*, *trunk* y *development*
  - *Adjusting your PATH environment*: permite indicar desde dónde podrá utilizarse el comando *git*. Por defecto el valor es *Git from the command line and also from 3rd-party software*, pudiendo utilizarse desde todas las líneas de comando disponibles (Git Bash, terminal de Windows y Powershell) y otras herramientas de terceros, dado que el directorio de Git se añadirá en la variable de entorno PATH.
  - *Choosing the SSH executable*: ejecutable de SSH que utilizará Git. Por defecto el valor es *\*Use bundled OpenSSH*, por lo que Git utilizará su cliente interno SSH que incorpora.
  - *Choosing HTTPS transport backend*: librería para el manejo de protocolos SSL/TLS utilizados por HTTPS. Por defecto el valor es *\*Use the OpenSSL Library*, por lo que Git utilizará la librería interna que incorpora.
  - *Configuring the line ending conversions*: permite indicar cómo debe tratar Git los finales de los archivos. Por defecto el valor es *"Checkout Windows-style, commit Unix-style line endings"*, por lo que Git convertirá LF (salto de línea, habitual en Linux) a CRLF (retorno de carro y salto de línea, habitual en Windows) en tareas de *checkout* y realizará el proceso inverso en tareas de *commit*. Es recomendable cuando los programadores trabajan en distintos sistemas operativos. Cuando todos trabajan en Linux es mejor utilizar LF.
  - *Configuring the terminal emulator to use with Git Bash*: permite configurar el emulador del terminal que será utilizado por Git Bash. Por defecto el valor es *MinTTY (the default terminal of MSY2)*, la colección de herramientas proporcionadas por *Cygwin*
  - *Choose the default behavior of git pull*: comportamiento de la instrucción *git pull*. Por defecto el valor es *fast-forward* o *merge*
  - *Choose a credential helper*: permite configurar el servicio de autenticación contra servicios como Gitlab o Github. Por defecto el valor es *Git Credential Manager Core*
  - Otras configuraciones adicionales y experimentales: recomendable configurarlas por defecto.

- En Linux es muy fácil instalar Git.

```
# Distribuciones derivadas de Debian  
sudo apt-get instal git
```

```
# Distribuciones derivadas de Red Hat  
sudo yum install git
```

- Finalmente puede comprobarse si la instalación fue correcta solicitando a *git* su versión actual.

```
git --version
```

- En ocasiones es necesario crear una SSH también para conexión a servidores remotos.

```
# la clave pública privada se almacenan normalmente en el directorio .ssh del usuario  
ssh-keygen -t rsa -b 4096 -C "email@example.com"
```

- A continuación es importante establecer una configuración inicial para Git (al menos un nombre un correo electrónico del usuario).

```
git config --global user.name "Alejandro"  
git config --global user.email "talaminos@gmail.com"
```

- El cambio de valores de otros parámetros de configuración también se realiza de forma análoga.

```
git config --global color.ui true  
git config --global core.editor vim
```

- También se puede consultar la configuración actual.

```
git --list
```

- Y dónde se almacena el archivo de configuración. Generalmente existen tres rutas:
  - Individuales (local): ajustes específicos del repositorio (con --local o simplemente sin utilizar ningún parámetro). Se almacena en ./git/config

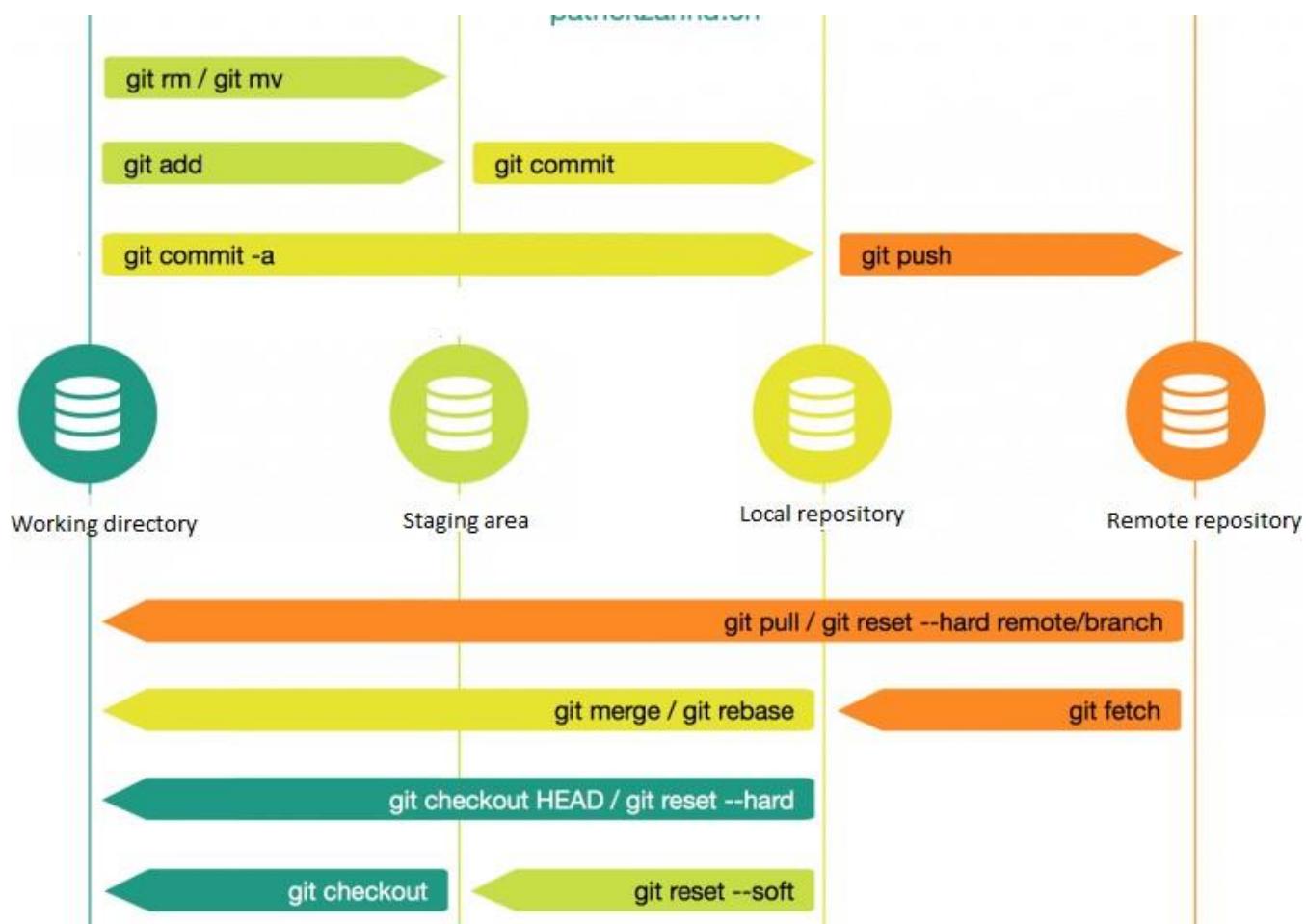
- Usuarios (global): ajustes específicos del usuario (con --global). Se almacena en `~/.gitconfig`
- Todo el sistema (sistema): ajustes de todo el sistema (con --system). Se almacena en `/etc/gitconfig`

```
git config --list --show-origin
```

- ♦ Git también permite el uso de alias

## Iniciando con *git*

- ♦ Antes de instalar git, únicamente era conocer de la existencia del directorio del proyecto, también conocido como *Working Directory* (espacio de trabajo), donde se encuentran los archivos y directorios de un proyecto software.
- ♦ Cuando se utiliza *git* es necesario conocer dos más:
  - *Staging Area* (área de ensayo): es un espacio de memoria intermedia o temporal (similar a la RAM de un ordenador) donde los archivos se preparan antes de convertirse en parte de una versión del proyecto (mandarlos al repositorio).
  - *Repository* (repositorio): es un espacio de memoria donde se almacenan cada una de las versiones del proyecto (ya sea en local o en remoto)



- ◆ Este conjunto de espacios en memoria son gestionados por Git desde su directorio interno .git, creado al inicializar un repositorio nuevo.
  - El directorio .git incluye los metadatos del repositorio, incluyendo subdirectorios de objetos, referencias y archivos de plantilla, ...
  - La ejecución de *git init* no tendrá efecto en un directorio que ya sea un repositorio de Git.

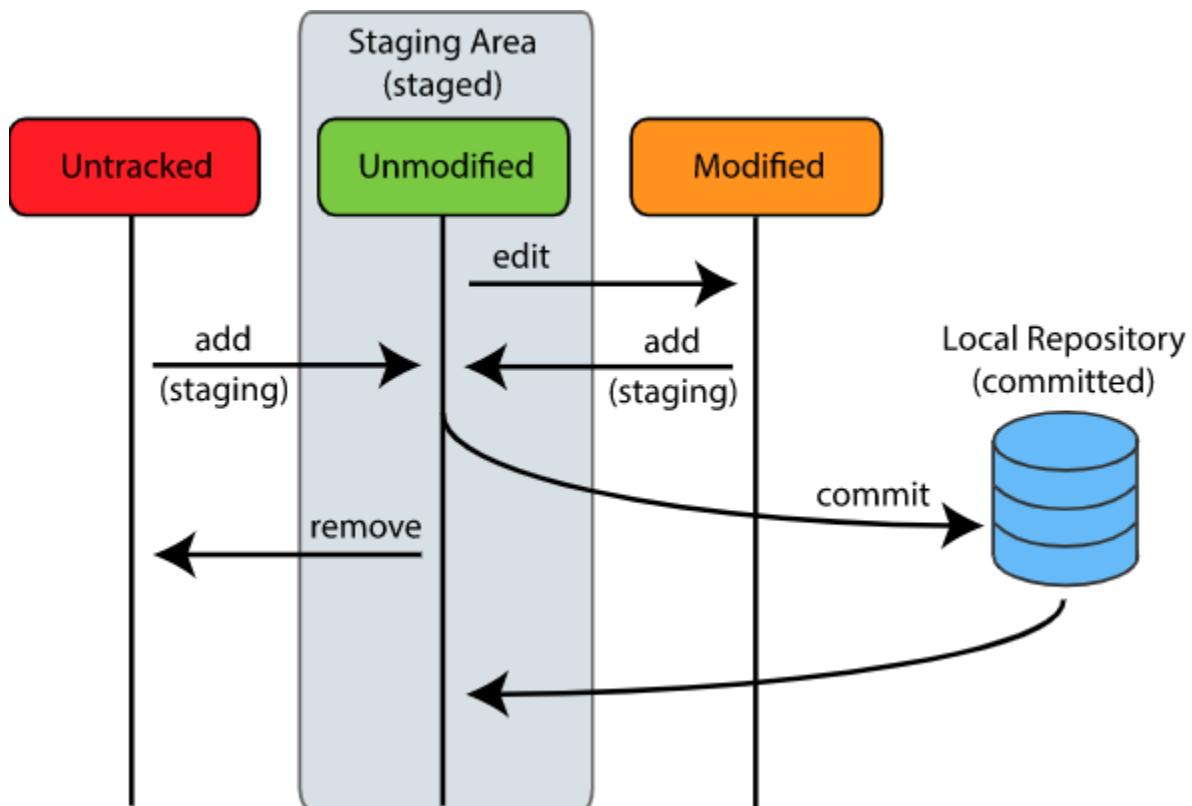
```
# inicializa un repositorio de Git nuevo  
git init
```

- ◆ Al iniciar, todos los archivos se encuentran en el espacio de trabajo y están sin ser monitorizados (*untracked*)
- ◆ Para que comiencen a ser monitorizados (*tracked*) es necesario llevarlos al *Staging area*

```
# lleva el archivo README.md al Staging Area  
git add README.md
```

```
# lleva todos los archivos al Staging Area (similar a git add --all)  
git add .
```

- ◆ Un archivo monitorizado (*tracked*) puede encontrarse en dos estados:
  - *Unmodified*: no se ha modificado desde la última vez que se añadió al *Staging Area* mediante *git add*
  - *Modified*: sí se ha modificado desde la última vez que se añadió al *Staging Area* mediante *git add*
- ◆ Si el archivo ha sido modificado, entonces tiene que volver a ejecutarse el comando *git add* para convertirlo de nuevo al estado *Unmodified*, como paso previo al envío al repositorio (*commit*).



- Seguidamente se envían los archivos al repositorio mediante un *commit*, donde es obligatorio el parámetro -m para añadir una descripción entre comillas dobles del commit realizado.

```
git commit -m "Primer commit"
```

- Las instrucciones *git add .* y *\*git commit -m ...\** pueden ser simplificadas en una única instrucción.

```
# similar a git add . seguido de git commit -m "Primer commit"
git commit -m "Primer commit" -a
```

- Una vez realizado el *commit*, los archivos permanecen en el estado *Unmodified* dentro del *Staging Area*
- Un archivo monitorizado (*Tracked*) puede volver a dejar de estarlo (*Untracked*) mediante *git rm*

```
git rm --cached README.md
```

## Comprobar el estado del repositorio

- Toda la información de los commits realizados puede consultarse mediante `git log`

```
# historial de todos los commits realizados  
git log
```

```
# historial de todos los commits realizados para un archivo en particular  
git log README.md
```

```
# historial de todos los commits realizados y los cambios específicos en cada archivo  
git log --stat
```

- La información más importante de cada commit es:

- Tag: cadena de caracteres que identifica de forma única al commit.
- Fecha: en la que se realizó el commit.
- Autor: persona que realizó el commit.
- Descripción del commit.
- HEAD: si el commit lleva este indicativo, entonces se trata del último commit de la actual rama (por defecto, la rama *master* si solamente hay una).

- El estado actual de los archivos puede consultarse con `git status`

```
git init
```

```
# no hay commits y se informa de que existen archivos no monitorizados  
git status
```

```
git add .
```

```
# no hay commits y se informa de que existen cambios que pueden confirmados  
(commit). También indica como convertir archivos monitorizados a no monitorizados  
git status
```

```
git commit -m "Primer commit"
```

```
# indica que no hay nada para confirmar  
git status
```

```
# algún archivo editado
```

```
# indica que hay nuevos cambios que pueden añadirse con git add y git commit.  
También indica que los cambios pueden ser revertidos al anterior commit  
git status
```

```
git commit -m "Segundo commit" -a
```

```
# algún archivo editado (por ejemplo, README.md)
```

```
# indica que hay nuevos cambios que pueden añadirse con git add y git commit.  
También indica que los cambios pueden ser revertidos al anterior commit  
git status
```

```
# reestablece el contenido del archivo README.md a su estado en el último commit  
git restore README.md
```

```
# algunos archivo editados
```

```
# reestablece todos los archivos modificados al estado del último commit  
git .
```

- También se puede analizar el historial de un archivo en particular, incluyendo descripciones del commit, diferencias de tamaño, diferencias de los archivos entre commits, ... (no aparece el contenido)

```
git show README.md
```

- La diferencia entre dos commits puede comprobarse mediante la instrucción *git diff*, pasando por parámetros los identificadores o tags de los commits a comparar.

```
git diff eb5.. 1a4..
```

- El comando \*git diff también permite mostrar los cambios realizados desde el último commit.

```
git diff
```

## Moverse entre commits

- Para alcanzar el estado de un determinado commit, se utiliza la instrucción *git checkout*. Es fundamental que antes de realizar esta acción no hay ningún archivo pendiente de confirmación (commit)

```
# todos los archivos vuelven al estado en el que se encontraban en el commit 1a45...  
git checkout 1a45...
```

```
# el HEAD se encuentra ahora en el commit 1a45  
git log
```

```
# hay archivos que pueden ser confirmados (commit)  
git status
```

```
# todos los archivos regresan al estado del último commit  
git checkout master
```

```
# ya no hay archivos que puedan ser confirmados (commit)  
git status
```

```
# el HEAD se encuentra ahora en el último commit  
git log
```

- También se puede utilizar la instrucción *git checkout* para archivos específicos

```
# el archivo README.md vuelve al estado en el que se encontraba en el commit 1a45...  
git checkout 1a45 README.md
```

```
# el archivo README.md regresa al estado del último commit  
git checkout master README.md
```

## Diferencias entre *git rm* y *git reset*

- *git rm* generalmente elimina los archivos que se encuentran monitorizados en el *Staging Area* y los convierte en no monitorizados. Tiene dos variantes:
  - *git rm ARCHIVO --cached*: elimina los archivos monitorizados del *Staging Area*
  - *git rm ARCHIVO --force*: elimina los archivos monitorizados del *Staging Area* y también el disco duro (muy peligroso). Sin embargo, todavía podrían recuperarse de un commit anterior.
- *git reset* es similar a *git checkout* pero más peligroso, porque un salto hacia otro commit es irreversible. Tiene dos variantes:
  - *git reset --soft TAG\_COMMIT*: borra todo todo el historial de Git hasta el commit establecido, pero mantiene los cambios en el *Staging Area*
  - *git reset --hard TAG\_COMMIT*: borra todo todo el historial de Git hasta el commit establecido, incluyendo los cambios en el *Staging Area*. Esta variante es más utilizada que *git reset --soft*

```
# MUY PELIGROSO: borra todos los archivos del Staging Area y del disco duro (de forma recursiva)
```

```
rm . -r --force
```

```
# recupera los datos al último commit  
git reset --hard HEAD
```

```
# todos los archivos regresan al estado de un determinado commit y los cambios son irreversibles
```

```
git reset --hard TAG_COMMIT
```

```
# comprueba los cambios realizados
```

```
git log
```

## Clonar un repositorio

- Si un repositorio ya está creado, puede clonarse desde otra ubicación con *git clone*

```
git clone URL_REPOITORIO
```

- La clonación del repositorio descargará todos los archivos en su última versión de su rama principal, incluyendo también el directorio .git donde se encuentra toda la configuración del repositorio.
- Git trabaja con múltiples protocolos de red y sus diferentes formatos de URL, incluyendo git (similar a ssh pero sin autenticación), ssh o https.
- El clonado de un repositorio remoto también incluye la vinculación automática (llamada *origin*) hacia el repositorio original mediante *git remote*, por lo que es posible realizar operaciones de push y pull si se tienen los permisos correspondientes.
- Por defecto, la creación de un repositorio inicial con *git init* no incluye una vinculación con un repositorio remoto (aunque puede añadirse con posterioridad).

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend (HTML)

HTML5 (I)



# HTML5 (I)

- ◆ [Introducción](#)
- ◆ [Historia de HTML](#)
- ◆ [Estructura de una página HTML5](#)
- ◆ [Codificación de caracteres](#)
- ◆ [Caracteres especiales](#)
- ◆ [Etiquetas básicas](#)
- ◆ [Formatear texto](#)
- ◆ [Iframes](#)
- ◆ [Enlaces](#)

## Introducción

- ◆ HTML (HyperText Markup Language) surge en el año 1991 como un lenguaje de marcado (de etiquetas) estándar para crear páginas web.
- ◆ HTML describe la estructura de las páginas web mediante elementos HTML.
- ◆ Los elementos HTML son fragmentos de contenido como listas, tablas o formularios.
- ◆ Un elemento HTML está representado por un par de etiquetas (una de apertura y otra de cierre). Las etiquetas HTML llevan un nombre y están rodeados de signos de mayor o menor que:

```
<tagname>contenido</tagname>
```

- ◆ La primera etiqueta es la de apertura y la segunda etiqueta, la de cierre.
- ◆ La etiqueta de cierre se escribe como la etiqueta de inicio, pero con una barra inclinada insertada antes del nombre de la etiqueta.
- ◆ Los navegadores no muestran las etiquetas HTML, pero las usan para representar el contenido de la página.
- ◆ Tipos de etiqueta:

```
<tagname>contenido</tagname>    <!-- Con contenido -->
<tagname></tagname>           <!-- Vacías -->
</tagname>                   <!-- Sin etiqueta de apertura -->
```

- ◆ Las etiquetas pueden poseer también atributos, que añaden información adicional a la etiqueta. Los atributos siempre se establecen en la etiqueta de inicio y su valor se

encierra entre comillas dobles.

```
<tagname attribute="value">contenido</tagname>
```

- Las etiquetas pueden estar anidadas:

```
<tagname1>
  <tagname2></tagname2>
</tagname1>
```

- Todas las etiquetas deberían cerrarse correctamente.
- Las etiquetas no son case sensitive, aunque es recomendable siempre utilizar minúsculas, tal y como recomienda la W3C.

## Historia de HTML

Version	Año
Tim Berners-Lee inventa www	1989
Tim Berners-Lee inventa HTML	1991
Dave Raggett drafted HTML+	1993
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML 1.0	2000
HTML5 Draft	2008
HTML5 Living Standard (WHATWG)	2012
W3C Recommendation: HTML5	2014
W3C Candidate Recommendation: HTML 5.1	2016
W3C Candidate Recommendation: HTML 5.2	2017
W3C Candidate Recommendation: HTML 5.3	2018
Living Standard (WHATWG)	Actual

- El estándar HTML avanzó en sus primeros años gracias a los esfuerzos no solamente de la World Wide Web Consortium (W3C), sino también de la Web Hypertext Application Technology Working Group (WHATWG).

- Aunque inicialmente trabajaban juntas, en julio de 2012, WHATWG y W3C decidieron separarse. El W3C continuó el trabajo de la especificación de HTML5, centrándose en un único estándar definitivo, que es considerado como un snapshot por WHATWG. La organización WHATWG continuó su trabajo con HTML5 como Living Standard (estándar de por vida).
  - El concepto de un estándar de vida es que nunca está completo y siempre se actualiza y mejora. Se pueden agregar nuevas características, pero no se eliminan las existentes.
  - W3C prefiere utilizar versiones en HTML.
- El W3C anunció el 28 de mayo de 2019 que WHATWG sería el único editor de los estándares HTML y del DOM, siguiendo el proceso de especificación de Living Standard. El consorcio W3C tiene la intención de aportar su opinión y respaldar los borradores en revisión con la WHATWG para incluirlos también como estándares de la W3C.
- El antecesor de HTML5 es XHTML (y el antecesor de XHTML es HTML4):
  - XHTML significa Lenguaje de Marcado de Hipertexto Extensible:
  - XHTML es casi idéntico a HTML.
  - XHTML es más estricto que HTML (más requerimientos respecto a la validación):
    - XHTML DOCTYPE es obligatorio.
    - Las etiquetas html, head, title, body son obligatorias.
    - Las etiquetas deben estar correctamente anidadas y cerradas (incluido etiquetas como br y similares).
    - Las etiquetas deben estar en minúscula siempre, con un único elemento raíz, atributos en minúscula, valores de los atributos obligatorios y valores de los atributos entre comillas dobles y en minúsculas.
  - En general, las reglas de XHTML deberían ser consideradas en HTML5 para lograr una validación W3C.

## Estructura de una página HTML5

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

- Solamente el contenido dentro de la sección body (el área blanca arriba) se muestra en un navegador. El resto no es mostrado.

```
<!DOCTYPE html>
<html lang="es">

  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="keywords" content="html5, curso">
    <meta name="author" content="John Doe">
    <meta name="description" content="Curso MEAN">
    <title>Page Title</title>
    <link rel="stylesheet" href="styles.css" />
    <script src="main.js"></script>
  </head>

  <body>
  </body>
</html>
```

- Declarar un idioma es importante para las aplicaciones de accesibilidad (lectores de pantalla) y los motores de búsqueda.
- La declaración DOCTYPE define que el documento es de tipo HTML5. Ayuda a los navegadores a mostrar las páginas web correctamente.
- El elemento html es el elemento raíz de una página HTML.

- ◆ El elemento head contiene metainformación sobre el documento.
- ◆ El elemento meta contiene habitualmente información útil para navegadores y rastreadores web (aunque algunos expertos de la industria desaconsejan su uso).
- ◆ El elemento title define un título en la pestaña del navegador, proporciona un título para la página cuando se agrega a favoritos y muestra un título para la página en los resultados del motor de búsqueda.
- ◆ El elemento link define una hoja de estilos CSS.
- ◆ El elemento script define un archivo de código JavaScript.
- ◆ El elemento body contiene el contenido de la página visible.
- ◆ Según el estándar HTML5, las etiquetas html, body y head pueden ser omitidas, pero no es recomendable hacerlo.

Ejercicio: crea los archivos main.js, styles.css e index.html y escribir el código HTML5 de la estructura básica de una página.

## Codificación de caracteres

- ◆ La codificación de caracteres es fundamental en todos los lenguajes. En HTML5 se establece a partir de la etiqueta meta y el atributo charset.

```
<meta charset="utf-8" />
```

- ◆ ASCII fue el primer estándar de codificación de caracteres (también llamado conjunto de caracteres). ASCII definió 128 caracteres alfanuméricos diferentes que podrían usarse en Internet: números (0-9), letras en inglés (A-Z) y algunos caracteres especiales como ! \$ + \* () @ <>.
- ◆ ANSI (Windows-1252) era el juego de caracteres original de Windows, con soporte para 256 códigos de caracteres diferentes:
  - Los caracteres 0-127 son los mismos que ASCII.
  - Los caracteres 128 a 159 eran propietarios de Microsoft.
  - Los caracteres 160 a 255 son los mismos que en UTF-8.
- ◆ ISO-8859-1 fue el conjunto de caracteres predeterminado para HTML 4. Este juego de caracteres también admitía 256 códigos de caracteres diferentes.
  - Los caracteres 0-127 son los mismos que ASCII.
  - Los caracteres 128 a 159 no se utilizan
  - Los caracteres 160 a 255 son los mismos que en UTF-8.
- ◆ Como ANSI e ISO-8859-1 eran muy limitados, HTML 4 también era compatible con UTF-8:
  - Los caracteres 0-127 son los mismos que ASCII.
  - Los caracteres 128 a 159 no se utilizan

- Los caracteres 160 a 255 son los mismos que ANSI e ISO-8859-1.
- A partir de 256, UTF-8 aporta más de 10000 caracteres adicionales.
- UTF-8 (Unicode) cubre casi todos los caracteres y símbolos del mundo (incluyendo caracteres de lenguas y otros que pueden sustituir a iconos).
- La codificación de caracteres predeterminada para HTML5 es UTF-8.

## Caracteres especiales

- En ciertas ocasiones será necesario sustituir los caracteres reservados en HTML por entidades de caracteres.

Espacio en blanco: &nbsp;

Menor que: &lt;

Mayor que: &gt;

Ampersand: &amp;

Comillas dobles: &quot;

Comillas simples: &apos;

Ejemplo de etiqueta: &lt;body&gt;

Entidad	Descripción
	Espacio en blanco
<	Menor que
>	Mayor que
"	Comillas dobles
'	Comillas simples
<body>	Ejemplo de etiqueta

- Más símbolos

## Etiquetas básicas

- Cabeceras (h1-h6):

```
<!-- El tamaño de cada cabecera puede modificarse mediante estilos si es necesario-->
<h1>Cabecera 1</h1>
<h2>Cabecera 2</h2>
<h3>Cabecera 3</h3>
<h4>Cabecera 4</h4>
<h5>Cabecera 5</h5>
<h6>Cabecera 6</h6>
```

- ◆ Párrafos:

```
<p>Esto es un párrafo</p>
<p>Esto es otro párrafo</p>
```

- ◆ Salto de línea:

```
Hola
</br>
Adiós
```

- ◆ Separador:

```
Hola
<hr></hr>
Adiós
```

- ◆ Botones:

```
<button>Pulsa</button>
```

- ◆ Texto preformatado:

```
<pre>
body {
  color: red;
}
a {
  color:green;
}
</pre>
```

- ◆ Código con texto preformatado:

```
<pre>
<code>
x = 5;
y = 6;
z = x + y;
```

```
</code>  
</pre>
```

- Otras etiquetas relacionadas con código: kbd (para visualizar texto que es entrada de teclado), samp (para visualizar texto que es salida por pantalla) y var (para visualizar variables matemáticas). Son etiquetas que añaden diferentes estilos CSS al texto.

Ejercicio: prueba cada una de las etiquetas previamente comentadas.

## Formatear texto

```
<b> Texto resaltado</b><br>  
<strong>Texto resaltado con importancia semántica</strong><br>  
<i>Texto itálica</i><br>  
<em>Texto itálica con importancia semántica</em><br>  
<mark>Texto resaltado con color</mark><br>  
<small>Texto pequeño</small><br>  
<del>Texto tachado</del><br>  
<ins>Texto subrayado</ins><br>  
<sub>Texto subíndice</sub><br>  
<sup>Texto superíndice</sup><br>  
<q>Cita</q><br>  
<blockquote>Bloque con cita</blockquote><br>  
<abbr>Abreviatura</abbr> <!-- abbr sustituye a acronym en HTML5 -->  
<address>Dirección de contacto</address><br>  
<cite>Juego de Tronos</cite><br>  
<bdo dir="rtl">Texto bidireccional</bdo><br> <!-- ltr es el valor por defecto>
```

Ejercicio: probar cada una de las etiquetas previamente comentadas.

- La importancia semántica se refiere a que el texto debería ser considerado por rastreadores web para temas de posicionamiento.
- Las citas se muestran con comillas dobles.
- Los bloques con cita suelen aparecer identado.
- Las direcciones de contacto se muestran habitualmente en itálica y también son etiquetas semánticas.
- Los títulos de trabajo (libro, película, serie, canción, ...) aparecen en cursiva.

## Iframes

- Un iframe se usa para mostrar una página web dentro de otra página web.

```
<iframe src="ejercicios/1-7/index.html" height="200" width="300">Hello world!
</iframe>
```

- Ejemplo de iframe.
- La etiqueta iframe sustituye a la etiqueta frame, ya obsoleta de HTML4, pero su comportamiento es similar.
- Otras etiquetas relacionadas como noframes y frameset también han quedado obsoletas.

## Enlaces

- Para insertar enlaces se utiliza la etiqueta a.

- El destino del enlace se especifica con el atributo href.

- Hay cuatro tipos de enlace

1. Enlaces externos
2. Enlaces a rutas absolutas
3. Enlaces a rutas relativas
4. Enlaces a hash o marcadores

```
<a href="https://www.corenetworks.es/">Core Networks</a> <!-- Enlace externo -->
<a href="/">Index </a> <!-- Enlace a ruta absoluta -->
<a href="/contacto">Contacto </a> <!-- Enlace a ruta absoluta -->
<a href="contacto">Contacto </a> <!-- Enlace a ruta relativa -->
```

- Con los enlaces también se pueden crear marcadores o hash con identificadores (atributo id) para saltar directamente a determinadas partes de una página.

```
<!-- index.html -->
<h2 id="C4">Chapter 4</h2>
```

- Y ahora es posible acceder directamente a ese elemento mediante un hash:

```
<!-- index.html -->
<a href="#C4">Saltar al capítulo 4</a>
```

```
<!-- otrapagina.html -->
<a href="index.html#C4">Saltar al capítulo 4 de la página index.html</a>
```

- Un link no necesariamente tiene que contener un texto. También puede contener una imagen u otro elemento.

```
<a href="/">
  
</a>
```

- Existe otro atributo target que define:
  - \_blank: abre el enlace en una nueva ventana o pestaña.
  - \_self: abre el documento vinculado en la misma ventana, pestaña o frame en la que se hizo click (es el valor predeterminado).
  - \_top: abre el documento vinculado en todo el cuerpo de la ventana.
  - \_parent: abre la página en el frame padre.
  - -framename-: abre el documento vinculado en un marco con nombre framename.

Ejercicio: prueba cada uno de los valores del atributo target en esta [página](#) y el uso del hash.

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend (HTML)

Nuevas etiquetas



# Nuevas etiquetas

- ◆ Resumen de los nuevos elementos
- ◆ Elementos eliminados
- ◆ Nuevos elementos semánticos (I)
- ◆ Nuevos elementos semánticos (II)
- ◆ Nuevos elementos gráficos
- ◆ Nuevos elementos multimedia
  - Elemento video
  - Elemento audio
  - Elementos object y embed
- ◆ Otros nuevos elementos

## Resumen de los nuevos elementos

- ◆ HTML5 aporta nuevos elementos y nuevas APIs (Application Programming Interfaces)
- ◆ Respecto a los nuevos elementos:
  - Nuevos elementos semánticos como header, footer, nav, article, section y aside.
  - Nuevos atributos de elementos de formulario.
  - Nuevos elementos gráficos como svg y canvas.
  - Nuevos elementos multimedia como audio y video.
  - ...
- ◆ Respecto a las APIs:
  - HTML Geolocation.
  - HTML Drag and Drop.
  - HTML Local Storage.
  - HTML Application Cache.
  - HTML Web Workers.
  - HTML SSE.
  - ...
- ◆ También se permite la posibilidad de añadir nuevas etiquetas, aunque no es muy utilizado.

```
<script>
document.createElement("myHero")
</script>
<myHero>My Hero Element</myHero>
```

- La compatibilidad de los nuevos elementos de HTML5 con los diferentes navegadores puede comprobarse en la página [Can I use](#).

## Elementos eliminados

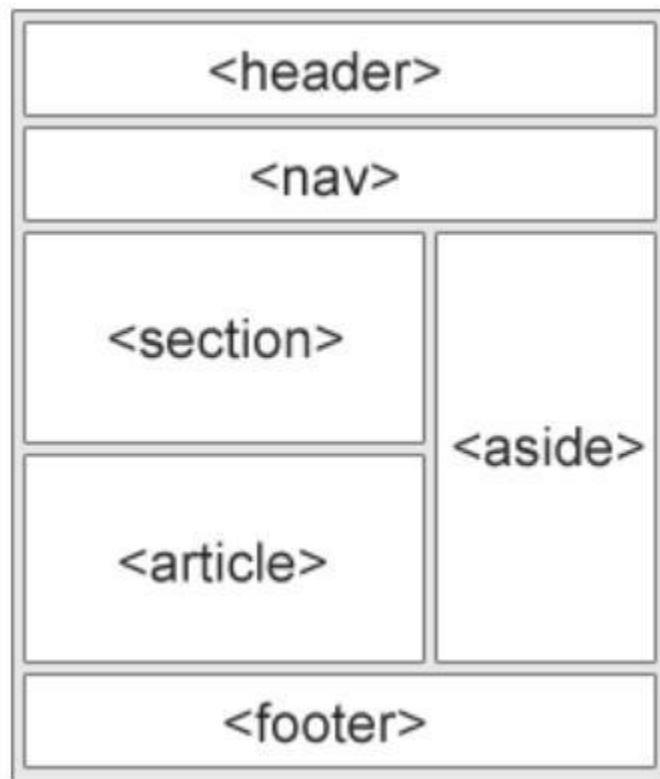
- Entre las etiquetas eliminadas se encuentran: acronym (sustituido por abbr), apple (sustituido por object), basefont (sustituido por CSS), big (sustituido por CSS), center (sustituido por CSS), dir (sustituido por ul), font (sustituido por CSS), frame, frameset, noframes, strike (sustituido por CSS), tt (sustituido por CSS).

## Nuevos elementos semánticos (I)

- Los elementos semánticos proporcionan un significado al contenido que tienen. Por ejemplo, en HTML4 existían algunas etiquetas "semánticas" como form o table y otras no semánticas como div o span.
- HTML5 define principalmente ocho nuevos elementos semánticos. Todos estos son elementos de de bloque.
- Para navegadores antiguos que no reconocen estas etiquetas pueden utilizarse los siguientes estilos.

```
header, section, footer, aside, nav, main, article, figure {  
    display: block;  
}
```

- Etiquetas semánticas más importantes:



- ♦ Header:
  - Especifica un encabezado para un documento o sección.
  - Debe usarse como un contenedor para el contenido introductorio.
  - Pueden utilizarse varios a lo largo de un documento.

```
<article>
  <header>
    <h1>What Does WWF Do?</h1>
    <p>WWF's mission:</p>
  </header>
  <p>WWF's mission is to stop the degradation of our planet's natural environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

- ♦ Etiqueta nav:
  - Define un conjunto de enlaces de navegación.
  - Pueden utilizarse varios a lo largo de un documento.

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

- Etiqueta aside:
  - Define algún contenido adicional (como una barra lateral).
  - Pueden utilizarse varios a lo largo de un documento.

```
<p>My family and I visited The Epcot center this summer.</p>

<aside>
  <h4>Epcot Center</h4>
  <p>The Epcot Center is a theme park in Disney World, Florida.</p>
</aside>
```

- Etiqueta footer:
  - Un pie de página generalmente contiene el autor del documento, información de copyright, enlaces a términos de uso, información de contacto, etc.
  - Únicamente debería existir un único footer en todo el documento.

```
<footer>
  <p>Posted by: Hege Refsnes</p>
  <p>Contact information: <a href="mailto:someone@example.com">
  someone@example.com</a>.</p>
  <address>JFK, 23</address>
</footer>
```

- Etiqueta section:
  - Define una sección de datos en el documento.
  - Una sección es una agrupación temática de contenido, generalmente con un título (con cabecera).

```
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is....</p>
</section>
```

- Etiqueta article:
  - Especifica contenido independiente y autónomo.
  - Un artículo debería tener sentido por sí mismo, y debería ser posible leerlo independientemente del resto del sitio web.
  - Ejemplo: post de un foro, noticia, post de un blog, etc.
  - Es posible anidar etiquetas section dentro de etiquetas article o viceversa, o etiquetas section dentro de etiquetas section y de la misma forma para etiquetas article. Todo depende del contenido.

```
<article>
  <h1>What Does WWF Do?</h1>
  <p>WWF's mission is to stop the degradation of our planet's natural environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

Ejercicio: crea una página HTML5 simple con contenido utilizando las etiquetas básicas header, nav, aside, section, article y footer.

## Nuevos elementos semánticos (II)

- Etiqueta details:
  - Especifica detalles adicionales que el usuario puede mostrar u ocultar a petición.

```
<details>
  <p> - by Refsnes Data. All Rights Reserved.</p>
  <p>All content and graphics on this web site are the property of the company Refsnes
Data.</p>
</details>
```

- Etiqueta summary:
  - Define un encabezado visible para el elemento details. Se puede hacer clic en el encabezado para mostrar u ocultar los detalles.

```
<details>
  <summary>Copyright 1999-2014.</summary>
  <p> - by Refsnes Data. All Rights Reserved.</p>
  <p>All content and graphics on this web site are the property of the company Refsnes
Data.</p>
</details>
```

- Etiqueta figure:
  - Especifica contenido autónomo como ilustraciones, diagramas, fotos, listas de códigos, etc.

```
<figure>
  
</figure>
```

- Etiqueta figcaption:
  - Define un título para un elemento figure.
  - Se puede colocar como el primer hijo o el último hijo del elemento figure.

- ◊ Es la forma recomendable de utilizar imágenes en HTML5, pero no es habitualmente usada.

```
<figure>
  
  <figcaption>Fig1. - A view of the pulpit rock in Norway.</figcaption>
</figure>
```

- ♦ Etiqueta main:
  - ◊ Especifica el contenido principal de un documento.
  - ◊ El contenido dentro del elemento main debe ser exclusivo del documento y no debe ser un descendiente de un elemento article, aside, footer, header o nav.

```
<main>
  <h1>Web Browsers</h1>
  <p>Google Chrome, Firefox, and Internet Explorer are the most used browsers today.</p>

  <article>
    <h1>Google Chrome</h1>
    <p>Google Chrome is a free, open-source web browser developed by Google, released in 2008.</p>
  </article>

  <article>
    <h1>Internet Explorer</h1>
    <p>Internet Explorer is a free web browser from Microsoft, released in 1995.</p>
  </article>

  <article>
    <h1>Mozilla Firefox</h1>
    <p>Firefox is a free, open-source web browser from Mozilla, released in 2004.</p>
  </article>
</main>
```

- ♦ Etiqueta time:
  - ◊ Define una fecha u hora legible por humanos.
  - ◊ El atributo datetime permite representar fechas/horas legibles por las máquinas, de modo que los motores de búsqueda pueden producir resultados de búsqueda más inteligentes, por ejemplo, con recordatorios para calendarios.

```
<p>Abrimos a las <time>10:00</time> todos los días</p>
<p>Hoy es <time datetime="2008-02-14 20:00">el día de San Valentín</time>.</p>
```

Ejercicio: crea una imagen con las etiquetas figure, img y figcaption.

Ejercicio proyecto final: asocia los diferentes elementos de las páginas del storyboard diseñadas con cada elemento semántico de HTML5.

## Nuevos elementos gráficos

- Canvas y SVG hacen uso de JavaScript y se verá posteriormente.

## Nuevos elementos multimedia

- Los archivos multimedia tienen formatos y diferentes extensiones como: .aac, .swf, .wav, .mp3, .mp4, .mpg, .wmv y .avi.
- Únicamente MP4 (desarrollado por Moving Pictures Expert Group), WebM (desarrollado por Mozilla, Opera, Adobe, y Google) y Ogg son los formatos compatibles de vídeo nativamente con el estándar HTML5. En general, MP4 es el formato recomendado.
- Únicamente AAC, MP3, WAV y Ogg son los formatos compatibles de audio nativamente con el estándar HTML5. En general, MP3 es el formato recomendado.

### Elemento video

- Antes de HTML5, un video solo podía reproducirse en un navegador con un complemento de un tercero (como flash player).
- Elemento vídeo:
  - El atributo controls agrega controles de video, como reproducir, pausar y volumen.
  - Desde el elemento source se pueden definir distintas fuentes de videos. El navegador seleccionará la compatible.
  - Para comenzar automáticamente el video, puede utilizarse el atributo autoplay (parece funcionar en Firefox, pero no en Chrome).
  - Para reproducir el video en loop, se puede utilizar el atributo loop.
  - Es una buena idea incluir siempre atributos de ancho y alto. Si el alto y el ancho no están configurados, la página puede parpadear mientras se carga el video.
  - Pueden también añadirse elementos de texto, como los subtítulos, mediante la etiqueta track.
  - Desde I can use puede accederse a la actual compatibilidad entre navegadores de los distintos contenedores de video (mp4, ogg, webm).

```
<video id="video" width="320" height="240" controls="controls" autoplay="autoplay" loop="loop">
  <source src="img/movie.mp4" type="video/mp4">
  <source src="img/movie.ogg" type="video/ogg">
  <source src="img/movie.webm" type="video/webm">
  <track src="subtitles_en.vtt" kind="subtitles" srclang="es" label="Spanish">
```

Tu navegador no es compatible con la etiqueta vídeo.  
</video>

- HTML5 define métodos DOM y propiedades que permiten cargar, reproducir y pausar videos, así como establecer la duración y el volumen. También hay eventos DOM que avisan cuando un vídeo comienza a reproducirse, está en pausa, etc.

```
let x = document.getElementById("video");
x.play();
x.pause();
```

Ejercicio: probar el uso de la etiqueta con un vídeo en diferentes navegadores y formatos.

## Elemento audio

- Antes de HTML5, un audio solo podía reproducirse en un navegador con un complemento (como flash).
- Desde el elemento source se pueden definir distintas fuentes de audio. El navegador seleccionará la compatible.
- Desde I can use puede accederse a la actual compatibilidad entre navegadores de los distintos contenedores de audio (ogg, mp3, wav, aac).

```
<audio controls="controls">
  <source src="img/horse.ogg" type="audio/ogg">
  <source src="img/horse.mp3" type="audio/mpeg">
  <source src="img/horse.wav" type="audio/wav">
</audio>
```

Tu navegador no es compatible con la etiqueta audio.

- Etiquetas como track, atributos como autoplay, controls o loop, y los métodos DOM comentados anteriormente para la etiqueta video, también son permitidos para la etiqueta audio.

Ejercicio: probar el uso de la etiqueta audio en diferentes navegadores y formatos.

## Elementos object y embed

- El propósito de un complemento es ampliar la funcionalidad de un navegador web.
- Algunos ejemplos de complementos conocidos son los applets de Java o los archivos swf de Flash.
- Los complementos se pueden agregar a páginas web con la etiqueta object o la etiqueta embed.

```
<object width="400" height="50" data="img/bookmark.swf"></object> <!--  
Soportado por HTML4 y HTML5 -->  
<embed width="400" height="50" src="img/bbookmark.swf"></embed> <!--  
Únicamente soportado por HTML5 -->
```

- Aunque también permiten incluir otro código HTML o imágenes:

```
<object width="100%" height="500px" data="snippet.html"></object>  
<embed width="100%" height="500px" src="snippet.html">  
  
<object data="audi.jpeg"></object>  
<embed src="audi.jpeg">
```

- En ocasiones, elementos externos también son introducidos mediante la etiqueta iframe.
  - Youtube

```
<iframe width="420" height="315"  
src="https://www.youtube.com/embed/tgbNymZ7vqY"></iframe>
```

- Google maps
- Twitter
- ...

Ejercicio: crea un archivo HTML con un mapa de [Google Maps](#).

## Otros nuevos elementos

- Etiqueta meter:
  - Representa datos dentro de un rango determinado.
  - Aunque es parecido a una barra de progreso, no debería ser utilizada para este fin. Para barras de progresos ya existe una etiqueta llamada progress.

```
<meter value="2" min="0" max="10">2 out of 10</meter><br>  
<meter value="0.6">60%</meter>
```

- Etiqueta progress:
  - Representa una barra de progreso.
  - No es adecuado para representar un indicador (por ejemplo, uso de espacio en disco o relevancia de un resultado de consulta). Para representar un indicador, es mejor usar la etiqueta meter.

```
<progress value="22" max="100"></progress>
```

- ◆ Etiqueta math (no compatible con Chrome):
  - msup: este elemento se utiliza para adjuntar un superíndice a una expresión. Utiliza la siguiente expresión donde:
    - mi: base.
    - mn: superíndice.
  - mo: representa un operador.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
<mrow>
<msup><mi>a</mi><mn>2</mn></msup>
<mo>+</mo>

<msup><mi>b</mi><mn>2</mn></msup>
<mo>=</mo>

<msup><mi>c</mi><mn>2</mn></msup>
</mrow>
</math>
```

Ejercicio proyecto final: crea la estructura base de la página principal con etiquetas semánticas, incluyendo main, section y article.

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend (CSS)

CSS básico (I)



# CSS básico (I)

- ◆ [Introducción](#)
- ◆ [Formas de insertar código CSS](#)
- ◆ [Selectores](#)
  - [El selector etiqueta](#)
  - [El selector id](#)
  - [El selector class](#)
- ◆ [Colores](#)
- ◆ [Especificidad y estilos en cascada](#)
- ◆ [Herencia](#)
- ◆ [Uso de múltiples selectores](#)
- ◆ [Combinaciones de selectores](#)
  - [Tipos de combinaciones de selectores](#)
- ◆ [Anotación !important](#)

## Introducción

- ◆ CSS (Hojas de Estilo en Cascada) es un lenguaje que describe el estilo y diseño de un documento HTML.
- ◆ El desarrollo y evolución de la tecnología CSS está a cargo de la organización W3C (World Wide Web Consortium).
- ◆ CSS ha contado con diversas versiones hasta llegar a la actualidad:
  - CSS1: lanzada en 1996.
  - CSS2: lanzada en 1998.
  - CSS3 (actual versión): lanzada en 2012. No está previsto que aparezca CSS4, sino que CSS3 se mantendrá con revisiones constantes y módulos que incorporarán características nuevas. Por esa razón, a CSS3 se le denomina también simplemente como CSS.
- ◆ CSS describe cómo deben mostrarse los elementos HTML, incluyendo layouts (columnas y filas), colores, fuentes, variaciones entre dispositivos, etc.
- ◆ La sintaxis de CSS consiste en un selector y un bloque de declaración.

Selector

Declaration

Declaration

h1

{ color:blue; font-size:12px; }

Property

Value

Property

Value

```
p {  
    color: red;  
    text-align: center;  
}
```

- ◆ El selector apunta a un elemento HTML, generalmente apuntando a:
  - El nombre de una etiqueta HTML.
  - El valor de un atributo id una etiqueta HTML.
  - El valor de un atributo clase en una etiqueta HTML.
  - La existencia de un determinado atributo en una etiqueta HTML.
- ◆ El bloque de declaraciones está delimitado por llaves y contiene una o más declaraciones separadas por punto y coma.
- ◆ Cada declaración incluye una propiedad CSS con un nombre y un valor, separado por dos puntos.

## Formas de insertar código CSS

- ◆ El código CSS puede añadirse de tres formas distintas:
  - En línea: utilizando el atributo style en elementos HTML.
  - Interno: utilizando un elemento style en la sección head.
  - Externo: utilizando un archivo CSS externo mediante la etiqueta link.
- ◆ En línea:

```
<h1 style="color:blue;">Cabecera azul</h1>
```

- ◆ Interno:

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <style>  
  
        body {  
            background-color: powderblue;  
        }  
  
        h1 {  
            color: blue;  
        }
```

```
}
```

```
p {
```

```
    color: red;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
    <h1>Cabecera</h1>
```

```
    <p>Párrafo</p>
```

```
</body>
```

```
</html>
```

- ♦ Externo:

```
<!-- index.html -->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <link rel="stylesheet" href="styles.css">
```

```
</head>
```

```
<body>
```

```
    <h1>Esto es una cabecera</h1>
```

```
    <p>Esto es un párrafo</p>
```

```
</body>
```

```
</html>
```

```
/* styles.css */
```

```
body {
```

```
    background-color: powderblue;
```

```
}
```

```
h1 {
```

```
    color: blue;
```

```
}
```

```
p {
```

```
color: red;  
}
```

- ◆ Utilizar un archivo externo es la forma recomendada por las siguientes razones:
  - Separa el código CSS del HTML.
  - ◊ Es más simple de leer y entender.
  - ◊ Es más fácil de depurar e identificar errores.
  - Es más fácil de modificar, mantener y reutilizar.
  - Se evita sobrecargar los archivos HTML.
  - ◊ Los navegadores pueden cachear los archivos CSS si son utilizados por distintas páginas HTML, aumentando la velocidad de renderizado.
- ◆ También pueden utilizarse referencias a archivos CSS externos en otros dominios.

```
<link rel="stylesheet" href="https://www.w3schools.com/html/styles.css">
```

- ◆ Si se han definido varias veces el mismo selector en una hoja de estilo (archivo CSS externo), se utilizará el último definido.
- ◆ ¿Qué estilo se usará cuando existan varias hojas de estilo definidas para un mismo elemento HTML dentro de un documento HTML? Se considerará de mayor a menor prioridad:
  - ◊ Inline (dentro de un elemento HTML).
  - Hojas de estilo externas e internas (se aplicará la última que haya sido definida).
  - Estilo predeterminado del navegador.

## Selectores

### El selector etiqueta

- ◆ Basado en el nombre de una etiqueta HTML.

```
<style>  
  
/* Se verán afectados por este estilo todas las etiquetas <p> */  
p {  
    text-align: center;  
    color: red;  
}  
  
</style>  
  
<span>Hola</span>  
<p>Adiós</p>
```

## El selector id

- Utiliza el atributo id de un elemento HTML para seleccionar un elemento HTML específico.
- El valor del id debería ser único en la página para un elemento HTML. Si embargo, si existen varios elementos HTML con el mismo valor de id, entonces se aplicarán los estilos a todos los elementos.
- En CSS se utiliza el carácter # seguido del nombre del id para referenciar al elemento HTML con ese id como atributo.

```
<style>

#saludo {
    text-align: center;
    color: red;
}

</style>

<p id="saludo">Hola</p>
<p>Adiós</p>
```

## El selector class

- Utiliza el atributo class de un elemento HTML para seleccionar múltiples elementos HTML.
- Pueden existir varios elementos HTML con el mismo valor para el atributo class dentro de la página.
- En CSS se utiliza el carácter . seguido del nombre de la clase para referenciar al elemento HTML con esa clase.

```
<style>

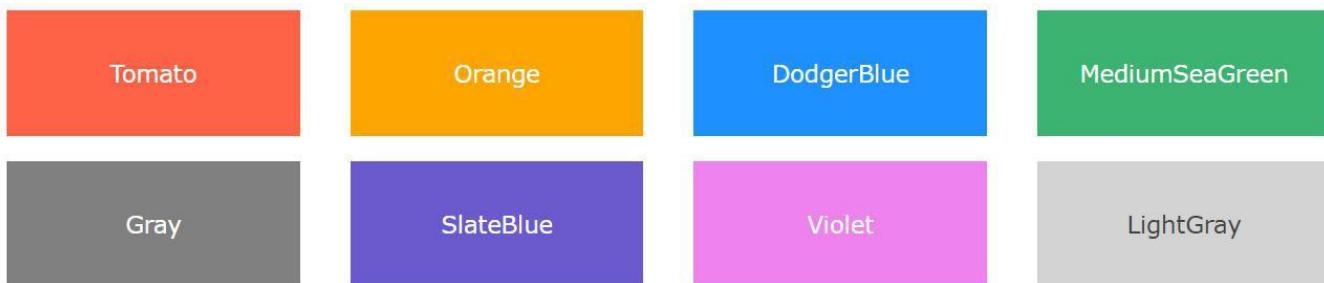
.saludos {
    text-align: center;
    color: red;
}

</style>

<p class="saludos">Hola</p>
<p class="saludos">Buenos días</p>
```

## Colores

- Los colores se especifican usando nombres de colores predefinidos o valores RGB, HEX o HSL.
- Nombres de colores:



- Posibles formas de representar los colores.

```
<!-- nombres de colores -->
<h1 style="background-color:red">Rojo</h1>

<!-- rgb(red, green, blue) entre 0 y 255 -->
<h1 style="background-color:rgb(255, 99, 71);">255, 99, 71</h1>

<!-- hexadecimal (#rrggbb) con rr (red), gg (green), bb(blue) entre 00 y FF -->
<h1 style="background-color:#ff6347">#ff6347</h1>

<!--
    hsl(hue, saturation, lightness, alpha) con tono, saturación, claridad, opacidad
    El tono varía de 0 a 360. 0 es rojo, 120 es verde, 240 es azul y 360 vuelve a ser el
    rojo
    La saturación (intensidad del color) es un valor porcentual, 0% significa un tono de
    gris y 100% es el color completo
    La claridad es también un porcentaje: 0% es oscuro y el 100% es claro
    La opacidad es un número: 0.0 (totalmente transparente) y 1.0 (totalmente opaco)
-->
<h1 style="background-color:hsla(120, 100%, 64%, 0.5);">120, 100%, 64%,
0.5</h1>
```

## Especificidad y estilos en cascada

- Un elemento HTML puede verse afectado por múltiples selectores CSS.

```
<style>

#green {
  color: green;
}
```

```
div {  
    color: red;  
}  
  
</style>  
  
<!-- este div es coincidente con el selector de etiqueta div -->  
<div>Hola</div>  
  
<!-- este div es coincidente con el selector de etiqueta div y el selector #green -->  
<div id="green">Hola</div>
```

- Si el selector es exactamente el mismo, entonces:
  - De las propiedades no coincidentes se aplican todas.
  - De las propiedades coincidentes únicamente se aplican las del selector colocado más adelante.

```
<style>  
  
div {  
    font-weight: bold;  
    color: green;  
}  
  
div {  
    color: red;  
}  
  
</style>  
  
<!-- se aplica la propiedad font-weight del primer selector div -->  
<!-- NO se aplica la propiedad color del primer selector div -->  
<!-- se aplica la propiedad color del segundo selector div -->  
<div>Hola</div>
```

- Los selectores de clase tienen mayor especificidad que los selectores etiqueta.

```
<style>  
  
.red-and-bold {  
    font-weight: bold;  
    color: green;  
}  
  
div {  
    color: red;  
}
```

```
}
```

```
</style>
```

```
<!-- se aplican las propiedades font-weight y color del selector .red-and-bold -->
```

```
<!-- NO se aplica la propiedad color del segundo selector div -->
```

```
<div class="red-and-bold">Hola</div>
```

- Y los selectores de id tienen mayor especificidad que los selectores de clase.

```
<style>
```

```
#black-and-lighter {  
    font-weight: lighter;  
    color: black;  
}
```

```
.red-and-bold {  
    font-weight: bold;  
    color: green;  
}
```

```
div {  
    color: red;  
}
```

```
</style>
```

```
<!-- se aplican las propiedades font-weight y color del selector #white-and-bold -->
```

```
<!-- NO se aplican las propiedades font-weight y color del selector .red-and-bold -->
```

```
<!-- NO se aplica la propiedad y color del selector div -->
```

```
<div id="black-and-lighter" class="red-and-bold">Hola</div>
```

- La especificidad entre selectores se observa bien con las herramientas de desarrollador de Google Chrome donde:
  - element.style se refiere al estilo que se aplica inline (dentro de la etiqueta) y es el que tiene mayor especificidad (en el ejemplo anterior está vacío).
  - div { display: block } se refiere al estilo que automáticamente añade el navegador y es el que tiene menor especificidad.

```

element.style {
}

#white-and-bold { a.html:4
  font-weight: 100;
  color: white;
}

.red-and-bold { a.html:10
  font-weight: bold;
  color: green;
}

div { a.html:16
  color: red;
}

div { user agent stylesheet
  display: block;
}

```

- En resumen, de menor a mayor, la especificidad de los selectores es la siguiente:
  - Selectores establecidos automáticamente por el navegador.
  - Selector universal (\*).
  - Pseudoselectores y selectores de etiqueta: si existe ambigüedad, la última propiedad dentro del archivo CSS es la que prevalece.
  - Clases, pseudoclases y selectores de atributo: si existe ambigüedad, la última propiedad dentro del archivo CSS es la que prevalece.
  - Selectores ID.
  - Estilos inline.
- Algunos desarrolladores prefieren utilizar exclusivamente selectores de clase y, excepcionalmente, el resto de selectores solamente para estilos muy concretos.
- El selector id suele utilizarse más para obtener el elemento mediante JavaScript a partir del método getElementById y también sirve para añadir marcadores y enlazar diferentes partes de una página mediante hipervínculos.

## Herencia

- La herencia se refiere a las propiedades CSS de un elemento que se heredan a los elementos que contiene.

```

<style>

#green-and-bold {
  font-weight: bold;
  color: green;
}

```

```
</style>

<div id="green-and-bold">
    <!-- se heredan las propiedades del selector #green-and-bold -->
    <div>Hola</div>
</div>
```

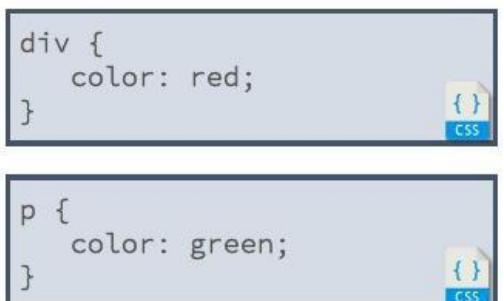
- Sin embargo, las propiedades que se heredan tienen una especificidad muy baja, más aún que los selectores establecidos automáticamente por el navegador. Cualquier selector sobre un elemento tiene más especificidad que uno que se hereda.

```
<style>

#green-and-lighter {
    font-weight: lighter;
    color: green;
}

</style>

<div id="green-and-lighter">
    <!-- solamente se aplica la propiedad color del selector #green-and-lighter -->
    <!-- la propiedad font-weight se toma del selector h1 (automáticamente impuesto por
el navegador) y con mayor especificidad que una propiedad heredada -->
    <h1>Hola</h1>
</div>
```



```
<div>
    <div>
        <h1>Inherited!</h1>
    </div>
    <p>Overwritten</p>
    <div>Inherited!</div>
    <article>
        <p>Overwritten</p>
    </article>
    <p>Overwritten</p>
</div>
```



- Puede迫使自己增加一个属性的特定性通过值 *inherit*

```
<style>
#green-and-lighter {
    font-weight: lighter;
    color: green;
}

#red-and-bold {
    font-weight: bold;
    color: inherit;
}

div {
    color: blue;
}

</style>
<div id="green-and-lighter">

<!-- orden de la especificidad: #red-and-bold &gt; div &gt; #green-and-lighter --&gt;
<!-- sin embargo, se aplica la propiedad color del selector #green-and-lighter porque el
selector más específico (#red-and-bold) indica que se herede esa propiedad --&gt;
&lt;div id="red-and-bold"&gt;
    Hola
&lt;/div&gt;
&lt;/div&gt;</pre>

```

- La herencia se utiliza generalmente para aplicar propiedades de tipo fuente a toda la página.

```
<head>
<style>

body {
    font-family: Georgia;
}

</style>
</head>

<body>
<div>
    <h1>Hola</h1>
</div>
</body>
```

- Si se heredan múltiples propiedades CSS iguales, entonces se considera la del ancestro más cercano.

## Uso de múltiples selectores

- Los selectores pueden combinarse para ser más específicos.
- Por ejemplo, un elemento puede estar referenciado por más de una clase, separado por espacios. En caso de coincidencia de propiedades se tiene en cuenta la última clase establecida el archivo CSS.

```
<style>

    .centrar {
        text-align: center;
    }

    .red {
        color: red;
    }

</style>

<!-- aplican los estilos de los selectores .centrar y .red -->
<p class="centrar red">Hola</p>

<!-- aplican los estilos del selector .centrar -->
<p class="centrar">Buenos días</p>
```

- También se pueden agrupar selectores para aplicar los mismos estilos utilizando una coma para separar los selectores.

```
<style>

/* afecta a todos los elementos con el atributo class="saludos" o class="red" */
.saludos, .red {
    text-align: center;
    color: red;
}

</style>

<p class="saludos red">Hola</p>
<p class="saludos">Buenos días</p>
<p class="red">¿Cómo estás?</p>
```

## Combinaciones de selectores

- Pueden utilizarse selectores de distinto tipo para aumentar la especificidad.

```
<style>
```

```
/* se aplica a un elemento p que esté contenido dentro de un elemento div y que no necesariamente tiene que ser ancestro directo */
```

```
div p {  
    background-color: yellow;  
}
```

```
p {  
    background-color: red;  
}
```

```
</style>
```

```
<div id="hola">
```

```
<!-- se aplica la propiedad background-color de la combinación de selectores div p, que es más específica que el selector p -->
```

```
<!-- sin embargo, la combinación de selectores div p no sería más específica que otros como #hola p o .saludos -->
```

```
<p class="saludos">Buenos días</p>
```

```
</div>
```

- Existen diferentes tipos de selectores de combinación

- Selector adyacente: +
- Selector adyacente general: ~
- Selector de descendencia: >
- Selector de descendencia general: sin símbolo

### Tipos de combinaciones de selectores

- Selector adyacente (+): aplica a los elementos inmediatamente adyacentes.

```
<style>
```

```
h2 + p {  
    color: red;  
}
```

```
</style>
```

```
<div>  
    <h2>NO aplica</h2>
```

```
<p>Aplica</p>
<h2>NO aplica</h2>
<h3>NO aplica</h3>
<p>NO aplica</p>
<h2>NO aplica</h2>
<p>Aplica</p>
</div>
```

- Selector adyacente general (~): aplica a todos los elementos adyacentes, independientemente de si se encuentran o no inmediatamente adyacentes.

```
<style>

h2 ~ p {
    color: red;
}

</style>

<div>
<h2>NO aplica</h2>
<p>Aplica</p>
<h2>NO aplica</h2>
<h3>NO aplica</h3>
<p>Aplica</p>
<h2>NO aplica</h2>
<p>Aplica</p>
</div>
```

- Selector de descendencia (>): aplica a los elementos inmediatamente descendientes.

```
<style>

div > p {
    color: red;
}

</style>

<div>
<div>NO aplica</div>
<p>Aplica</p>
<div>NO aplica</div>
<article>
    <p>NO aplica</p>
</article>
```

```
<p>Aplica</p>
</div>
```

- Selector de descendencia general ( ): aplica a todos los elementos descendientes, independientemente si son o no inmediatamente descendientes.

```
<style>
div p {
  color: red;
}

</style>

<div>
<div>NO aplica</div>
<p>Aplica</p>
<div>NO aplica</div>
<article>
  <p>Aplica</p>
</article>
<p>Aplica</p>
</div>
```

## Anotación !important

- La anotación !important permite que una propiedad sea la más específica posible (incluso más que las propiedades declaradas inline utilizando el atributo style).

```
<style>
.rojo {
  color: red !important;
}

</style>

<!-- se muestra rojo porque la anotación !important tiene más especificidad que el estilo inline -->
<p style="color:blue;" class="rojo">Hola</p>
```

- En la práctica no es buena idea utilizarla salvo para propósitos de prueba o depuración.

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend

# (JavaScript)

Fundamentos



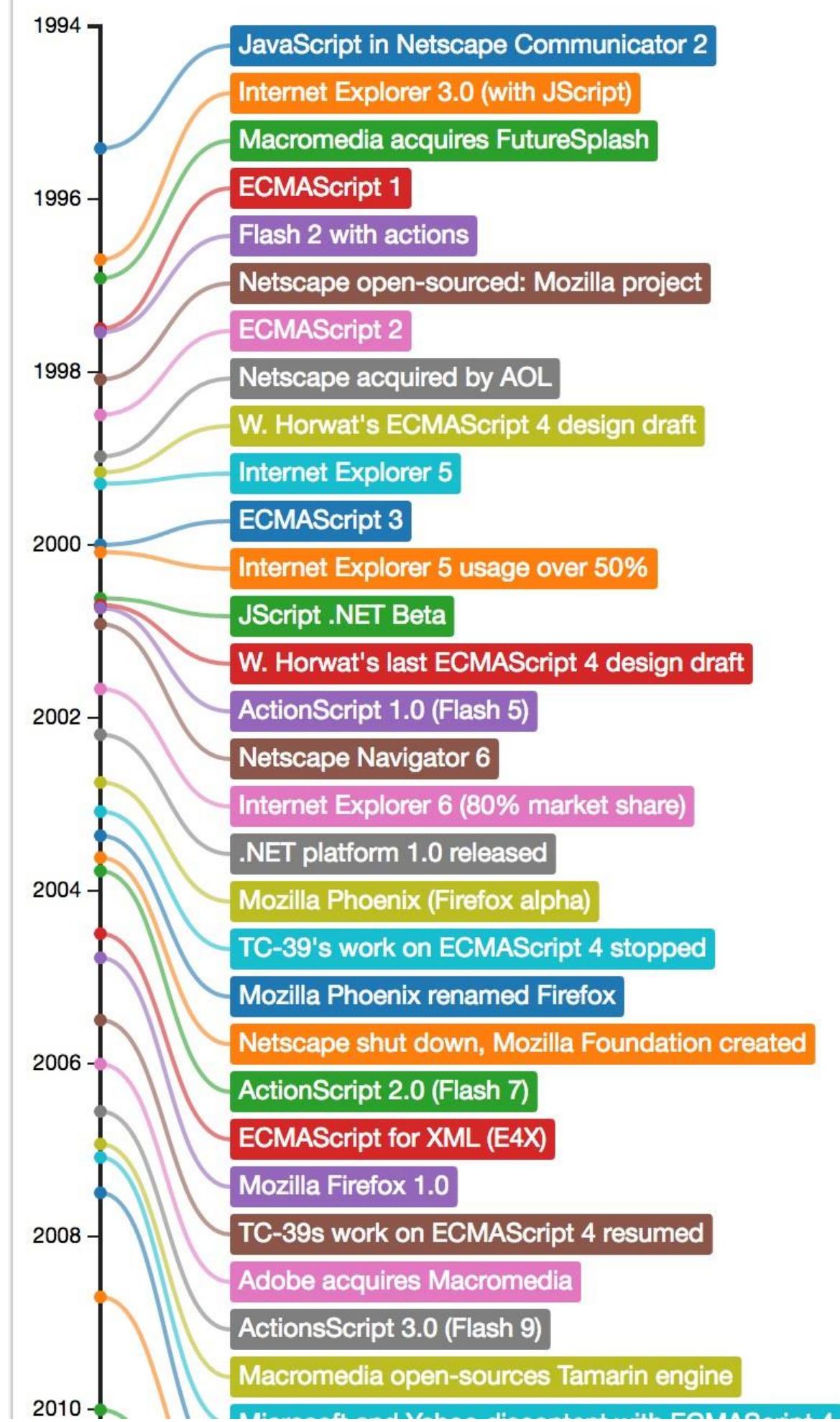
# Fundamentos

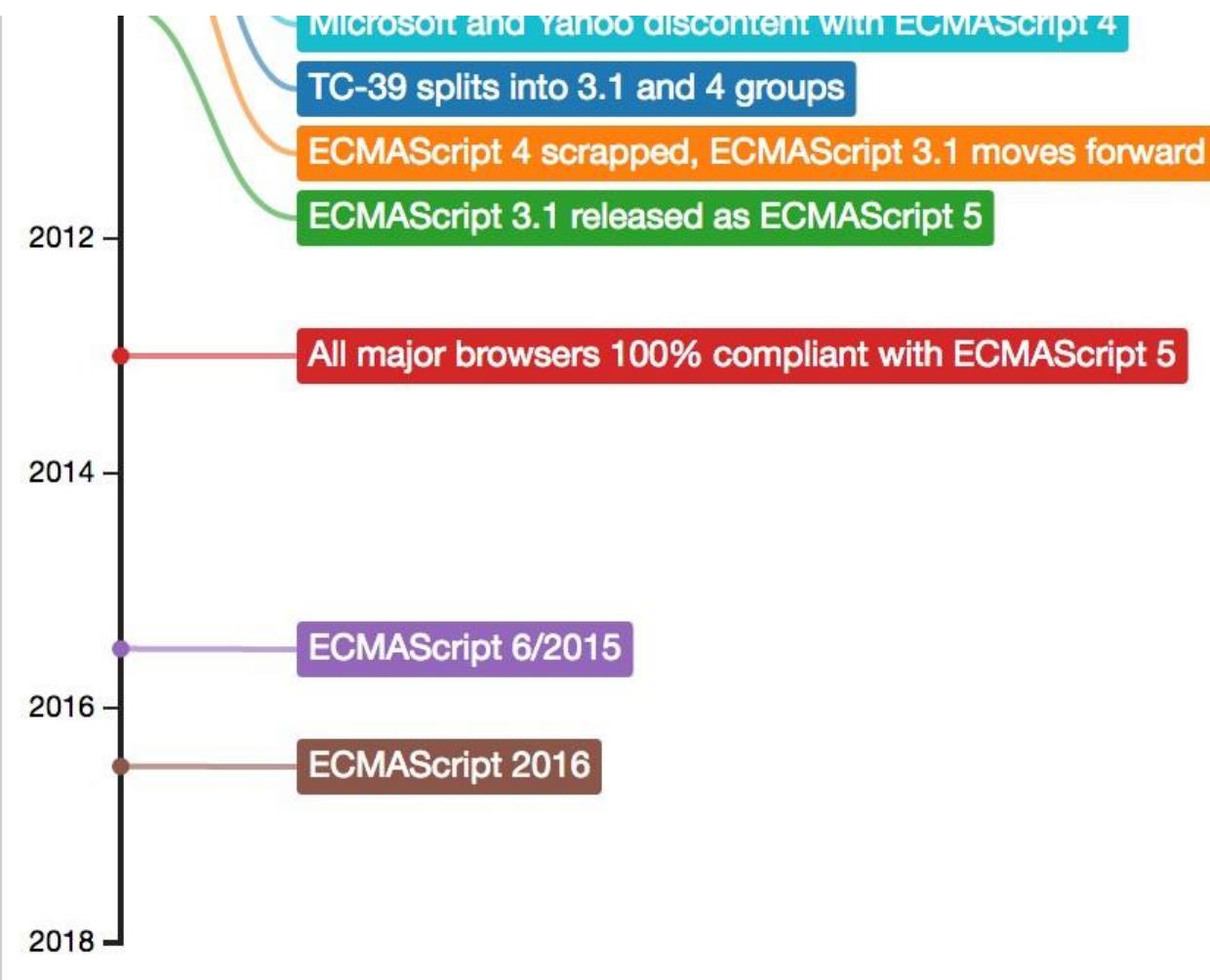
---

- ◆ Historia
- ◆ Introducción
- ◆ Ejecución de código JavaScript
- ◆ JavaScript en el lado del cliente (frontend)
- ◆ JavaScript en el lado del servidor (backend)
- ◆ Uso de paquetes de JavaScript en el lado del cliente (frontend)
- ◆ Uso de paquetes de JavaScript en el lado del servidor (backend)
- ◆ Conceptos básicos de JavaScript

## Historia

- ◆ JavaScript (abreviado comúnmente JS), e inicialmente llamado Livescript, fue inventado en 1995 por Brendan Eich.
- ◆ Originariamente fue concebido como un lenguaje de programación interpretado, no orientado a objetos, basado en prototipos y con tipado débil (no es necesario declarar el tipo de variable) y dinámico (una misma variable puede tomar valores de distinto tipo en diferentes momentos del código).
- ◆ ECMAScript es el estándar actual que define la sintaxis de JavaScript. Actualmente también está aceptado como estándar internacional ISO 16262.
- ◆ La primera versión de ECMAScript fue publicada en 1997 por la organización ECMA International.
- ◆





- Todos los navegadores modernos (desde Internet Explorer 11 hacia adelante) soportan ECMAScript 5 (2009), mientras que los antiguos son solamente compatibles con la versión 3.
- ECMA International publicó la sexta versión de ECMAScript en Junio del 2015, conocida como ECMAScript 2015, ECMAScript 6 o ES6.
- ECMAScript 6 supuso un cambio sustancial en JavaScript por la importante cantidad de funcionalidades que fueron incorporadas al lenguaje.
- Desde entonces, todos los años se han publicado nuevas versiones del estándar ECMAScript:
  - ECMAScript 2016 / ECMAScript7 / ES7.
  - ECMAScript 2017 / ECMAScript8 / ES8.
  - ECMAScript 2018 / ECMAScript9 / ES9.
  - ECMAScript 2019 / ECMAScript10 / ES10.
  - ECMAScript 2020 / ECMAScript11 / ES11.
  - ESNext (incluye las características de la próxima versión del estándar)
- El transpilador Babel permite convertir código de JavaScript de una versión a otra para garantizar compatibilidad con todos los navegadores.

## Introducción

- ◆ Entre otras funcionalidades, el estándar ECMAScript define:
  - Tipos de datos: boolean, number, string, function, object, etc.
  - Sintaxis: reglas de análisis, palabras clave, flujos de control, etc.
  - Mecanismos de control de excepciones: throw, try/catch y creación de excepciones personalizadas por el usuario.
  - Los objetos globales: el objeto global en un navegador (JavaScript en el lado del cliente) es el objeto window, pero ECMAScript sólo define aquellos métodos no específicos para navegadores. Ej: parseInt, parseFloat, decodeURI, encodeURI, etc.
    - En Node.js (JavaScript en el lado del servidor) el objeto global se denomina global.
  - Mecanismo de herencia basada en prototipos.
  - Algunos objetos y funciones: JSON, Math, métodos para extender un prototipo, etc.
  - Modo estricto.
- ◆ El objeto global del navegador y el Document Object Model (DOM) están estandarizados por el World Wide Web Consortium (W3C). En particular, el DOM define una serie de objetos y métodos para manejar elementos HTML desde el navegador (JavaScript en el lado del cliente).
- ◆ Otras APIs de JavaScript también definidas por el W3C son:
  - Las funciones setTimeout and setInterval.
  - Manejo de eventos e interacción con el usuario.
  - El objeto XMLHttpRequest, que permite manejar peticiones HTTP asíncronas (AJAX).
  - Webworkers para computación paralela.
  - Websockets para comunicación bidireccional.
  - Canvas 2D para dibujo.
  - WebGL para realizad virtual.
  - Web Storage API.
  - Otras Web API.
- ◆ En mayo del 2019, el W3C entrega el desarrollo de los estándares HTML y DOM a un consorcio de proveedores de navegadores y de empresas denominado WHATWG (Web Hypertext Application Technology Working Group).

## Ejecución de código JavaScript

- JavaScript puede ejecutarse:
  - En el lado del cliente (frontend).
  - En el lado del servidor (backend).

<b>JavaScript (frontend)</b>	<b>JavaScript (backend)</b>
Se ejecuta generalmente en el navegador web del usuario final (Chrome, Firefox, Edge, Safari, etc)	Se ejecuta en un entorno de ejecución independiente del navegador (generalmente con Node.js)
El código es visible por el usuario final	El código no es visible por el usuario final
El usuario final puede manipular el código que se ejecuta	El usuario final no puede manipular el código que se ejecuta
Maneja una interfaz gráfica web	Generalmente no maneja interfaces gráficas de ningún tipo
Se encuentra limitado por las restricciones del sandbox del navegador	No posee limitaciones de ningún tipo
Maneja principalmente eventos de usuario, renderizado de páginas HTML o conexiones a servicios web	Maneja conexiones a bases de datos, ficheros, validación de datos, procesado de peticiones, generación de respuestas y prácticamente cualquier tarea computacional posible
Capacidad computacional limitada por las prestaciones del dispositivo del usuario final	Capacidad computacional limitada por las prestaciones del servidor o del sistema de cloud computing en el que se ejecuta el programa
Generalmente no maneja tareas con un alto costo computacional	Puede manejar tareas con un alto costo computacional
El número de librerías o paquetes de JavaScript que puede utilizarse está reducido	La mayoría de librerías o paquetes de JavaScript pueden ser utilizadas

## JavaScript en el lado del cliente (frontend)

- El código de JavaScript ejecutado en el cliente es embebido en HTML mediante la etiqueta script.

```
<!-- index.html -->
```

```
<script>
  alert("Hello world!");
  document.write("Hello world!");
  console.log("Hello world!")
</script>
```

- Aunque lo habitual es utilizar un archivo JavaScript independiente y referenciarlo mediante el atributo src de la etiqueta script.

```
<!-- index.html -->

<script src="main1.js"></script>
<script src="js/main2.js"></script>
<script src="../js/main2.js"></script>
<script src="/js/main2.js"></script>
```

- La etiqueta noscript se usa para proporcionar un contenido alternativo para aquellos usuarios que están en disposición de navegadores con JavaScript desactivado o que no admiten scripts del lado del cliente.

```
<script>
  document.write("Hello world!");
</script>

<noscript>
  Lo siento, tu navegador no acepta JavaScript!
</noscript>
```

- Es recomendable que el código JavaScript se cargue al final de la página HTML.

```
<body>
<!-- ... -->

<script>

  function myFunction() {
    document.getElementById("demo").innerHTML = "Párrafo cambiado";
  }

</script>
</body>
```

## JavaScript en el lado del servidor (backend)

- ◆ Para ejecutar JavaScript en el lado del servidor es necesario instalar el software Node.js
- ◆ Node.js es un entorno multiplataforma, de código abierto, asíncrono y basado en ECMAScript.
- ◆ Características de Node.js:
  - Tecnología del lado del servidor (backend).
  - Gratuito y software libre.
  - Arquitectura orientada a eventos.
  - Basado en el motor V8 de Google.
  - Compatibilidad con cualquier sistema operativo.
  - Soporte para bases de datos y servicios REST.
  - Acceso al sistema de archivos del equipo, a los procesos en ejecución y a información del sistema operativo.
  - Rápido y fácil de configurar.
  - Miles de paquetes disponibles.
  - Tecnología fácil de aprender para aquellos que ya conocen Javascript.
- ◆ Existen dos versiones de Node.js para descargar: LTS (estable) y Actual (últimas características).
- ◆ Node.js no necesita de archivos HTML para poder ejecutar código JavaScript.

```
// main.js  
  
console.log('Hello world!');
```

- ◆ Para ejecutar un programa con Node.js se utiliza el comando node, seguido del nombre del archivo que contiene código JavaScript (con extensión js).

```
node main.js
```

## Uso de paquetes de JavaScript en el lado del cliente (frontend)

- ◆ El uso de paquetes o librerías de JavaScript en el lado del cliente es muy simple. Basta con referenciar la ubicación externa de la librería mediante el atributo src de la etiqueta script.
- ◆ La librería estará disponible justo después de haber sido importada.

```
<!-- importación de la librería jQuery -->  
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js">  
</script>
```

```
<!-- ahora puede utilizarse la librería jQuery -->
<script>
```

```
$(document).ready(function(){
    console.log('Librería jQuery cargada');
});
```

```
</script>
```

## Uso de paquetes de JavaScript en el lado del servidor (backend)

- La instalación de paquetes de JavaScript en el lado del servidor es gestionada mediante la herramienta npm (Node Package Manager).
- npm se instala junto a Node.js.

```
npm -version
```

- El repositorio de npm posee miles de paquetes para descargar. Otras librerías pueden encontrarse también en Github.
- Para crear un proyecto nuevo con Node.js también se utiliza npm.

```
mkdir nuevo-proyecto
```

```
cd nuevo-proyecto
```

```
# crea un nuevo proyecto de Node.js
npm init
```

- Al crear un nuevo proyecto se genera un archivo llamado package.json que contiene todo el listado de paquetes que requiere el proyecto.
- Formas de instalar un paquete con npm:
  - Con -g: instala el paquete de forma global (requiere permisos de administrador y no crea la carpeta node\_modules en el directorio actual puesto que el paquete es instalado en directorios del sistema).
  - Con --save: instala el paquete de forma local, dentro del directorio node\_modules y añadiendo la dependencia en el archivo package.json.
  - Con --save-dev: instala el paquete de forma local, dentro del directorio node\_modules, añadiendo la dependencia en el archivo package.json, pero el paquete únicamente será utilizado para propósitos de desarrollo.
  - Sin ningún parámetro: similar a --save

- Es importante que npm esté actualizado a su última versión.

```
# actualiza npm de forma global  
npm install -g npm
```

- Otras librerías también se instalan con el comando npm install

```
# posibles formas de instalar el paquete redis  
npm install -g redis  
npm install --save redis  
npm install redis  
npm i redis  
npm install redis --save-dev
```

- A continuación se exponen otros posibles usos del comando npm.

Acción	Ejemplo
Listar los paquetes de Node.js instalados de forma local	npm ls
Listar los paquetes de Node.js instalados de forma global	npm ls -g
Buscar un paquete	npm search redis
Actualizar un paquete de forma local	npm update redis
Actualizar un paquete de forma global	npm update -g redis
Desinstalar un paquete de forma local	npm uninstall redis
Desinstalar un paquete de forma global	npm uninstall -g redis

- Finalmente, para importar un paquete desde JavaScript en el lado del servidor se utiliza require.

```
const redis = require("redis");
```

## Conceptos básicos de JavaScript

- Para mostrar por consola en JavaScript se utiliza la instrucción `console.log`

```
<script>  
  
// hola  
console.log('hola');
```

```
// en un lugar de la mancha
console.log('en', 'un', 'lugar', 'de', 'la', 'Mancha');
console.info('Info');
console.warn('Warning');
console.error('Error');

</script>
```

- Para escribir contenido en el documento HTML de una página se invoca a la instrucción `document.write`.

```
<pre>
<script>

  document.write("Texto2");

</script>
</pre>
```

- Con la función `alert` es posible mostrar una advertencia o alerta por pantalla en el navegador.

```
<script>

  alert("Hello world!");

</script>
```

- Otra acción habitual es mostrar contenido dentro de una etiqueta HTML. Para ello primero será necesario obtener el elemento HTML (por ejemplo, mediante su `id` con `document.getElementById`) y posteriormente cambiar el valor de la propiedad `innerHTML`.

```
<p id="demo"></p>

<script>

  document.getElementById("demo").innerHTML = "hola";

</script>
```

- Los comentarios en JavaScript pueden ser de una única línea o multilínea.

```
// Esto es un comentario de una única línea  
  
/*  
Esto es  
un comentario  
multilínea  
*/
```

- El uso del punto y coma es opcional en JavaScript.

```
<script>  
  
// correcto  
const nombre = "Alejandro";  
function saludar() {  
    document.write(nombre);  
}  
  
// error porque Javascript interpretaría:  
// const nombre = "Alejandro" (function saludar() {console.log('Hola'); })()  
const nombre = "Alejandro"  
(function saludar() {  
    document.write("Hola");  
})();  
  
// para evitarlo se añade el punto y coma  
const nombre = "Alejandro";  
(function saludar() {  
    document.write("Hola");  
})();  
  
</script>
```

- Los espacios en blanco entre asignaciones de variables son permitidos.

```
const person1="Hege";  
const person2 = "Hege";
```

- La función *prompt* permite pedir información por pantalla en el navegador.

```
<script>  
  
const nombre = prompt("Escriba su nombre", "Escriba aquí");
```

```
console.log(nombre);  
</script>
```

- La función *confirm* permite pedir confirmación por pantalla en el navegador.

```
<script>  
  
const conf = confirm("Sí o no");  
  
// true o false  
console.log(conf);  
  
</script>
```

- ◆ Existen diferentes palabras reservadas de JavaScript que no pueden utilizarse como nombres de variables.

\*break\*, \*case\*, \*catch\*, \*const\*, \*continue\*, \*debugger\*, \*do\*, \*else\*, \*finally\*,  
\*for\*, \*if\*, \*in\*, \*instanceof\*, \*let\*, \*new\*, \*switch\*, \*this\*, \*throw\*, \*try\*,  
\*typeof\*, \*void\*, \*while\*

- JavaScript es un lenguaje que puede resultar confuso respecto a algunas salidas.

```
// 1000000000000000000000000
console.log(999999999999999999999999);

// 0.6
console.log(0.5+0.1);

// 0.30000000000000004
console.log(0.1+0.2);

// 0.3
console.log(Math.round((0.1+0.2)*100)/100);
console.log(parseFloat((0.1+0.2).toFixed(1)));

// -Infinity
console.log(Math.max());

// Infinity
console.log(Math.min());

// ""
console.log([]+[]);
```

```
// "aa"
console.log(["a"]+["a"]);

// "[object Object]"
console.log([]+{});

// 0
console.log({}+[]);

// true
console.log(true+true+true === 3);

// true
console.log(true === 1);

// false
console.log(true === 1);

// 9
console.log((!+[]+[]+![]).length);

// "91"
console.log(9+"1");

// 90
console.log(91 - "1");

// true
console.log([] == 0);
```

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend

# (JavaScript)

Variables



# Variables

- ◆ [Introducción](#)
- ◆ [Valores especiales](#)
- ◆ [Uso de var y let](#)
- ◆ [Constantes](#)
- ◆ [Number](#)
- ◆ [Boolean](#)
- ◆ [String](#)
- ◆ [Array](#)
- ◆ [Date](#)
- ◆ [Conversión de tipos](#)
- ◆ [Comprobación de tipos](#)
- ◆ [Operadores](#)

## Introducción

- ◆ Las variables se declaran en JavaScript utilizando la partícula `let` o, preferiblemente, `const`.
- ◆ Las reglas generales para establecer nombres de variables (identificadores únicos) son:
  - Los nombres pueden contener letras, dígitos, guiones bajos y signos de dólar.
  - Los nombres deberían comenzar con una letra. También pueden comenzar con los caracteres `$` y `_`, pero no es una práctica recomendable.
  - Los nombres son sensibles a mayúsculas y minúsculas.
  - Las palabras reservadas (como palabras clave de JavaScript) no se pueden utilizar como nombres de variables.
- ◆ Convenciones con el nombre de variables.

```
// kebab-case (no permitido en JavaScript)
const last-name = 'Doe';

// snake_case (recomendado en otros lenguajes como Python)
const last_name = 'Doe';

// PascalCase (recomendado en JavaScript para las clases)
const LastName = 'Doe';

// camelCase (recomendado en JavaScript para las variables y funciones)
const lastName = 'Doe';
```

- ◆ Es una buena práctica declarar todas las variables al comienzo del script.

- Se pueden declarar varias variables al mismo tiempo utilizando la coma como separador.

```
const person = 'John Doe', carName = 'Volvo', price = 200;
```

- Los tipos son dinámicos.

```
// x está indefinido (undefined)
let x;

// ahora x es un número
x = 5;

// ahora x es un string
x = 'John';
```

## Valores especiales

- Valor especial Infinity: únicamente en operaciones con números.

```
// Infinity
const variable1 = 1/0;
```

- Valor especial NaN: únicamente cuando se realiza una conversión incorrecta a variable numérica.

```
// NaN
const variable2 = parseFloat('prueba');
```

- Variable especial undefined: indica que la variable ha sido declarada pero no se le ha asignado valor.

```
let variable4;

// undefined
console.log(variable4);

// error
console.log(variable_nodeclarada);
```

- ◆ Variable especial null: indica que la variable ha sido declarada y se le ha asignado un valor nulo.

```
const variable3 = null;
```

- ◆ null y undefined son similares, pero no son exactamente iguales.

```
// true
console.log(null == undefined);

// false
console.log(null === undefined );
```

- ◆ Estados que puede tomar una variable respecto a los valores especiales vistos con anterioridad:
  1. No declarada (error al utilizarla).
  2. Declarada pero sin valor (undefined).
  3. Declarada y con valor null.
  4. Declarada y con valor.

## Uso de var y let

- ◆ Con let no es posible declarar las variables dos veces.

```
var alert1 = 5;

// permitido
var alert1 = "";

let alert2 = 5;

// no permitido porque ya se declaró antes
let alert2 = 5;
```

- ◆ Con var se sobreescreiben las propiedades o métodos del objeto global (objeto window en el navegador o global en Node) si existe coincidencia de nombre.

```
// f alert() { [native code] }
console.log(window.alert);

// f alert() { [native code] }
console.log(alert);
```

```
// true
console.log(window.alert === alert);
```

```
var alert = 5;
// 5
console.log(alert);

// 5
console.log(window.alert);

// error
alert('Hello world!');

// error
window.alert('Hello world!');
```

```
let alert = 5;
// 5
console.log(alert);

// function
console.log(window.alert);

// error
alert('Hello world!');

// correcto
window.alert('Hello world!');
```

- Hay también más diferencias entre let y var con respecto al scope de las funciones y métodos.

## Constantes

- Las constantes son variables que no pueden modificar su valor.

```
const pi = 3.14;
// error
pi = 3;
```

## Number

- Las variables de tipo numérico en JavaScript permiten definir números enteros y de tipo flotante.

```
const numero0 = 1;
const numero1 = 20.1;

// 32
const numero2 = 3.2e1;

// 300
const numero3 = 3e2;

// hexadecimal
const numero4 = 0x1f;

// binario
const numero5 = 0b1010;

// octal
const numero6 = 0o744;

// otra forma de declarar una variable de tipo numérico
const numero7 = Number(2);
```

## Boolean

- Acepta dos posibles valores: true o false.

```
const boolean1 = true;
const boolean2 = false;
```

## String

- Las comillas tanto simples como dobles son permitidas para declarar variables de tipo string. Sin embargo son preferibles las comillas simples.

```
const str1 = "Hola a todos";
const str2 = 'Hola a todos';
const str3 = "";
const str4 = "";
```

- La propiedad length obtiene la longitud de un string.

```
const str1 = 'hola';
// 4
console.log(str1.length);
```

- El método indexOf busca un string dentro de otro string y devuelve la posición donde encuentra la coincidencia, donde la primera posición es la 0. Devuelve -1 en caso de no encontrar el String.

```
const str2 = 'hola';
// 3
console.log(str2.indexOf('a'));

// 1
console.log(str2.indexOf('ol'));

// -1
console.log(str2.indexOf('r'));
```

- El método substring extrae una parte del string en base a posiciones del mismo. Puede recibir como parámetro uno o dos números que representan la posición inicial y la final.
  - Si se recibe una posición, entonces se devuelve el fragmento del string a partir de esa posición y hasta el final.
  - Si se recibe dos posiciones, entonces se devuelve el fragmento del string a partir de la primera posición y hasta la segunda (sin incluirla).

```
const str3 = 'hola';
// 'lo'
console.log(str3.substring(1, 3));

// 'ola'
console.log(str3.substring(1));
```

- El método charAt obtiene el carácter de una determinada posición. Es similar a utilizar corchetes.

```
const str4 = 'hola';
// h
console.log(str4.charAt(0));
```

```
// a  
console.log(str4.charAt(3));  
  
//  
console.log(str4.charAt(4));  
  
// h  
console.log(str4[0]);  
  
// a  
console.log(str4[3]);  
  
// undefined  
console.log(str4[4]);
```

## Array

- Los arrays contienen un conjunto ordenado de variables agrupadas entre corchetes y separadas con comas.

```
const array1 = [];  
const array2 = [20, 3, 8];  
const array3 = ['Carmen', 'Juan'];  
  
// otra forma de declarar arrays  
const array4 = new Array(20, 3, 8);  
const array5 = new Array(null, undefined, "", 8);
```

- Puede accederse a alguna de las variables o elementos de un array mediante un índice o número encerrado entre corchetes. El índice denota la posición del elemento del array, siendo el índice 0 el primer elemento del array.

```
const array7 = new Array('Carmen', 'Juan');  
  
// Carmen  
console.log(array7[0]);  
  
// undefined  
console.log(array7[2]);
```

-Para obtener la longitud de un array se utiliza la propiedad length.

```
const array8 = new Array('Carmen', 'Juan');
```

```
// 2  
console.log(array8.length);
```

- ◆ Los arrays son dinámicos en longitud, es decir, pueden crecer en tamaño.

```
const array9 = new Array('Carmen', 'Juan');  
array9[3] = 'Alejandro';  
  
// [ 'Carmen', 'Juan', <1 empty item>, 'Alejandro' ]  
console.log(array9);
```

- ◆ La longitud de un array puede modificarse.

```
const array10 = new Array('Carmen', 'Juan');  
array10.length = 1;  
  
// [ 'Carmen' ]  
console.log(array10);
```

## Date

- ◆ JavaScript utiliza de forma predeterminada la zona horaria del navegador y muestra la fecha como un string. El valor de Date almacenado en Node.js no es el mismo.

```
const date1 = new Date();  
  
// Sun Apr 22 2018 12:37:06 GMT+0200 (Hora de verano romance)  
console.log(date1);  
  
// milisegundos en formato Unix  
const milisegundos = new Date().getTime();  
  
// fecha creada a partir del número de milisegundos en formato Unix  
const date2 = new Date(milisegundos);
```

## Conversión de tipos

- ◆ Conversión de string a number: con el método parseInt, con el operador unario (+) o con el objeto Number.

```
// 2  
const a = parseInt('2');
```

```
// 2
const b = +'2';

// NaN
const c = +'d';

// 2
const d = Number('2');
```

- Conversión de string a number con decimales: con el método parseFloat, con el operador unario (+) o con el objeto Number.

```
// 2.32
const a = parseFloat('2.32');

// 2.32
const b = +'2.32';

// 2.32
const c = Number('2.32');

// 0
const d = Number('');

// NaN
const e = Number('99 88');
```

- Conversión de number a string: con el método toString o el objeto String.

```
const a = 2;

// 2 (pero de tipo string)
console.log(a.toString());

// 2 (pero de tipo string)
console.log(String(a));
```

- Conversión de boolean a string: con el objeto String o con el método toString.

```
const a = false;

// false (pero de tipo string)
console.log(String(a));
```

```
// false (pero de tipo string)
console.log(false.toString());
```

- Conversión de boolean a number: con el objeto Number o con el operador unario (+).

```
// 0
console.log(Number(false));

// 1
console.log(Number(true));

// 1
console.log(+true);

// 0
console.log(+false);
```

- Conversión de date a string: con el objeto String o con el método toString.

```
// Sun Apr 22 2018 15:18:13 GMT+0200 (Hora de verano romance)
console.log(String(Date()));

// Sun Apr 22 2018 15:18:13 GMT+0200 (Hora de verano romance)
console.log(Date().toString());
```

- Conversión de date a number: con el objeto Number o con el método getTime.

```
const d = new Date();

// 1404568027739
console.log(Number(d));

// 1404568027739
console.log(d.getTime());
```

- Conversión automática: convertir tipos de datos incompatibles puede dar lugar a resultados inesperados.

```
// 5 porque null es convertido a 0
console.log(5 + null);

// 5null convertido null es convertido a null
console.log('5' + null);
```

```
// 52 porque 2 es convertido a string  
console.log('5' + 2);  
  
// 3 porque 5 es convertido a numérico  
console.log('5' - 2);  
  
// 3 porque 5 y 2 son convertidos a 5 y 2 en numérico  
console.log('5' - '2');
```

## Comprobación de tipos

- ◆ Para comprobar el tipo de variable se utiliza `typeof`.
- ◆ `typeof` puede devolver:
  - `number`
  - `string`
  - `boolean`
  - `function`
  - `object`
  - `undefined`

```
// number  
console.log(typeof 3);  
  
// number  
console.log(typeof NaN);  
  
// number  
console.log(typeof parseInt('hola'));  
  
// string  
console.log(typeof 'hola');  
  
// boolean  
console.log(typeof true);  
  
// function  
console.log(typeof function() {});  
  
// object  
console.log(typeof {foo: 'bar'});  
  
// object  
console.log(typeof ['a', 'b', 'c']);  
  
// object  
console.log(typeof new Date());
```

```
// object  
console.log(typeof null);  
  
// undefined  
console.log(typeof undefined);  
  
// undefined  
console.log(typeof unknown);
```

- ◆ Existen principalmente dos formas para comprobar si una variable es de tipo array.

```
const myArray = ['a', 'b', 'c'];  
  
// true  
console.log(Array.isArray(myArray));  
  
// true  
console.log(fruits instanceof Array);
```

- ◆ Con las variables de tipo date puede utilizarse instanceof.

```
const fecha = new Date();  
  
// true  
console.log(fecha instanceof Date);
```

- ◆ También existen métodos específicos para comprobar si una variable tiene el valor NaN (isNaN) o es finito (isFinite).

```
// true  
console.log(isNaN(parseInt('hola')));  
  
// false  
console.log(isFinite(1/0));
```

## Operadores

- ◆ Operadores aritméticos: suma (+), resta (-), multiplicación (\*), división (/), módulo (%), incremento (++) y decremento (--).

```
// suma  
const a = 2 + 3;
```

```
// resta
const b = 2 - 3;

// multiplicación
const c = 2 * 3;

// división
const d = 2 / 3;

// módulo
const e = 2 % 3;

// incremento posterior
a++;

// incremento anterior
++a;

// decremento posterior
a--;

// decremento anterior
--a;

const x1 = 4;

// y1 = 5; x1 = 5 (primero incrementa y luego asigna)
const y1 = ++x1;

const x2 = 4;

// y2 = 4; x2 = 5 (primero asigna y luego incrementa)
const y2 = x2++;
```

- Operadores de asignación: suma (`+=`), resta (`-=`), multiplicación (`\*=`), división (`/=`) y módulo (`%=`).

```
let a = 2;

// a = a + 2
a += 2;

// a = a - 2
a -= 2;

// a = a * 2
a \*= 2;
```

```
// a = a / 2
a /= 2;

// a = a % 2
a %= 2;
```

- ◆ Operadores de comparación: igual que (==), distinto que (!=), exactamente igual que (===), exactamente distinto que (!==), menor que (<), igual o menor que (<=), mayor que (>), mayor o igual que (>=).
- ◆ El comparador == realiza una comparación de valor únicamente, mientras que === realiza una comparación de valor y también de tipo.

```
// true
console.log('2' == 2);

// false
console.log('2' === 2);

// true porque ambos valores representan un objeto vacío
console.log(null == undefined);

// false porque no son el mismo objeto
console.log(undefined === null);

// false
console.log('2' != 2);

// true
console.log('2' !== 2);

// true porque '2' es convertir a number
console.log('2' < 4);

// false
console.log(4 > 8);

// true
console.log(3 <= 3);

// true
console.log(3 >= 3);
```

- ◆ Operadores Lógicos: AND (&&), OR (||) y NOT (!).

```
// true
console.log(true && true);
```

```
// false  
console.log(true && false);  
  
// false  
console.log(false && true);  
  
// false  
console.log(false && false);  
  
// true  
console.log(true || true);  
  
// true  
console.log(true || false);  
  
// true  
console.log(false || true);  
  
// false  
console.log(false || false);  
  
// false  
console.log(!true);  
  
// true  
console.log(!false);
```

- En JavaScript todas las variables con valores son true si se realiza una conversión a booleano, excepto 0, NaN, null, undefined y una cadena vacía.

```
// false  
console.log(Boolean(0));  
  
// true  
console.log(Boolean(4));  
  
// true  
console.log(Boolean('Hola'));  
  
// false  
console.log(Boolean(""));  
  
// false  
console.log(Boolean(""));
```

- Con el operador OR (||) es habitual asignar valores a variables en función de la existencia o no de otra variable.

```
// port toma el valor 5000 si process.env.PORT toma el valor de NaN, null o undefined  
const puerto = puertoPorDefecto || 5000;
```

- ♦ Operador ternario (?): asigna un valor u otro a una variable en base a una condición.

```
// a = 4  
const a = true ? 4 : 3;
```

```
// a = 3  
const b = false ? 4 : 3;
```

- ♦ Operador concatenación (+): permite unir varios string.

```
const str1 = 'hola';  
  
// hola a todos 5true  
const str2 = str1 + ' a todos' + 5 + true;  
  
// hola a todos 5true  
console.log(str2);  
  
// 20  
console.log(+13 + 7);  
  
// 137  
console.log('13' + 7);
```

- ♦ En ECMAScript 6 se introdujo las comillas invertidas, llamadas template literal o plantillas de cadenas.

```
const str1 = 'Hola';  
const str2 = 'todos';  
const str = `${str1} a ${str2}`;  
  
// Hola a todos  
console.log(str);
```

Ejercicio 1: escribe un programa que declare tres variables de nombre a, b y c, con valores de tipo number. A continuación:

1. Escribe una sentencia que muestre por pantalla la suma de las tres variables utilizando console.log

2. Cambia el valor de la variable c.
3. Escribe de nuevo una sentencia que muestre por pantalla la suma de las tres variables utilizando console.log

Ejercicio 2: escribe un programa en JavaScript que realice las siguientes tareas:

1. Mostrar por pantalla Hello world.
2. Mostrar por pantalla la expresión  $2*3$  como texto.
3. Mostrar por pantalla el resultado de la expresión  $2*3$ .

Ejercicio 3: realiza la siguientes conversiones entre variables de diferente tipo:

1. De string a number.
2. De number a string.
3. De boolean a string.

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas frontend

## (JavaScript)

Control de flujo



# Control de flujo

- ◆ Condicionales - if
- ◆ Condicionales - switch
- ◆ Bucles
  - Bucle for
  - Bucle for/in
  - Bucle for/of
  - Métodos de los arrays
  - Bucle while
- ◆ Manejo de excepciones

## Condicionales - if

- ◆ Ejecuta una sentencia si una condición establecida es evaluada como verdadera. Si la condición es evaluada como falsa, otra sentencia puede ser ejecutada.
- ◆ JavaScript utiliza la partícula if para evaluar condiciones.
- ◆ El condicional más simple incluye un if con una única condición.

```
const a = 4;
const b = 2;

if (a > b) {
  console.log(`a (${a}) es mayor que b (${b})`);
}
```

- ◆ Pueden establecerse varias condiciones anidadas con else if.

```
const a = 4;
const b = 2;

if (a > b) {
  console.log(`a (${a}) es mayor que b (${b})`);
}
else if (a < b) {
  console.log(`b (${b}) es mayor que b (${a})`);
}
else if (a === b) {
  console.log(`b (${b}) es igual y posee el mismo tipo que b (${a})`);
}
```

- ◆ Y también una condición de else si las anteriores condiciones no se cumplen.

```
const a = 4;
const b = 2;

if (a > b) {
    console.log(`a (${a}) es mayor que b (${b})`);
}
else if (a < b) {
    console.log(`b (${b}) es mayor que b (${a})`);
}
else {
    console.log(`b (${b}) es igual que b (${a})`);
}
```

Ejercicio 1: probar si las siguientes evaluaciones devuelven true o false.

```
4.3 >= 4
1 == 2
4 < 4
2 !== 5
5 == '5'
5 === '5'
5 == 5
5 === 5
```

Ejercicio 2: escribe un programa con cuatro variables de tipo number (a, b, c y d) y un condicional que imprima por pantalla si la suma de a y b es mayor que la suma de c y d.

Ejercicio 3: escribe un programa que almacene tres ángulos de un triángulo en variables de tipo entero (angulo1, angulo2 y angulo3). Crea un condicional que compruebe si esos tres ángulos juntos pueden formar un triángulo (los ángulos de un triángulo suman siempre 180 grados).

Ejercicio 4: escribe un programa con una variable de tipo number (a) y un condicional que evalúe si el entero es par o impar utilizando el operador %.

Ejercicio 5: escribe un programa que dado tres números imprima por pantalla cuál es el mayor.

Ejercicio 6: ¿cuál es el resultado de ejecutar el siguiente código?

```
const i = 25;

if(i == 25) {
    console.log("a");
}
else {
```

```
    console.log("b");
}
```

Ejercicio 7: ¿cuál es el resultado de ejecutar el siguiente código?

```
const i = 25;

if(i == 25) {
    console.log("a");
}
if(i == 24) {
    console.log("b");
}
else {
    console.log("c");
}
```

## Condicionales - switch

- El switch es una alternativa a los condicionales.
- Con un switch, una vez obtenido una comprobación exitosa no se realizan más evaluaciones y todo el código sucesivo es ejecutado hasta el final del switch o hasta encontrar un break.
- break permite detener la ejecución del código dentro del switch.
- La última comprobación que se realiza no requiere de break.
- La palabra clave default (no es obligatoria establecerla) establece el bloque de código que se ejecutará si no hay coincidencia del resto de evaluaciones.

```
const a = 4;
switch (a) {
    case 1:
        console.log('El valor de a es 1');
        break;
    case 2:
        console.log('El valor de a es 2');
        break;
    case 3,4:
        console.log('El valor de a es 3 ó 4');
        break;
    default:
        console.log('El valor de a es desconocido');
}
```

## Bucles

- Existen diferentes tipos de bucles en JavaScript:
  - for clásico: recorre un bloque de código varias veces.
  - for/in: recorre las propiedades de un objeto.
  - for/of: recorre los elementos contenidos en un array.
  - métodos de los arrays: forEach, map, filter, find, reduce, etc.
  - while: ejecuta un bloque de código mientras que una condición establecida es verdadera.
  - do/while: recorre un bloque de código mientras una condición establecida es verdadera, pero el código del bloque es ejecutado siempre al menos una vez.

### Bucle for

- Compuesto por tres sentencias separadas por punto y coma.
  - Primera sentencia:
    - Define la inicialización.
    - Es opcional.
    - Pueden inicializarse varias variables (separadas por coma)
  - Segunda sentencia:
    - Evalúa la condición de la variable inicializada en cada iteración. Si la condición es false, se finaliza el bucle for. En caso contrario, se sigue ejecutando.
    - Es opcional, aunque si es omitido, entonces el bucle no terminará nunca excepto si existe una instrucción de break en el interior del bucle.
  - Tercera sentencia:
    - Incrementa el valor de la variable inicializada en cada iteración. Habitualmente el incremento es de 1 (i++), pero es posible otras combinaciones (i--, i+=15, etc)
    - Es también opcional y puede modificarse el valor de la variable inicializada desde dentro del bucle.

```

for (let i = 0; i < 5; i++) {
  // 0, 1, 2, 3, 4 (en distintas líneas)
  console.log(i);
}

const array = ['En', 'un', 'lugar', 'de', 'la', 'mancha'];
const len = array.length;

// recorre un array mediante un for clásico
for (let i = 0; i < len; i++) {

  // En, un, lugar, de, la, mancha (en distintas líneas)
  console.log(array[i]);
}

```

Ejercicio 8: escribe un programa que muestre todos los números desde el 1 al 50.

Ejercicio 9: escribe un programa que imprima toda la tabla de multiplicar del 5 (desde 0 hasta 10).

Ejercicio 10: escribe un programa que pida al usuario una palabra (mediante la función prompt) y lo muestre por pantalla 10 veces.

Ejercicio 11: escribe un programa que pida al usuario un número entero positivo (mediante la función prompt) y muestre por pantalla la cuenta atrás desde ese número hasta cero, utilizando comas como separación.

## Bucle for/in

- Será visto posteriormente en el apartado de objetos.

## Bucle for/of

- Será visto posteriormente en el apartado de ES6.

## Métodos de los arrays

- Será visto posteriormente en el apartado de ES6.

## Bucle while

- Ejecuta un bloque de código siempre que una condición establecida sea verdadera.

```
// while
let contador1 = 0;
while (contador1 <= 5) {
    contador1++;
}

// 6
console.log(contador1);

// do-while
let contador2 = 0;
do {
    contador2++;
} while (contador2 <= 5);

// 6
console.log(contador2);
```

## Manejo de excepciones

- Al igual que en otros lenguajes de programación, el manejo de excepciones en JavaScript se realiza con el bloque try-catch.

```
try {  
    adddlert('a');  
}  
catch (err) {  
  
    // ReferenceError: adddlert is not defined  
    alert(err);  
}  
  
// lanzar excepción personalizada  
throw 'Too big';
```

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas

# frontend

# (JavaScript)

Funciones



# Funciones

- ◆ [Introducción](#)
- ◆ [Formas de declarar una función](#)
- ◆ [Callbacks](#)
- ◆ [Los objetos this y arguments en una función](#)
- ◆ [El scope en las funciones](#)

## Introducción

- ◆ Existen diferentes formas de declarar funciones en JavaScript.
- ◆ La función de flecha es la forma moderna de declarar una función en JavaScript.
- ◆ Independientemente de cómo son declaradas, las funciones en JavaScript:
  - Declaran los parámetros que reciben mediante comas.
  - Pueden retornar cualquier tipo de valor mediante return o simplemente no retornar nada (retornarían undefined).
  - En la invocación se pueden pasar por parámetros cualquier tipo y número de variables.
    - Si se pasan más parámetros de los declarados, entonces la función ignorará al resto.
    - Si se pasan menos parámetros de los declarados, entonces la función asignará el valor de undefined al resto.

## Formas de declarar una función

- ◆ Función declarada.

```
function sumar(a, b) {  
    console.log(a + b);  
}  
  
// 5  
sumar(2, 3);  
  
// NaN (undefined + undefined)  
sumar();  
  
// 3  
sumar(1, 2);  
  
// 3  
sumar(1, 2, 3, 4);  
  
// 12
```

```
sumar('1', 2);  
  
// 12  
sumar(1, '2');
```

- ◆ Función anónima (sin nombre) asignada a una variable.

```
const sumar = function(a, b) {  
    console.log(a + b);  
};  
  
// 5  
sumar(2, 3);
```

- ◆ Función método.

```
const a = {  
    f: function() {  
        console.log('Hola');  
    }  
};  
  
// invocación utilizando la notación punto  
a.f();  
  
// invocación utilizando la notación con corchetes  
a['f']();
```

- ◆ Función autoejecutable que se ejecuta inmediatamente y en un contexto separado.

```
(function saludar() {  
    console.log('Hola');  
})();  
  
// error  
saludar();
```

- ◆ Función creadora de objetos mediante la partícula new, donde la función es considerada el constructor del objeto (closure).

```
function hacerAlgo(a, b) {  
    this.a = a;
```

```
this.b = b;  
}  
  
// creación del objeto  
const b = new hacerAlgo(1,2);  
  
// error porque b es un objeto  
b();  
  
// 1  
console.log(b.a);  
  
// 2  
console.log(b.b);  
  
// undefined  
console.log(this.a);
```

- Función de flecha (arrow function) incorporadas en ES6.

```
// función normal  
function funcion(altura) {  
    return 5 * altura / 2;  
};  
  
// función de flecha  
const funcion1 = (altura) => {  
    return 5 * altura / 2;  
};  
  
// forma simplificada de la función de flecha  
const funcion2 = altura => 5 * altura / 2;
```

Ejercicio 1: escribir una programa con cuatro funciones de flecha: sumar, restar, multiplicar y dividir (cada una de ellas debe aceptar dos parámetros y devolver el resultado). Posteriormente invocar a las cuatro funciones con valores arbitrarios.

Ejercicio 2: adaptar el programa del ejercicio anterior simplificando las funciones de flecha eliminando las llaves y el return.

## Callbacks

Generalmente, las funciones suelen devolver un valor que es almacenado posteriormente en una variable tras la invocación de la función.

```
const sumar = (a,b) => {  
    return a + b;
```

}

```
const resultado = sumar(1,2);
console.log(resultado);
```

- ◆ Sin embargo, en JavaScript es común utilizar los llamados callbacks.
- ◆ Los callbacks son funciones que se invocan desde dentro de otra función.

```
// función que espera recibir un callback (función que es ejecutada posteriormente
dentro del cuerpo de la función sumar)
const sumar = (a, b, callback) => {
    callback(a + b);
};

// declaración del callback donde se espera el resultado de la operación
const callback = (resultado) => {
    console.log(resultado);
}

// invocación de la función sumar, pasando por parámetros los valores 1, 2 y el callback
sumar(1, 2, callback);
```

- ◆ Normalmente el callback suele estar integrado dentro de la propia invocación.

```
const sumar = (a, b, callback) => {
    callback(a + b);
};

// ahora el callback está integrado en la invocación de la función smar
sumar(1, 2, (resultado) => {
    console.log(resultado);
});
```

Ejercicio 3: adaptar las cuatro funciones del Ejercicio 2 para que reciban un callback que sea invocado dentro de la función con el resultado de la operación. Posteriormente invocar a las cuatro funciones creando una función de callback por separado y pasándola después por parámetro. Utilizar la herramienta de depuración de Chrome (Herramientas de desarrollador --> Source) para comprobar cómo se ejecuta el programa.

Ejercicio 4: adaptar el ejercicio anterior para integrar el callback en la invocación de las cuatro funciones, en lugar de creándola como una función de callback por separado. Utilizar la herramienta de depuración de Chrome para comprobar cómo se ejecuta el programa.

- Los callback son utilizados principalmente para funciones que no devuelven inmediatamente la respuesta. En estos casos, los callback ayudan a evitar que el código se detenga en la invocación y la respuesta sea gestionada de forma asíncrona.
- Todos los ejemplos anteriores no son asíncronos porque no utilizan funciones de JavaScript puramente asíncronas.
- `setTimeout` y `setInterval` son las funciones asíncronas de JavaScript más simples.

```
setTimeout(() => {
  console.log('Esta instrucción se ejecuta después de 5 segundos');
}, 5000);

setInterval(() => {
  console.log('Esta instrucción se ejecuta cada 5 segundos');
}, 5000);
```

- Estas funciones `setTimeout` y `setInterval` pueden también anidarse.

```
setTimeout(() => {
  console.log('Hola');
  setTimeout(() => {
    console.log('Adiós');
  }, 500);
}, 1000)
setTimeout(() => {
  console.log('Buenas tardes');
}, 100)
console.log('Buenos días');
```

Ejercicio 5: adaptar las cinco funciones del Ejercicio 4 para que las cuatro funciones invoquen el callback una vez han transcurrido 1 segundo (para la función sumar), 2 segundos (para la función restar), 3 segundos (para la función multiplicar) y 4 segundos para dividir (para la función dividir). Añadir un `console.log` en la última línea del código del programa. Utilizar la herramienta de depuración de Chrome para comprobar cómo se ejecuta el programa.

- JavaScript está especialmente orientado al desarrollo de aplicaciones asíncronas.

```
/*
 código síncrono
*/
const consultarDatabase = () => {
  const startPoint = new Date().getTime();
  while (new Date().getTime() - startPoint <= 2000);
```

```

    return "Consulta realizada";
}

console.log("Primera consulta al servidor");
const consulta1 = consultarDatabase();
console.log(consulta1);

console.log("Segunda consulta al servidor");
const consulta2 = consultarDatabase();
console.log(consulta2);

console.log("Más tareas a realizar...");

```

```

/*
  código asíncrono
*/
const consultarDatabase = (callback) => {
  setTimeout(() => {
    callback("Consulta realizada");
  }, 2000);
}

console.log("Primera consulta al servidor");
consultarDatabase(function(consulta) {
  console.log(consulta);
});

console.log("Segunda consulta al servidor");
consultarDatabase(function(consulta) {
  console.log(consulta);
});

console.log("Más tareas a realizar...");

```

## Los objetos this y arguments en una función

- El objeto this apunta al objeto global (window en el caso de los navegadores o global en Node.js) fuera de una función.
- this sigue referenciando al objeto global dentro de una función.
- Las variables que se pasan por argumentos en la invocación de una función normal se comportan como variables locales dentro de la misma.

```

const hacerAlgo = (c, d) => {
  // apunta al objeto global

```

```
console.log(this);

// la variables se convierten en globales
this.a = c;
this.b = d;

// a
console.log(a);

// b
console.log(b);
}

const resultado = hacerAlgo('a', 'b');

// a
console.log(a);

// b
console.log(b);

// error
console.log(c);

// error
console.log(d);
```

- Utilizar new para invocar a una función creará un objeto a partir de la función y el objeto this ya no referenciará al objeto global dentro de la misma, sino que hará referencia al propio objeto.

```
function hacerAlgo(c, d) {
```

```
// hacerAlgo {}
console.log(this);
```

```
this.a = c;
this.b = d;
```

```
// a
console.log(this.a);
```

```
// undefined
console.log(this.c);
```

```
// error
// console.log(a);
```

```
// error
```

```
// console.log(b);
}

let resultado = new hacerAlgo('a', 'b');

// a
console.log(resultado.a);

// b
console.log(resultado.b);

// error
// console.log(a);

// error
// console.log(b);

// error
// console.log(c);

// error
// console.log(d);
```

- ◆ En JavaScript las funciones no pueden ser sobrecargadas (funciones con el mismo nombre y distinto número de argumentos). Una función con el mismo nombre que otra función la sobreescribe.
- ◆ Sin embargo, el número de argumentos suministrados en la invocación no necesariamente debe coincidir con los establecidos en la declaración de la función.

```
function saludar(saludo) {
  if (!saludo) console.log('no hay saludo');
  else console.log(saludo);
}

// no hay saludo
saludar();

// hola
saludar('hola');

// hola
saludar('hola', 'Alejandro');
```

- ◆ Dentro de la función se puede acceder también a los argumentos mediante el objeto arguments.

```
function saludar() {  
  
    const saludo = arguments[0];  
    const nombre = arguments[1];  
  
    // undefined  
    console.log(arguments[2]);  
  
    // Adiós, Carlos!  
    console.log(saludo + ', ' + nombre + '!');  
}  
  
saludar('Adiós', 'Carlos');
```

- Al igual que las variables declaradas con var (no con las declaradas con let), al crear una función, ésta también se añade al contexto (this) y al objeto window en el navegador.

```
function hola() {  
    console.log('hola');  
}  
  
// hola  
console.log(window.hola());  
  
// hola  
console.log(this.hola());  
  
// hola  
hola();
```

- En JavaScript no es necesario especificar el tipo de dato que será retornado en la función. Una función puede retornar, por ejemplo, otra función.

## El scope en las funciones

- JavaScript tiene un scope o ámbito a nivel de función. El scope determina la accesibilidad (visibilidad) de estas variables.
- En JavaScript hay dos tipos de alcance: scope local, scope global.
- Las variables declaradas con var o con let dentro de una función pertenecen al scope local y no son accesibles (visibles) desde fuera de la función.
- Las variables globales se declaran sin utilizar var, aunque su uso es poco recomendado. Sin embargo, una variable declarada como var en el ámbito global también se convierte en variable global.

- ◆ Las variables locales declaradas en una función se eliminan cuando finaliza la función. En un navegador web, las variables globales se eliminan cuando se cierra la ventana (o pestaña) del navegador.
- ◆ ¿Por qué se hizo necesario la inclusión de let?
- ◆ Variable local declarada en el ámbito de una función.

```
function saludar() {  
    var edad = 14;  
}  
saludar();  
  
// error  
console.log(edad);
```

- ◆ Variable global declarada en el ámbito de una función.

```
function saludar() {  
    edad = 25;  
}  
  
function saludar2() {  
    // 25  
    console.log(edad);  
}  
  
saludar();  
saludar2()  
  
// 25  
console.log(edad);
```

- ◆ Variable local declarada en el ámbito global (se convierte en variable global).

```
var edad = 25;  
  
function saludar() {  
    // 25  
    console.log(edad);  
}  
  
saludar();
```

- ◆ Variable global declarada en el ámbito global (funciona igual que el anterior).

```
edad = 25;

function saludar() {
    // 25
    console.log(edad);
}

saludar();
```

- Variable global que es sobrescrita dentro de una función.

```
edad = 25;

function saludar() {

    // Javascript busca el scope superior
    edad = 27;

    // 27
    console.log(edad);
}

saludar();

// 27
console.log(edad);
```

- Cuando se utiliza la partícula var en la declaración de una variable, JavaScript ubica esta declaración al comienzo de la función o al comienzo del script. No sucede lo mismo cuando se declara las variables sin var. Esto es conocido como Hoisting.
  - Este comportamiento de JavaScript es desconocido para muchos desarrolladores.
  - Por tanto, si se utiliza var es recomendable declarar todas las variables al comienzo del bloque o de la función.

```
function saludar1() {
    // error
    console.log(edad);
}

saludar1();

function saludar2() {
    // undefined
    console.log(edad);
    var edad = 25;
```

}

saludar2();

- El Hoisting es problemático y con let/const no sucede.

```
function saludar1() {  
  
    // var i;  
  
    for (var i = 0; i < 10; i++) {  
        console.log('Hola');  
    }  
  
    // 10, aunque se podría pensar que i en este punto no debería existir porque se creó en  
    // el ámbito del for  
    console.log(i);  
}  
  
saludar1();  
  
function saludar2() {  
  
    for (let i = 0; i < 10; i++) {  
        console.log('Hola');  
    }  
  
    // error porque con let no hay Hoisting  
    console.log(i);  
}  
  
saludar2();
```

- Otro ejemplos en el ámbito global.

```
// var b;  
// var a;  
  
// undefined  
console.log(b);  
var b = 4;  
  
// error porque en las variables globales no hay Hoisting  
console.log(a);  
a = 4;  
  
function varTest() {
```

```
var x = 31;

if (true) {

    // misma variable que la anterior declarada
    var x = 71;

    // 71
    console.log(x);
}

// 71
console.log(x);
}
varTest();
```

- ◆ Para solucionar estos problemas se introdujo en ECMAScript 6 la partícula let. De esta forma, es conveniente no volver a utilizar var, siempre let o aún mejor, const cuando la variable es constante.
- ◆ Utilizar var puede tener sentido cuando se emplea la consola o cuando se pretende hacer uso del Hoisting.

```
function saludar() {

    for (let j = 0; j < 10; j++) {
        console.log(j);
    }

    // error porque la variable j solamente existe en el ámbito del for
    console.log(j);
}
saludar();
```

```
// error porque con let no hay Hoisting
console.log(a);
let a = 4;

function varTest1() {

    let x = 31;

    if (true) {

        // error porque la variable x todavía no existe
        console.log(x);
```

```
let x = 71;  
  
// 71  
console.log(x);  
}  
  
// 31  
console.log(x);  
}  
  
function varTest2(){  
  
let x = 31;  
  
if (true) {  
  
// 31  
console.log(x);  
}  
  
// 31  
console.log(x);  
}
```

```
function letTest() {  
  
let x = 31;  
  
if (true) {  
  
// error porque la variable x va a ser declarada posteriormente y tiene preferencia  
respecto a la declarada anteriormente  
console.log(x);  
  
// variable diferente a la anterior declarada  
let x = 71;  
  
// 71  
console.log(x);  
}  
  
// 31  
console.log(x);  
}
```

- ◆ Otro problema que soluciona let es la sobrescritura que realiza var sobre las propiedades del objeto global.

```
// function
console.log(window.alert);

var alert = 2;

// 2
console.log(window.alert);

// 2
console.log(alert);
```

```
// function
console.log(window.alert);

let alert = 2;

// function
console.log(window.alert);

// 2
console.log(alert);
```

- ◆ Let se verá con más detalle en la parte de ECMAScript 6.

Ejercicio: probar los anteriores ejemplos para entender cómo funciona el scope en JavaScript.

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas frontend

## (TypeScript)

Fundamentos

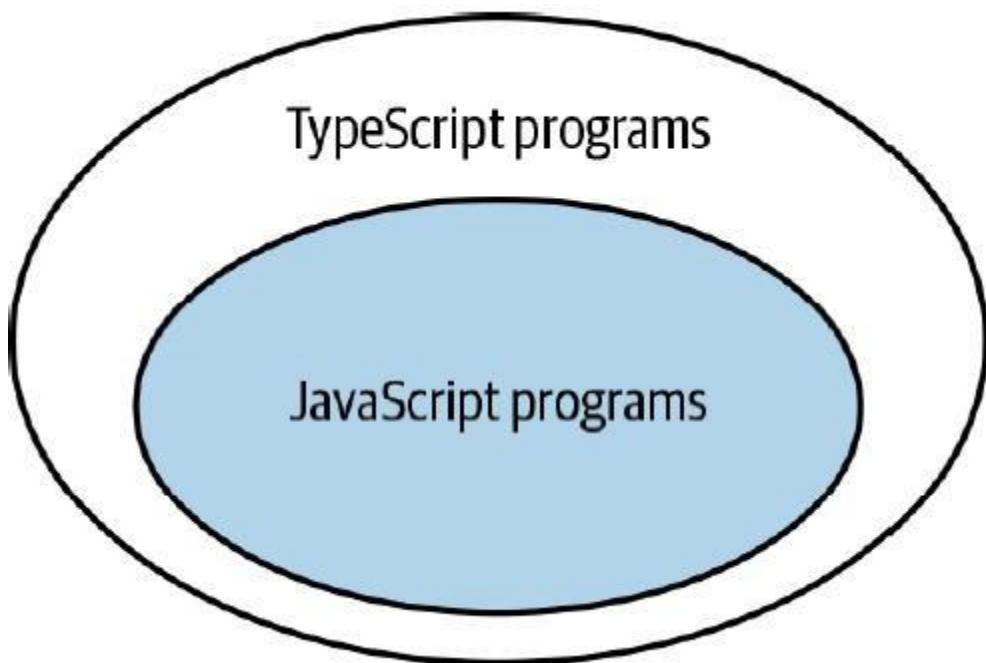


# Fundamentos

- ◆ [Introducción](#)
- ◆ [Primer programa en TypeScript](#)
- ◆ [Uso de TypeScript dentro de un proyecto Angular](#)
- ◆ [@types](#)
- ◆ [Comando tsc](#)
- ◆ [Depuración](#)
- ◆ [Parámetros de comprobación estricta en TypeScript](#)
- ◆ [Parámetros de calidad de código en TypeScript](#)

## Introducción

- ◆ TypeScript es un lenguaje de programación de código abierto y gratuito desarrollado por Microsoft.
- ◆ El código TypeScript no puede ejecutarse directamente, sino que necesita ser transpilado al lenguaje JavaScript.
- ◆ Toda la sintaxis de JavaScript está soportada por TypeScript. Adicionalmente, TypeScript añade un conjunto de características no disponibles en JavaScript, como el tipado.
- ◆ En definitiva, TypeScript es un superconjunto (*superset*) de JavaScript.



- ◆ Algunas ventajas que proporciona TypeScript con respecto a JavaScript son:
  - Tipado.
  - Mejor auto-completado.
  - Clases y módulos más completos.
  - Detección cuando una variable no está definida.

- Detección cuando un objeto no posee una determinada propiedad.
  - Detección cuando se desconoce cómo trabaja una función (parámetros de entrada, salida y tipos).
  - Detección de errores en la sintaxis del lenguaje.
  - Detección de malas prácticas escribiendo código.
- ◆ El tipado de TypeScript permite evitar comportamientos no esperados en la ejecución del código.

```
/*
 Código en JavaScript
 */

const sumar = (numero1, numero2) => {
    return numero1 + numero2;
}

// devuelve 3
sumar(1, 2);

// devuelve 12 (comportamiento no esperado)
sumar('1', '2')
```

- ◆ Evitar este tipo de problemas en JavaScript puede ser tedioso.

```
/*
 Código en JavaScript
 */

const sumar = (numero1, numero2) => {

    if(typeof(numero1) !== 'number' || typeof(numero2) !== 'number') {
        return console.log('Al menos uno de los valores no es de tipo numérico');
    }
    return numero1 + numero2;
}
```

- ◆ El tipado en TypeScript permite que el código sea más claro y limpio.

```
/*
 Código en TypeScript
 */

const sumar = (numero1: number, numero2: number) => {
    return numero1 + numero2;
}
```

```
// devuelve 3  
sumar(1, 2);  
  
// error en TypeScript porque la función sumar espera dos variables de tipo numérico  
// sumar('1', '2')
```

- La página web oficial de TypeScript presenta un [editor](#) muy útil para comprobar el código convertido a JavaScript.
- También existen [repositorios](#) muy completos con multitud de recursos relacionados con TypeScript.

## Primer programa en TypeScript

- El primer paso es instalar Typescript de forma global en el sistema (parámetro `-g`) con el gestor de paquetes `npm` de [Node.js](#).

```
npm install -g typescript
```

- También es recomendable instalar TypeScript como dependencia de desarrollo en todos los proyectos en los que se trabajen, dado que si se actualiza TypeScript a nivel global, se mantienen las versiones locales de los proyectos.
  - Esto evita posibles problemas de incompatibilidad que puede acarrear una nueva versión de TypeScript en proyectos que ejecutan versiones antiguas.

```
# inicia un proyecto Node  
npm init  
  
# instala la typescript como dependencia local de desarrollo  
npm install --save-dev typescript
```

- Puede comprobarse la versión de TypeScript instalada en el sistema mediante la opción `-v`

```
tsc -v
```

- El siguiente paso es escribir un programa simple en TypeScript (con extensión `ts`).

```
// main.ts  
  
const saludos = (persona: string) => {  
    return "Hola, " + persona;
```

}

```
const usuario: string = "Marcos";
console.log(usuario);
document.body.innerHTML = saludos(usuario);
```

- A continuación, se transpila el archivo de código mediante el comando *tsc*

```
tsc main.ts
```

- *tsc* genera un archivo de JavaScript transpilado con el mismo nombre que el archivo de TypeScript, pero con la extensión de los archivos JavaScript (*js*).

```
// main.js

var saludos = function (persona) {
    return "Hola, " + persona;
};
var usuario = "Marcos";
console.log(usuario);
document.body.innerHTML = saludos(usuario);
```

## Uso de TypeScript dentro de un proyecto Angular

- TypeScript es el lenguaje de programación utilizado para la construcción de aplicaciones del framework web Angular.
- Angular realiza automáticamente tareas como la transpilación de código TypeScript o el refresco automático de la página tras la modificación del código (tareas que en otras circunstancias tiene que realizarse de forma manual).
- Por tanto, Angular es una herramienta ideal para comenzar a aprender a TypeScript dado que proporciona todas las herramientas necesarias para comenzar a escribir código de la forma más simple y cómoda posible.

```
# instalación de Angular
npm install -g @angular/cli

# creación de un proyecto de Angular llamado my-app
# durante la creación del proyecto --> Routing: NO y stylesheet: CSS
ng new my-app

# acceso al directorio que contiene el proyecto de Angular recién creado
cd my-app
```

```
# ejecución de la aplicación  
ng serve
```

- La aplicación de Angular estará disponible en la dirección <http://localhost:4200>
- El código TypeScript puede comenzar a escribirse en el archivo  
`./src/app/app.component.ts`

## @types

- Definitely Typed (@types) es una de las mayores fortalezas de TypeScript. Se trata de paquetes *npm* que añaden tipado a las librerías más importantes de JavaScript.
- Por ejemplo, para utilizar JQuery con TypeScript puede instalarse la librería `@types/jquery`

```
npm install @types/jquery --save-dev
```

```
// main.tsc  
  
$( "button.continue" ).html( "Next Step..." );
```

```
tsc main.tsc
```

- También se puede transpilar varios archivos de TypeScript al mismo tiempo.

```
tsc archivo1.ts archivo2.ts archivo3.ts
```

- Antes que los `@types` se utilizaban los `tsd` y posteriormente los `Typings` (ambos están obsoletos a día de hoy).
- Para obtener el paquete `@types` asociado a una librería de Javascript puede accederse a la página web oficial de @types o al buscador de Microsoft. Aunque también se puede buscar directamente en Google o en el directorio *npm*

## Comando *tsc*

- El comando *tsc* permite también la opción de monitorizar cambios en archivos TypeScript (*watch*), transpilando automáticamente el código a JavaScript.

- Es posible definir un archivo de configuración llamado `tsconfig.json` en el que puede indicarse distintas opciones de transpilación. Algunos de los parámetros dentro de la clave `compilerOptions` son:

- `target`: versión de ECMAScript a transpilar: `ES3` (valor por defecto), `ES5`, `ES6/ES2015`, `ES7/ES2016`, `ES2017`", `ES2018`, `ES2019`, `ES2020` o `ESNext` (últimas novedades).
- `rootDir`: directorio donde se ubican los archivos TypeScript.
- `outDir`: directorio donde se ubicarán los archivos JavaScript tras la transpilación. No se copiarán al directorio `outDir` los archivos contenidos en el directorio `rootDir` que no pueden ser transpilados (por ejemplo, los archivos `html` o `css`).
- `module`: establece el sistema de módulos que se utilizará en la transpilación. El valor por defecto es "commonjs" (habitual en proyectos Node mediante el uso de `exports` y `require`), aunque también se puede utilizar la forma moderna de utilizar módulos (valor `ESNext`) con `import` y `export` (habitual en proyectos TypeScript y en Angular)
- `lib`: incluye un conjunto de tipos específicos de alguna librería de JavaScript (DOM, Webworker, alguna versión específica de ECMAScript, ...). Este parámetro `/lib` está muy relacionado con `target`. Asignar el valor `es6` a `target`, incluirá las siguientes librerías (parámetro `/lib`): `dom`, `es6`, `dom.iterable` y `scripthost`
- `allowJs` y `checkJs`: parámetros booleanos que permiten incluir en la transpilación los archivos de JavaScript, verificando al mismo tiempo (con el parámetro `checkJs`) los posibles errores de sintaxis que puedan encontrarse.
- `sourceMap`: parámetro booleano que ayuda a la depuración de proyectos TypeScript. Genera un archivo con extensión `map` que entienden los navegadores web y ayuda a la depuración cuando se ejecuta el código (por ejemplo, desde la consola de Chrome)
- `removeComments`: parámetro booleano que permite eliminar o no los comentarios en los archivos JavaScript tras la compilación.
- `noEmit`: no realiza la transpilación. Puede ser útil en grandes proyectos para aprovechar las funcionalidades del tipado de TypeScript sin necesidad de estar continuamente consumiendo recursos con la transpilación.
- `noEmitError`: parámetro booleano que indica si los archivos transpilados serán o no generados si existe cualquier error en los archivos TypeScript. El valor por defecto es `false` (se generarán los archivos siempre a pesar de los errores).
- `strict`: parámetro booleano que permite o no realizar una comprobación estricta de los tipos. Si su valor es `true` (valor por defecto), entonces el resto de parámetros relacionados con la comprobación estricta son establecidos a `true` (excepto si explícitamente se establecen a `false`). Ver más adelante la sección de los parámetros de comprobación estricta en TypeScript.

```
/* tsconfig.json */
```

```
{  
  "compilerOptions": {  
    "target": "es5",
```

```

    "outDir": "dist",
    "rootDir": "src"
}
}

```

- ◆ Fuera de la clave *compilerOptions* también se pueden definir otras opciones
  - *exclude*: array de archivos/directorios que pueden excluirse de la transpilación, con la posibilidad de utilizar wildcards: \* (coincidencia de cero caracteres o más), ? (coincidencia de un carácter) y \*\*/ (coincidencia recursiva en cualquier directorio)
    - El directorio *node\_modules* es excluido por defecto si el parámetro *exclude* es omitido. En cambio, si es establecido, entonces es fundamental añadir el directorio *node\_modules*
  - *include*: array de archivos/directorios que son incluidos en la transpilación. Si se incluye este parámetro, entonces solamente será compilado el contenido de este array, menos lo establecido en el parámetro *exclude*.
  - *files*: array de archivos/directorios que son incluidos en la transpilación siempre, aunque estén incluidos dentro del parámetro *exclude*. Es un parámetro poco utilizado.

```
/* tsconfig.json (ejemplo de archivo de configuración) */
```

```
{
  "compilerOptions": {
    /* ... */
  },
  // no se transpilará el directorio node_modules, el archivo main1.ts, todos los archivos que cumplan *.dev.ts y todos los archivos que cumplan *.test.ts en cualquier directorio interno
  "exclude": [
    "node_modules",
    "main1.ts",
    "*.*.dev.ts",
    "**/*.*.test.ts",
  ]
}
```

```
/* tsconfig.json (ejemplo de archivo de configuración) */
```

```
{
  "compilerOptions": {
    /* ... */
  },
  // compila únicamente todo el contenido del directorio src (incluyendo directorios

```

```
internos)
  "include": ["src/**/*"],
}
```

- Un archivo *tsconfig.json* predefinido puede crearse automáticamente mediante el comando *tsc*.

```
tsc --init
```

- Y posteriormente es posible lanzar el watch de *tsc*. El *watch* vigilará también los subdirectorios internos.

```
tsc -w
```

- O simplemente transpilar (todos los archivos del directorio actual y subdirectorios internos) y finalizar.

```
tsc
```

## Depuración

- Ver parámetro *sourceMap* del archivo *tsconfig.json* para depurar con Chrome.
- También es útil la extensión *Debugger for Chrome* de Visual Studio Code, junto al parámetro *sourceMap* y los breakpoints en Visual Studio Code.

## Parámetros de comprobación estricta en TypeScript

- *noImplicitAny*: permite o no las variables con el tipo *any* implícito (aquellas que no establecen explícitamente el valor de *any*) en los parámetros declarados de una función (donde no se puede garantizar el tipo que se recibe).

```
// error si el parámetro noImplicitAny es true
function suma(a, b) {
  return a + b;
}
```

```
// correcto si el parámetro noImplicitAny es true
function suma(a: number, b: number) {
```

```
return a + b;  
}
```

- ◆ *strictNullChecks*: permite o no una comprobación de posibles valores nulos en las variables.

```
const button = document.querySelector('button');  
// error si strictNullChecks es true porque no se puede determinar si la variable button  
posee un valor o es null  
button.addEventListener('click', () => {  
    console.log('Click')  
});
```

```
const button = document.querySelector('button')!;  
// correcto si strictNullChecks es true porque se ha añadido anteriormente el operador de  
aserción no nulo (!)  
button.addEventListener('click', () => {  
    console.log('Click')  
});
```

```
const button = document.querySelector('button');  
// correcto si strictNullChecks es true porque se ha añadido una comprobación de la  
variable  
if (button) {  
    button.addEventListener('click', () => {  
        console.log('Click')  
    })  
}
```

- ◆ *strictBindCallApply*: las propiedades *bind*, *call* y *apply* de las funciones se encuentran fuertemente tipadas y son comprobadas.

```
const button = document.querySelector('button')!;  
  
function manejador(mensaje: string) {  
    console.log(mensaje);  
}  
  
// error si strictBindCallApply es true porque la función manejador espera un string  
button.addEventListener('click', manejador.bind(null));
```

```
const button = document.querySelector('button')!;

function manejador(mensaje: string) {
    console.log(mensaje);
}

// correcto si strictBindCallApply es true porque se le suministra un string a la función
manejador
button.addEventListener('click', manejador.bind(null, "hola"));
```

- ◆ *alwaysStrict*: garantiza que todos los archivos JavaScript transpilados sean de tipo estricto.

## Parámetros de calidad de código en TypeScript

- ◆ *noUnusedLocals*: notifica de error si existen variables locales que no son utilizadas. No tiene efecto sobre variables de script, dado que TypeScript no puede saber si estas variables serán utilizadas por otros scripts.

```
function sumar(a: number, b: number) {
    // error si noUnusedLocals es true porque la variable local no es utilizada
    const usuario = 'Alejandro';
    return a + b;
}
```

- ◆ *noUnusedParameters*: notifica de error si hay algún parámetro recibido por una función que no es utilizado.

```
// error si noUnusedParameters es true porque el parámetro c no es utilizado dentro de
la función
function sumar(a: number, b: number, c: number) {
    return a + b;
}
```

- ◆ *noImplicitReturns*: notifica de error si una función internamente posee un *return*, pero no devuelve valor para todas las rutas del código.

```
// error si noImplicitReturns es true porque sumar no se devuelve ningún valor cuando a
<= 0
function sumar(a: number, b: number) {
    if (a > 0) {
        return a + b;
    }
}
```

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas frontend

## (Angular)

Fundamentos I



# Fundamentos (I)

- ◆ Introducción
- ◆ Instalación y creación del primer proyecto
- ◆ Angular-CLI
- ◆ Archivos de un proyecto Angular
- ◆ Archivos de código de un proyecto Angular
- ◆ Funcionamiento general de Angular al renderizar una página
- ◆ Importar librerías externas
  - Método 1: utilizar un CDN
  - Método 2: utilizar el directorio assets
  - Método 3: agregar la librería al proyecto
- ◆ Snippets - Extensiones para Visual Studio Code

## Introducción

- ◆ Angular es una evolución de AngularJS. Incorpora mejoras importantes y sustituye a JavaScript por TypeScript como lenguaje de programación para construir las aplicaciones.
- ◆ Angular utiliza los denominados Componentes Web (Web Components) como tecnología para el desarrollo de aplicaciones web de tipo single-page (de página única).
- ◆ Web Componentes: es un estándar del W3C para la creación de contenedores web dedicados a una determinada funcionalidad. Los Componentes Web están compuestos por:
  - Archivos con el lenguaje de marcado HTML.
  - Hojas de estilo CSS para la presentación y diseño.
  - Código JavaScript para la lógica de negocio.
- ◆ Los Componentes Web se pueden ubicar en el código de las páginas HTML a través de una etiqueta específica, que renderizará el componente y su funcionalidad.

```
<nombre-del-componente></nombre-del-componente>
```

- ◆ [Buscador de Componentes Web](#).
- ◆ [Ejemplo de un Componente Web para Twitter](#).

## Instalación y creación del primer proyecto

- ◆ En primer lugar es necesario instalar [Node.js](#), que incluye también *npm* (gestor de paquetes de Node.js).

```
node -v  
npm -v
```

- El siguiente paso es instalar Angular CLI de forma global (con el parámetro `-g`). Los CLI (command line interface) son herramientas que ayudan a simplificar y automatizar ciertas tareas que de otra manera tendrían que realizarse de forma manual.
  - En este caso, Angular CLI proporciona herramientas para gestionar una aplicación desarrollada con Angular: crear un proyecto nuevo, generar componentes o servicios, agregar archivos nuevos y realizar una gran variedad de tareas relacionadas con el desarrollo, como pruebas, testing o despliegue.

```
npm install -g @angular/cli
```

- Comprobar la versión de Angular CLI.

```
ng --version
```

- Para actualizar Angular CLI.

```
npm uninstall -g angular-cli @angular/cli  
npm cache verify  
npm install -g @angular/cli
```

- Instalar otra versión de Angular CLI (la versión específica a instalar puede consultarse desde el [repositorio de npm de Angular CLI](#)). La documentación de Angular también ofrece un [manual para migrar](#) entre diferentes versiones.

```
npm uninstall -g angular-cli @angular/cli  
npm cache verify  
npm install -g @angular/cli@7.3.9
```

- Crear nuevo proyecto en Angular llamando `my-app` en el directorio actual.
  - No es necesario forzar la comprobación de tipos estricta (no recomendable cuando se está comenzando a aprender)
  - No es necesario añadir Angular routing.

```
ng new my-app
```

- Crear nuevo proyecto en Angular llamando my-app en un directorio específico.

```
ng new my-app --directory c:\proyecto_angular
```

- Acceder al directorio creado y arrancar Angular.

```
cd my-app  
ng serve
```

- El comando ng serve inicia el servidor, monitoriza los archivos y reconstruye la aplicación a medida que se realizan cambios.
- Una aplicación Angular escucha en el puerto 4200 por defecto: <http://localhost:4200>
- Visual Studio reconoce actualmente los decoradores como una funcionalidad experimental sujeta a cambios y la resalta como un error. Para evitar esto, puede copiarse el archivo tsconfig.json en el directorio de trabajo que se importa en Visual Studio, que generalmente es el directorio src de un proyecto Angular).

```
{
  "compilerOptions": {
    "outDir": "./out-tsc/app",
    "experimentalDecorators": true, // solamente añadir esta línea
    "types": []
  },
}
```

- El directorio donde se encuentran los principales archivos de código de una aplicación Angular es my-app/src/app/

## Angular-CLI

- Todos los comandos comentados a continuación deben ser ejecutados desde el directorio donde se encuentra el proyecto de Angular (aquel que contiene el directorio node\_modules).
- Cambiar puerto

```
ng serve --port 80
```

- Abrir directamente el navegador.

```
ng serve --open
```

- Crear componentes, directivas, rutas, pipes y servicios.

```
# Componentes
```

```
ng generate component components/component1  
ng g c components/component2 # Similar al anterior
```

```
# Directivas
```

```
ng generate directive directives/directive1  
ng g d directives/directive1 # Similar al anterior
```

```
# Pipes
```

```
ng generate pipe pipes/pipe1  
ng g p pipes/pipe2 # Similar al anterior
```

```
# Servicios
```

```
ng generate service services/service1  
ng g s services/service2 # Similar al anterior
```

- Otras opciones: no crear archivos de estilos (-s), ni de pruebas (--skipTests).

```
ng g c components/component3 --skipTests -s
```

- No crear archivos de estilos (-s) y ubicar las plantilla en el propio archivo ts (--inlineTemplate).

```
ng g c components/component4 -s --inlineTemplate=true
```

- No crear archivos de estilos (-s), ni de pruebas (--skipTests) y ubicar la plantilla en el propio archivo ts (--inlineTemplate). Por tanto, únicamente crea un archivo .ts

```
ng g c components/component5 -s --inlineTemplate=true --skipTests
```

- Más comandos

Ejercicio: crear un proyecto Angular con varios componentes, directivas, servicios y pipes en sus respectivos directorios.

## Archivos de un proyecto Angular

- ◆ /e2e: describen un tipo de testing llamado End to End, que se ejecutan generalmente con Jasmine. El objetivo de realizar pruebas End to End es identificar las dependencias del sistema y garantizar que la información correcta se transmite entre los distintos componentes.
- ◆ /node\_modules: directorio donde se encuentran todas las dependencias npm del proyecto.
- ◆ /src: directorio donde se encuentra todo el código fuente de la aplicación Angular.
- ◆ .editorconfig: configuración simple para el editor de texto que se utiliza. La mayoría de los editores admiten este tipo de configuraciones.
- ◆ .gitignore: archivos que no indexará Git.
- ◆ angular.json: configuración del CLI de Angular, incluido opciones para compilar y servir una aplicación Angular. Aquí se puede referenciar a librerías externas en la sección de styles o scripts.
- ◆ browserslist: permite centralizar la configuración de diferentes herramientas de frontend como Autoprefixer (adición de prefijos CSS para distintos navegadores), Stylelint (Lint para CSS) y Babel para distintas versiones de navegadores web.
- ◆ karma.conf.js: archivo de configuración de tests unitarios para el módulo Karma.
- ◆ package.json: configuración que describe las dependencias que requiere Angular. Las dependencias normales se guardan con --save, mientras que para las dependencias de desarrolladores se utiliza --save-dev.
- ◆ package-lock.json: archivo que se genera automáticamente para cualquier operación en la que npm modifique el árbol de node\_modules o package.json. Describe el árbol exacto que se generó, de modo que las instalaciones posteriores puedan generar árboles idénticos, independientemente de las actualizaciones de dependencia intermedias.
- ◆ README.md: es el archivo de texto plano típico de cualquier proyecto de Github o Gitlab en formato Markdown. Inicialmente ofrece información sobre algunos comandos de Angular.
- ◆ tsconfig{app|spec}.json:archivos de configuración de TypeScript para el proyecto y para los tests. Pueden copiarse al directorio src.
- ◆ tslint.json: Es la configuración del Lint para typescript. Lint es una herramienta de programación utilizada para detectar código sospechoso, confuso o incompatible entre distintas arquitecturas, es decir, errores de programación que escapan al habitual análisis sintáctico que realiza el transpilador de TypeScript.

## Archivos de código de un proyecto Angular

- ◆ /src/favicon.icon: icono de la aplicación.

- ◆ /src/index.html: es el HTML principal que es cargado al inicio. La mayoría de las veces no será necesario modificarlo. La CLI de Angular agrega automáticamente todos los archivos de JavaScript y CSS al compilar la aplicación para que nunca sea necesario agregar manualmente ninguna etiqueta script o link.
- ◆ /src/main.ts: es el punto de entrada a la aplicación. Arranca el módulo de arranque ( AppModule) de la aplicación.
- ◆ /src/polyfills.ts: asegura compatibilidad con todos los navegadores modernos.
- ◆ /src/styles.css: son los estilos generales del proyecto.
- ◆ /src/test.ts: es el punto de entrada para los tests unitarios.
- ◆ /src/app: en esta carpeta se encuentran los componentes, servicios y todos aquellos elementos importantes de una aplicación Angular.
- ◆ /src/app.component.{ts,html,css,spec.ts}: define el componente principal de la aplicación ( AppComponent). Incluye una plantilla HTML, un estilo CSS y un archivo para test. Es el componente raíz que se convertirá en un árbol de componentes anidados a medida que la aplicación evolucione.
- ◆ /src/app.module.ts: define el módulo principal de la aplicación ( AppModule), que es el módulo raíz que indica a Angular cómo ensamblar la aplicación. Inicialmente únicamente contiene un componente (el AppComponent).
- ◆ /src/assets: en este directorio se encuentran aquellos archivos estáticos que no son propios de un componente en Angular (imágenes, videos o archivos de cualquier tipo).
- ◆ /src/environments: este directorio contiene un archivo para cada uno de los posibles entornos de producción donde se desplegará la aplicación. Contiene variables de configuración simples.

Ejercicio: seguir el [tutorial del Tour de héroes](#) de la documentación oficial de Angular.

## Funcionamiento general de Angular al renderizar una página

1. Carga de la página src/index.html
2. Angular inyecta varios scripts de JavaScript en el archivo /src/index.html anterior.
  - Estos scripts ejecutan el código JavaScript asociado al archivo de TypeScript /src/main.ts
3. En el archivo /src/main.ts se incluye el método *bootstrapModule*, que inicializa la aplicación de Angular a través del módulo principal (llamado *AppModule* en una aplicación Angular recién creada).
  - Un módulo es un contenedor de elementos de programación en una aplicación Angular.

- ❖ Los elementos más importantes de un módulo son los componentes.

4. La información del módulo *AppModule* se encuentra por defecto en el archivo */src/app/module.ts*

- ❖ En este archivo se declara toda la configuración del módulo *AppModule*.
- ❖ Los componentes constituyentes del módulo se encuentran definidos en el array *declarations*
- ❖ Entre estos componentes se encuentra, por defecto, únicamente el componente llamado *AppComponent*, que además es el componente que se ejecuta inicialmente al arrancar la aplicación (ver el array de la propiedad *bootstrap*).

5. El componente *AppComponent* está compuesto por defecto por cuatro archivos:

*/src/app/app.component.css* (código CSS), */src/app/app.component.html* (código HTML), */src/app/app.component.spec.ts* (código para tests) y */src/app/app.component.ts* (código TypeScript).

- ❖ El archivo más importante de todos es el *app.component.ts*. En este archivo se define la configuración del componente *AppComponent*.

6. En el archivo */src/app/app.component.ts* se encuentra el selector del componente *AppComponent* (*app-root*, por defecto).

- ❖ Este selector es reconocido por Angular en el archivo */src/index.html* y en ese punto del archivo coloca el código HTML asociado al *AppComponent* (archivo */src/app/app.component.html*).
- ❖ Los selectores de otros componentes del módulo deben ubicarse en el archivo HTML del componente *\*AppComponent*.
- ❖ El único selector que puede ubicarse por defecto en el archivo */src/index.html* es el *app-root*, porque su componente asociado (*AppComponent*) es el que se encuentra establecido en el array *bootstrap* del módulo *AppModule*.

Ejercicio: seguir el [tutorial de la primera App con Angular](#) de la documentación oficial de Angular, utilizando StackBlitz.

## Importar librerías externas

### Método 1: utilizar un CDN

- ❖ Abrir el fichero *src/index.html* y colocar todas las librerías dentro de la etiqueta head.

```
<link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
```

### Método 2: utilizar el directorio assets

- ❖ Copiar la librería dentro del directorio *src/assets* y luego referenciarla desde el archivo *angular.json*.

```
...
"scripts": [
  "./assets/vendor/chart.js/Chart.min.js"
]
```

### Método 3: agregar la librería al proyecto

- Instalar el módulo npm.

```
npm install jquery --save
```

- Abrir el archivo angular.json y referenciar la librería (ubicada dentro del directorio node\_modules).

```
"scripts": [
  "./node_modules/jquery/dist/jquery.min.js"
]
```

- Parar el servidor de Angular y volver a ejecutar (ng serve).
- Importar la librería desde un componente.

```
import - as $ from 'jquery';
```

- Utilizar la librería desde el componente.

```
const div = $('div');
console.log(div);
```

Ejercicio 1 del proyecto: añadir Boostrap y un proxy a Angular para la comunicación con el servicio web.

Ejercicio 2 del proyecto: crear esquelo de la aplicación.

Ejercicio 3 del proyecto: crear el componente header dentro de components/headers

## Snippets - Extensiones para Visual Studio Code

- Angular 8 Snippets (Michael Morlund):

ng-: Angular Snippets  
fx-: Angular Flex Layout Snippets  
ngrx-: Angular NgRx Snippets  
ngxs-: Angular Ngxs Snippets  
m-: Angular Material Design Snippets  
rx-: RxJS Snippets for both TypeScript and JavaScript  
sw-: Service Workers Snippets  
t-: Test Snippets  
e-: Test Expect Snippets  
pwa-: Progressive Web Applications Snippets

- ◆ Angular v8 Snippets (John Papa):

a-component: component  
a-component-inline: component with inline template  
a-component-root: root app component  
a-directive: directive  
a-guard-can-activate: CanActivate guard  
a-guard-can-activate-child: CanActivateChild guard  
a-guard-can-deactivate: CanDeactivate guard  
g. uard-can-load: CanLoad guard  
h. ttp-get: http.get with Rx Observable  
a-httpclient-get: httpClient.get with Rx Observable  
a-http-interceptor: Empty Angular HttpInterceptor for HttpClient  
a-http-interceptor-headers: Angular HttpInterceptor that sets headers for HttpClient  
a-http-interceptor-logging: Angular HttpInterceptor that logs traffic for HttpClient

- ◆ Angular 8 and TypeScript/HTML VS Code Snippets (Dan Wahlin):

ag-AppModule: Create the root app module (@NgModule) snippet  
ag-AppFeatureModule: Angular app feature module (@NgModule) snippet  
ag-AppFeatureRoutingModule: Angular app feature routing module (@NgModule) snippet  
ag-CanActivateRoutingModuleGuard: Create a CanActivate routing guard snippet  
ag-CanDeactivateRoutingModuleGuard: Create a CanDeactivate routing guard snippet  
ag-Component: Component snippet  
ag-HttpClientService: Service with HttpClient snippet  
ag-InputProperty: @Input property snippet  
ag-InputGetSet: @Input property with get/set  
ag-OutputEvent: @Output event snippet  
ag-Pipe: Pipe snippet  
ag-Routes: Angular routes snippet  
ag-Route: Route definition snippet  
ag-Service: Service snippet  
ag-Subscribe: Observable subscribe snippet  
ag-Subscription: RxJS Subscription property  
ag-ConcatMap: ConcatMap snippet used to handle multiple observables returned from a service (Http calls or others)

ag-ClassBinding: [class] binding snippet  
ag-NgClass: [ngClass] snippet  
ag-NgFor: \*ngFor snippet  
ag-NgForm: ngForm snippet  
ag-NgIf: \*ngIf snippet  
ag-NgModel: [(ngModel)] binding snippet  
ag-RouterLink: Basic routerLink snippet  
ag-RouterLinkWithParameter: [routerLink] with route parameter snippet  
ag-NgSwitch: [ngSwitch] snippet  
ag-NgStyle: [ngStyle] snippet  
ag-Select: select control using \*ngFor snippet  
ag-StyleBinding: [style] binding snippet

Tu carrera digital ~

# Módulo 2

# Diseño de páginas interactivas frontend

## (Angular)

Fundamentos II

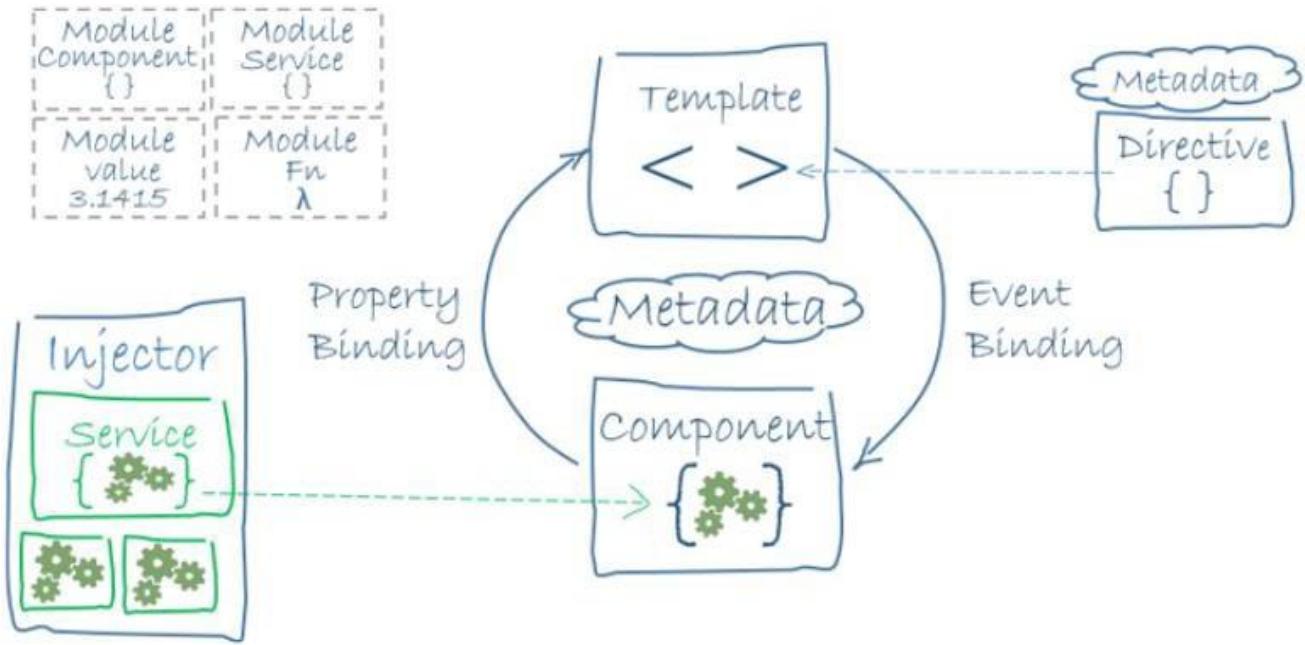


# Fundamentos (II)

- ◆ [Introducción](#)
- ◆ [NgModules o módulos](#)
- ◆ [Componentes](#)
  - [Plantillas](#)
  - [Data binding](#)
  - [Pipes](#)
  - [Directivas](#)
  - [Servicios e inyección de dependencias](#)
  - [Routing](#)
- ◆ [Clases](#)
- ◆ [Depurando una aplicación Angular con Visual Studio Code](#)

## Introducción

- ◆ Angular es una plataforma para crear aplicaciones frontend utilizando HTML, CSS y TypeScript.
- ◆ Angular está programado en TypeScript e implementa un conjunto de librerías que pueden importarse en las aplicaciones construidas.
- ◆ Los bloques básicos de construcción en una aplicación Angular son los NgModules o simplemente módulos (en la documentación oficial de Angular aparece como NgModule para diferenciarlos de los módulos clásicos de JavaScript).
- ◆ Una aplicación Angular siempre posee al menos un módulo raíz (por defecto, AppModule), que es iniciado al arranque.
- ◆ Los NgModule engloban principalmente a una serie de componentes (aunque también comprenden otro tipo de elementos como directivas, pipes, servicios, ...), generalmente agrupados de forma jerárquica.



- Los componentes controlan a los elementos gráficos y de interacción que se muestran en pantalla.
- Los componentes utilizan servicios, que proporcionan una funcionalidad específica no relacionada con la información que se muestra por pantalla.
- Los servicios pueden inyectarse en los componentes como dependencias, haciendo que el código sea más modular, reutilizable y eficiente.
- Todos los elementos importantes de Angular (módulos, componentes, directivas o servicios) son clases de TypeScript con decoradores especiales que marcan su tipo y proporcionan metadatos que indican a Angular cómo usarlos.

## NgModules o módulos

- Los NgModules son los elementos básicos de una aplicación en Angular. Difieren de los módulos de JavaScript (ES2015/ES6) en que declaran un contexto de compilación para un conjunto de componentes específico, dedicados a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades relacionadas.
- Son llamados NgModules para no confundirlos con los módulos normales de JavaScript (ES2015/ES6).
- Los NgModules pueden importar funcionalidades de otros NgModules y permitir que las suyas sean exportadas y utilizadas por otros NgModules.
- Una aplicación Angular está constituida por un conjunto de NgModules. Sin embargo, las aplicaciones pequeñas habitualmente sólo constan de un único módulo.
- Al crear una aplicación Angular mediante el comando `ng`, se genera también un único módulo: el módulo raíz, denominado habitualmente AppModule (cuyo código se encuentra

en el archivo app.module.ts del directorio src).

- El AppModule es el módulo raíz que provee el código necesario para arrancar la aplicación. Es referenciado en el archivo src/main.ts, que es el archivo principal de una aplicación Angular.

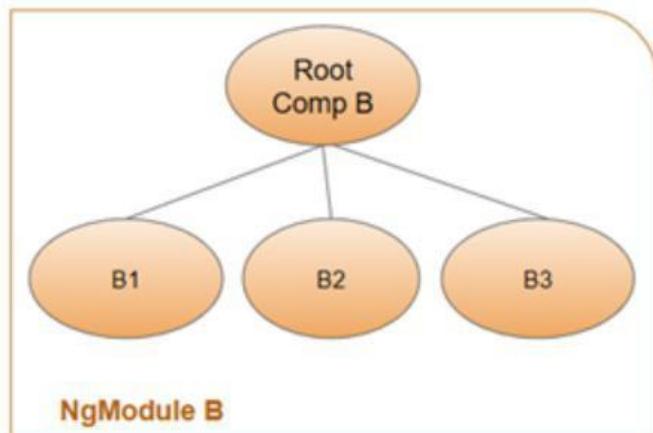
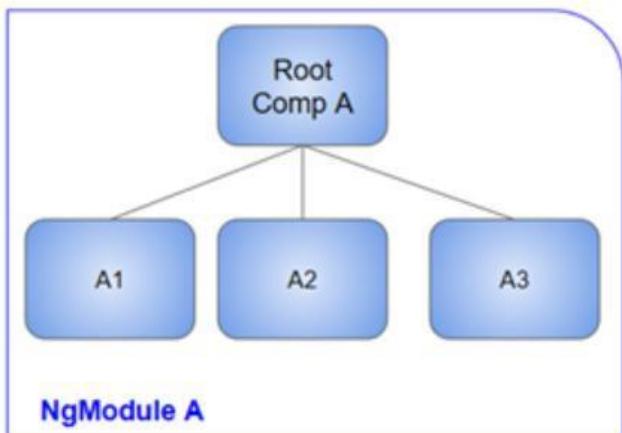
```
// src/main.ts
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

- Un NgModule está constituido por un conjunto de componentes, directivas, pipes y servicios, representando un dominio de aplicación, un flujo de trabajo o un conjunto de funcionalidades específicas que realiza la página web. Un NgModule siempre tiene al menos un componente (el componente raíz).



```
// app.module.ts
```

```
// Todos los módulos, componentes, directivas o pipes que se utilicen deben importarse aquí
import { BrowserModule } from '@angular/platform-browser'; // módulo de Angular
import { AppComponent } from './app.component'; // componente de la aplicación
import { HeroesComponent } from './heroes/heroes.component'; // componente de la aplicación

// @NgModule es un decorador que recibe un objeto de metadatos que definen al
```

módulo

```
@NgModule({}
```

```
// componentes, directivas o pipes que constituyen el módulo
```

```
declarations: [
```

```
  AppComponent,
```

```
  HeroesComponent
```

```
],
```

```
// importaciones de otros módulos que serán utilizados dentro del módulo actual
```

```
imports: [
```

```
  BrowserModule,
```

```
  FormsModule
```

```
],
```

```
// servicios que se importan y que pueden ser inyectados por todos los componentes, directivas o pipes que constituyen el módulo. También se pueden inyectar servicios a nivel de componente (más eficiente)
```

```
providers: [ Logger ],
```

```
// componente raíz
```

```
bootstrap: [AppComponent]
```

```
// componentes, directivas o pipes que estarán disponibles para ser usados en otros módulos. En la práctica, no se utiliza si la aplicación únicamente tiene un único módulo.
```

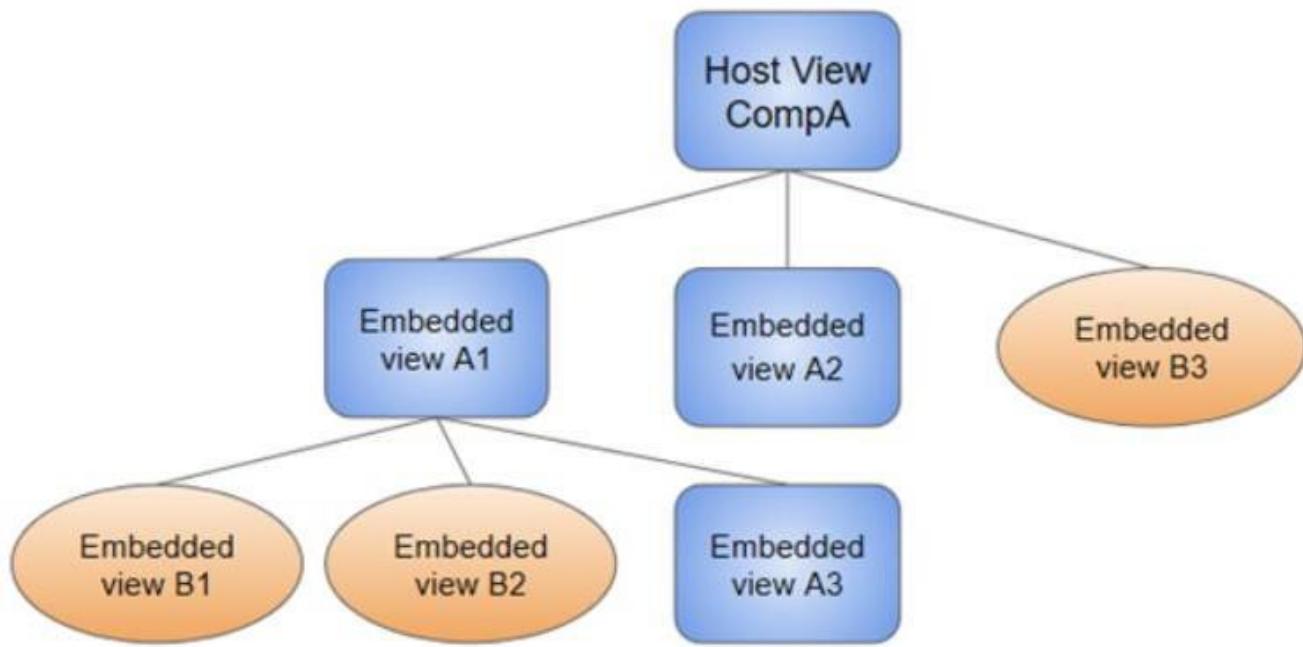
```
// exports: [ AppComponent ]
```

```
)
```

```
// exporta el módulo para que sea visible y utilizable por otros módulos
```

```
export class AppModule { }
```

- ◆ Al igual que los módulos de JavaScript, los NgModules pueden importar diferentes elementos (como los componentes) desde otros NgModules y también exportar los suyos propios.
- ◆ Un componente define una vista. Una vista puede incluir otras vistas que son definidas en otros componentes que pertenecen al mismo NgModule o a otro NgModule distinto.
  - A la vista raíz de esta jerarquía se denomina host view.
  - Al resto de vistas que cuelgan de la vista raíz se les denominadas vistas embedded (embebidas).



- Angular proporciona NgModules ya construidos y que pueden ser importados desde cualquier otro NgModule.

```

// carga de los NgModules
import { Component } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

// importación de los NgModules
@NgModule({
  imports: [ BrowserModule ]
  ...
})
  
```

- Creación de un módulo:

```

ng generate module module1
ng g m module2 # Otra forma
  
```

## Componentes

- Los componentes definen las vistas, que son conjuntos de elementos de pantalla (HTML y CSS) que Angular puede modificar según la lógica y los datos (TypeScript).
- Por tanto, los componentes están constituidos por tres partes: una plantilla (HTML), un estilo (CSS) y una lógica (TypeScript).

- Cada aplicación posee al menos un componente raíz, habitualmente llamado AppComponent (que se encuentra dentro del módulo raíz, habitualmente llamado AppModule).
- Los componentes son simplemente clases de TypeScript con un decorador (@Component) que define su tipo y proporciona metadatos que indican a Angular cómo deben ser usados.

```
// importación de la librería de Angular para poder utilizar el decorador @Component
import { Component } from '@angular/core';

@Component({
    // selector del componente (referenciado en el archivo src/index.html)
    // un selector puede colocarse en cualquier otra plantilla HTML y cargar en ese punto el
    // componente asociado completo
    selector: 'app-root',
    // plantilla del componente. En este archivo no están permitidas las etiquetas <script>
    // por cuestiones de seguridad
    templateUrl: './app.component.html',
    // estilos del componente
    styleUrls: ['./app.component.css'],
    // servicios que se importan y que pueden ser inyectados en el componente a través de
    // su constructor
    // La diferencia entre importar un servicio aquí o en el módulo es que si el servicio es
    // importado en el componente, se crearía una nueva instancia del servicio cuando se
    // genere una nueva instancia del componente. Esto no sucede si se importa desde el
    // módulo
    providers: [ HeroService ]
})

export class AppComponent {
    title = 'Tour of Heroes'; // propiedad de la clase
}
```

- La plantilla del componente combina HTML con directivas, pipes, selectores de otros componentes y mecanismos de data binding que permiten modificar el HTML antes de ser visualizado.
- La plantilla (HTML y CSS) también puede estar integrada dentro de los metadatos del decorador.

```
@Component({
    selector: 'app-root',
    template: `
        <h1>Tour of Heroes</h1>
```

```

    styles: ['h1 { font-weight: normal; }']
})
export class AppComponent { }

```

- Obligatoriamente debe existir la propiedad template o templateUrl en el decorador del componente (es realmente la única propiedad obligatoria). El selector puede omitirse y reubicar el componente utilizando rutas (ver sección de Rutas).
- Los componentes pueden utilizar los denominados servicios, que proporcionan una funcionalidad específica que no está directamente relacionada con la vista (HTML). Los servicios son habitualmente inyectados a través del constructor de la clase del componente.

```

export class AppComponent {

  title = 'Tour of Heroes'; // propiedad de la clase

  constructor(private http: HttpClient, private messageService: MessageService) {
    // ahora se puede utilizar httpClient y messageService como propiedades de la clase a
    // través del objeto this
  }

  private log(message: string) {
    this.messageService.add(`HeroService: ${message}`);
  }
}

```

- Angular crea, actualiza y destruye componentes mientras el usuario navega por la aplicación. Puede capturarse cada etapa del ciclo de vida de un componente por medio de los hooks. El hook más importante es el ngOnInit (invocado al crearse el componente).

```

export class AppComponent implements OnInit, OnDestroy {

  // el constructor de la clase suele estar vacío y únicamente suele utilizarse para
  // injectar dependencias

  constructor(private logger: LoggerService) {}

  // ngOnInit actúa como una especie de constructor (es invocado después del
  // constructor de la clase)
  ngOnInit() {
    this.logIt('onInit');
  }

  ngOnDestroy() {

```

```
this.logIt(`onDestroy`);  
}  
  
private logIt(msg: string) {  
  this.logger.log(`Spy ${msg}`);  
}  
}
```

- ◆ La inicialización de propiedades de un componente también puede realizarse en el constructor del mismo.

```
export class AppCtorComponent {  
  
  title: string;  
  myHero: string;  
  
  constructor() {  
    this.title = 'Tour of Heroes';  
    this.myHero = 'Windstorm';  
  }  
}
```

- ◆ Creación de un componente:

```
ng generate component component1  
ng g c component2 # Otra forma
```

- ◆ Por defecto, al crear un proyecto en Angular se crea un módulo (*AppModule*) en el archivo *app.module.ts* y un componente (*AppComponent*) en el archivo *app.component.ts*. El selector para este componente creado por defecto se ubica en el archivo *src/index.html*.

```
<!-- index.html -->  
  
<body>  
  <app-root></app-root>  
</body>
```

- ◆ El selector de un componente se puede utilizar en cualquier plantilla del resto de componentes del módulo.

```
<app-component1></app-component1>
```

- El selector habitualmente es una etiqueta HTML, pero también puede ser un atributo.

```
// app.component.ts utilizando el selector de atributo como selector del componente
@Component({
  selector: '[app-root]',
  template: 'Contenido del AppComponent',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-app';
  constructor() {
  }
}
```

```
<!-- index.html utilizando el selector de atributo -->
<body>
  <div app-root></div>
</body>
```

- O también como selector de clase.

```
// app.component.ts utilizando el selector de clase como selector del componente
@Component({
  selector: '.app-root',
  template: 'Contenido del AppComponent',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-app';
  constructor() {
  }
}
```

```
<!-- index.html utilizando el selector de clase -->
<body>
  <div class="app-root"></div>
</body>
```

- Sin embargo, el id y los pseudoselectores de CSS no pueden ser utilizados como selectores del componente.

- Una plantilla combina HTML con marcado especial de Angular (directivas, data binding, pipes y selectores de otros componentes) que modifica los elementos HTML antes de ser mostrados.
- Las directivas de las plantillas proveen lógica de programación, mientras que los data binding conecta los datos de la aplicación (en TypeScript) y el DOM. Finalmente, los pipes pueden realizar algún tipo de transformación sobre los valores que son mostrados (por ejemplo, convertir fechas a una zona horaria específica, cadenas de texto a minúsculas, conversiones monetarias, etc).

```
<!-- directiva *ngIf -->
<div *ngIf="hero">Existe</div>

<!-- otra directiva *ngFor -->
<li *ngFor="let hero of heroes" (click)="selectHero(hero)">
  {{hero.name}}
</li>

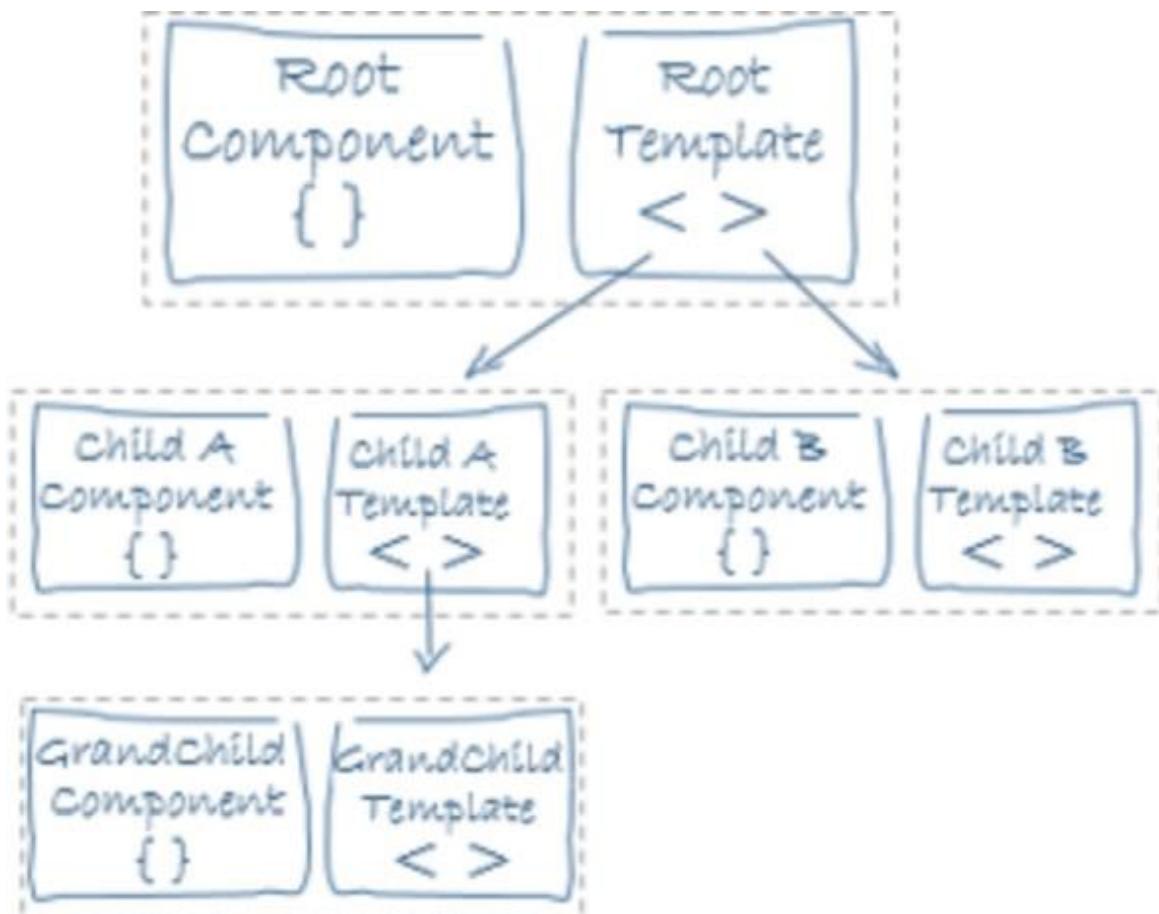
<!-- binding de tipo interpolación -->
<p>My current hero is {{currentHero.name}}</p>

<!-- pipe (denotado con el carácter |) -->
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>

<!-- selector de otro componente -->
<app-hero-detail></app-hero-detail>

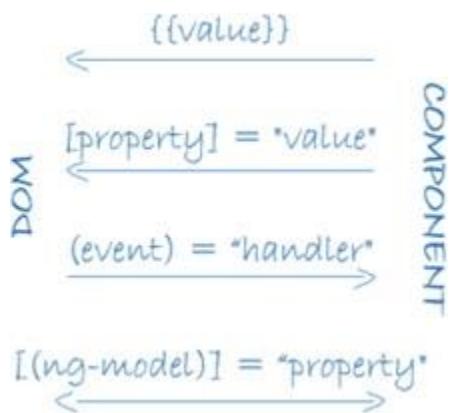
<!-- selector de otro componente combinado con la directiva *ngIf y un tipo de binding entre componentes -->
<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

- Un componente (código TypeScript) y una plantilla (HTML) definen una vista.
- Las vistas generalmente se organizan jerárquicamente, permitiendo modificar, mostrar y ocultar secciones específicas de la página a través del uso de selectores de otras vistas.
- Una jerarquía de vistas puede incluir vistas de distintos componentes en el mismo NgModule, pero también puede incluir vistas de componentes que están definidos en otros NgModules.



## Data binding

- Sin Angular, para introducir los valores de los datos en HTML y convertir las respuestas de los usuarios en acciones y actualizaciones sería necesario escribir lógica push/pull a mano, por ejemplo, utilizando jQuery o mediante los métodos del DOM en JavaScript.
- En Angular existen básicamente dos formas de pasar datos del código a la plantilla y viceversa:
  - One-way binding:
    - Desde el código a la plantilla: interpolación, property binding, attribute binding, class binding y style binding.
    - Desde la plantilla al código: event binding.
  - Two-way binding: en ambas direcciones (del código a la plantilla y viceversa).



<!-- interpolación: muestra en la plantilla el valor de la propiedad del componente (hero.name en este caso) -->

```
<li>{{hero.name}}</li>
```

<!-- property binding: pasa un valor de la plantilla (el valor de la propiedad disabled del botón en este caso) a la propiedad del componente (isUnchanged en este caso) -->

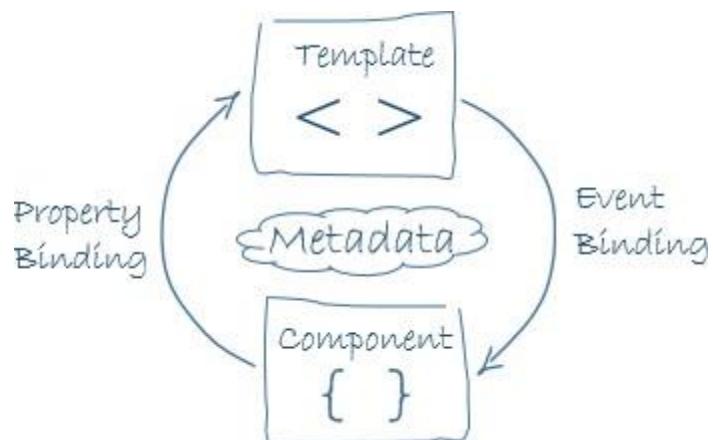
```
<button [disabled]="isUnchanged">Disabled Button</button>
```

<!-- event binding: ejecuta un determinado método del componente (selectedHero en este caso) cuando ocurre un evento concreto (click en este caso), generalmente disparado por una acción del usuario -->

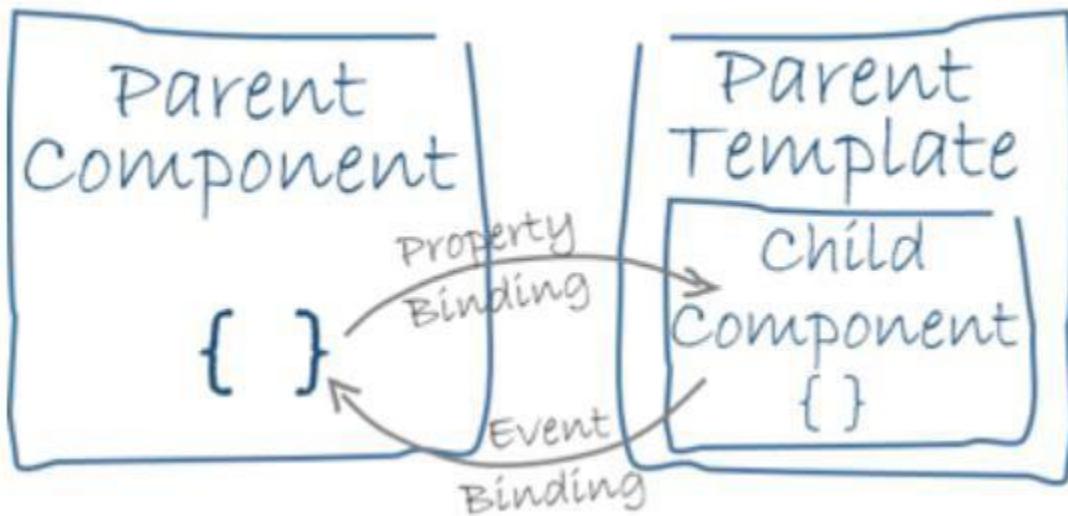
```
<li (click)="selectHero(hero)"></li>
```

<!-- two way data binding (utilizada principalmente en formularios): combina property binding y event binding en una única notación `[(ngModel)]` y la comunicación es bidireccional -->

```
<input [(ngModel)]="hero.name">
```



- Los data binding juegan un papel importante en la comunicación entre una plantilla y su componente, y también es importante dentro de una jerarquía de componentes (entre un componente y sus hijos).



```
<!-- property binding entre un componente padre y un componente hijo -->
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

## Pipes

- Los pipes permiten declarar transformaciones de valores en la plantilla HTML.
- Angular define varios pipes, como el pipe de fecha, de moneda, de strings y algunos más.
- Para establecer un pipe en una plantilla HTML es necesario utilizar el operador de tubería (|).

```
<!-- 'Jun 15, 2015'-->
<p>{{today | date}}</p>

<!-- 'Monday, June 15, 2015'-->
<p>{{today | date:'fullDate'}}</p>

<!-- '9:43 AM'-->
<p>{{today | date:'shortTime'}}</p>
```

- Los pipes pueden ser concatenados y pueden recibir parámetros.
- También es posible crear un pipe personalizado utilizando el decorador @Pipe en una clase de TypeScript.

## Directivas

- Las directivas alteran el flujo HTML de la página mediante un conjunto de instrucciones.
- Hay dos tipos de directivas: directivas estructurales y directivas de atributos.

- Las directivas estructurales alteran el diseño, añadiendo, eliminando o reemplazando elementos en el DOM.

```
<!-- directivas estructurales más importantes: *ngFor o *ngIf -->
<div *ngFor="let hero of heroes">
  <span>{{hero.name}}</span>
</div>

<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

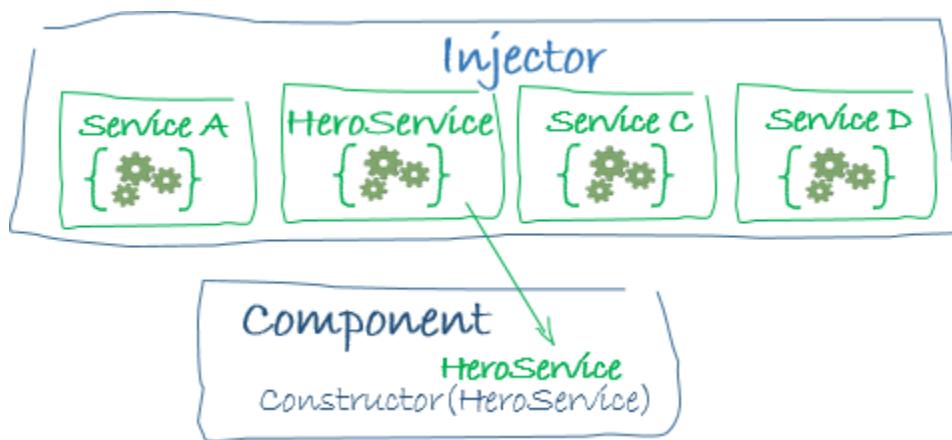
- Las directivas de atributo alteran la apariencia o el comportamiento de un elemento HTML. En las plantillas se muestran como atributos HTML normales (de ahí el nombre).

```
<!-- la directiva ngModel implementa el two-way data binding, que es un ejemplo de directiva de atributos. ngModel modifica el comportamiento de un elemento HTML (generalmente un <input>), configurando su valor de visualización y respondiendo a eventos -->
<input [(ngModel)]="hero.name">
```

- Angular provee otra directiva estructural (ngSwitch) y otras directivas de atributo (ngStyle y ngClass).
- También es posible crear directivas personalizadas mediante el decorador @Directive.

## Servicios e inyección de dependencias

- Los servicios contienen lógica de programación que no está asociada con una vista específica.
- Un servicio puede ser la validación de los datos introducidos por el usuario, el logging o la conexión con un servidor utilizando HTTP/s.
- En definitiva, los servicios en Angular se utilizan para tareas pesadas y liberar de carga a los componentes (que deben tener el menor código posible y nunca poseer instrucciones bloqueantes como la llamada a un servidor).
- Para crear un servicio de Angular se utiliza el decorador @Injectable.
- Durante la creación del servicio es necesario registrar el proveedor donde el servicio será inyectado.
  - Un inyector crea dependencias y mantiene un contenedor de instancias que reutiliza si es posible.
  - Un proveedor es un objeto que indica a un inyector cómo obtener o crear una dependencia.



- Cuando Angular descubre que un componente depende de un servicio, primero verifica si el injector tiene alguna instancia existente de ese servicio.
  - Si aún no existe una instancia del servicio solicitado, entonces el injector crea una utilizando el proveedor registrado y la agrega al injector.
  - Cuando todos los servicios solicitados por el componente se han resuelto, Angular puede invocar al constructor del componente con esos servicios como argumentos.
- El registro del proveedor puede realizarse de dos formas distintas (utilizar una, otra, o ambas a la vez tiene el mismo comportamiento):
  - Usar la propiedad providedIn del decorador @Injectable, asignando el valor 'root' (el servicio se injectará en toda la aplicación en todas partes se utilizará la misma instancia) o el módulo específico donde será injectado.
  - Asignar el servicio en el array providers del decorador de un NgModule o de un componente.

```
// primera forma (en el archivo del servicio)
import { Injectable } from '@angular/core';
```

```
// de forma predeterminada, el comando "ng generate service" de Angular registra un proveedor utilizando el proveedor raíz (root). Este proveedor es responsable de crear una instancia del servicio (invocando a su constructor) y hacer que esté disponible en toda la aplicación
```

```
@Injectable({
  providedIn: 'root',
})
export class HeroService {
  constructor() { }
  print() {
    console.log('Hello world');
  }
}
```

```
// segunda forma (en el archivo de un NgModule)

// importación del servicio
import { HeroService } from './services/hero.service';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  // inyección del servicio en el módulo AppModule (el servicio estará disponible en todos los elementos que constituyen el NgModule)
  providers: [ HeroService ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

```
// segunda forma (en el archivo de un componente)

// importación del servicio
import { HeroService } from './services/hero.service';

@Component({
  selector: 'app-hero-list',
  template: '',
  // inyección del servicio en el módulo AppComponent (el servicio estará disponible solamente en este componente)
  providers: [ HeroService ]
})
export class AppComponent { }
```

- A continuación puede inyectarse el servicio en el componente para poder ser utilizado. Cuando Angular descubre que un componente depende de un servicio (desde el constructor del componente), primero comprueba si el inyector ya posee una instancia de ese servicio. Si aún no existe, el inyector crea una y la inyecta en el componente.

```
import { Component, OnInit } from '@angular/core';

// importación del servicio
import { HeroService } from './services/hero.service';

@Component({
```

```

selector: 'app-root',
template: `

`)

export class AppComponent implements OnInit {

// inyección en el constructor del componente
constructor(private heroService: HeroService) {

}

ngOnInit() {
// uso del servicio desde el constructor
this.heroService.print();
}
}

```

- ◆ Algunas veces no es deseable que el servicio pueda ser inyectado desde cualquier parte de la aplicación. En este caso, la propiedad providedIn debe referenciar a la clase o conjunto de clases de tipo @NgModule o @NgComponent donde será inyectado.
- ◆ Cuando se registra un servicio desde un componente específico, se obtiene una nueva instancia del servicio con cada nueva instancia de dicho componente.

```

@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})

```

- ◆ Los servicios pueden depender de otros servicios.

```

@Injectable({
  providedIn: 'root',
})

export class HeroService {
  private heroes: Hero[] = [];

  constructor(private backend: BackendService, private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes);
    });
    return this.heroes;
  }
}

```

## Routing

- Angular provee un NgModule denominado Router para definir rutas de navegación entre las diferentes partes de una aplicación.
- El Router mapea las URL como rutas a las vistas (en lugar de a páginas específicas como tradicionalmente hacen las aplicaciones web normales).
- De esta forma, cuando el usuario realiza una acción (como hacer click sobre un determinado elemento), no se abre una nueva página en el navegador, sino que el Router intercepta esa acción y muestra o esconde una vista (componente + plantilla) o jerarquía de vistas.
- Las posibles eventos de navegación que pueden ser capturados por el Router de Angular son:
  - Ingresar una URL en la barra de direcciones.
  - Hacer clic en enlaces de la página.
  - Hacer clic en los botones de avance y retroceso del navegador.
- El Router registra la actividad en el historial del navegador para que los botones de avance y retroceso también funcionen. Esto no es posible en otros frameworks web como GWT, que siguen una filosofía similar a la de Angular.

## Clases

- Los datos manejados por la aplicación deberían tener una estructura definida por una clase. Para ello se utilizan las clases normales de TypeScript.

```
ng generate class hero
```

```
// por defecto, la clase está vacía
export class Hero {
}
```

- En el constructor de la clase pueden recibirse todos los valores que tendrá el objeto cuando sea instanciado. Estas propiedades estarán disponibles en todo el objeto utilizando this.

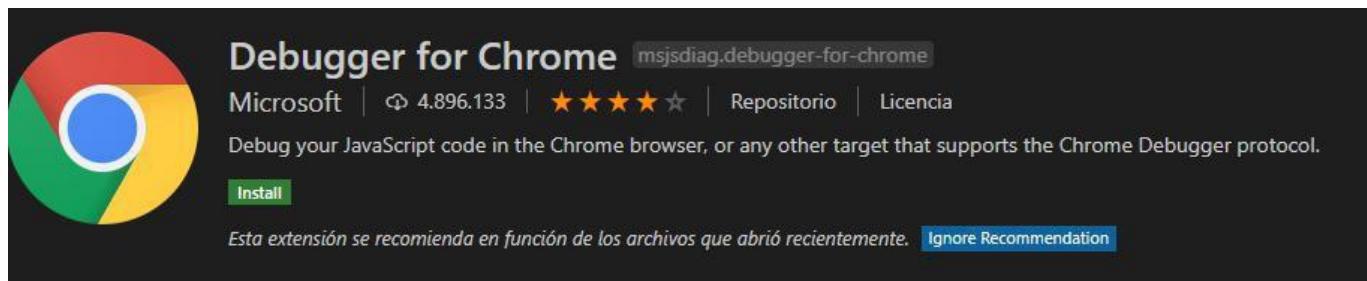
```
export class Hero {
  constructor(public id: number, public name: string) { }
}
```

- Y en otro componente distinto se puede utilizar la clase creada.

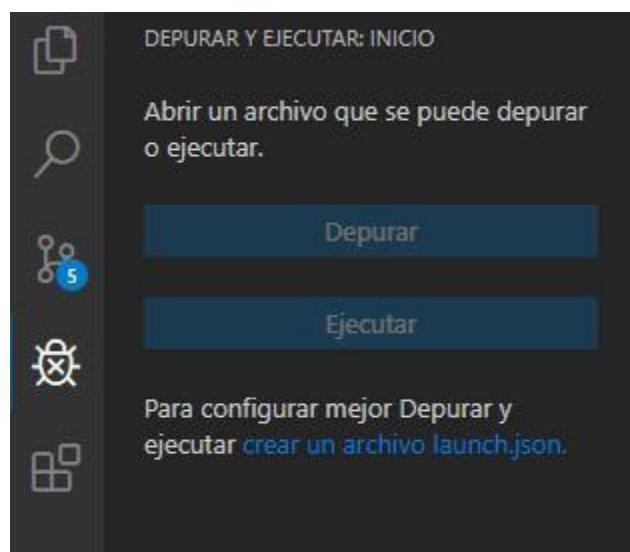
```
const heroes = [
  new Hero(1, 'Windstorm'),
  new Hero(13, 'Bombasto'),
  new Hero(15, 'Magneta'),
  new Hero(20, 'Tornado')
];
const myHero = this.heroes[0];
```

## Depurando una aplicación Angular con Visual Studio Code

- El primer paso para depurar una aplicación Angular con Visual Studio es instalar la extensión Debugger for Chrome.



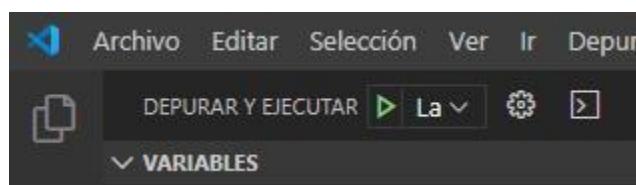
- El siguiente paso es pulsar sobre la pestaña de Bug (en la barra lateral izquierda de Visual Studio Code), crear un archivo launch.json y seleccionar el entorno de Chrome.



- Automáticamente se creará un nuevo archivo launch.json con datos por defecto.
  - Es necesario cambiar el puerto 8080 (en la propiedad url) al puerto 4200 (el utilizado por Angular).
  - La propiedad webRoot debe apuntar al directorio raíz del proyecto de Angular. Pueden utilizarse rutas absolutas.

```
{
    // Use IntelliSense para saber los atributos posibles.
    // Mantenga el puntero para ver las descripciones de los existentes atributos.
    // Para más información, visite: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "chrome",
            "request": "launch",
            "name": "Launch Chrome against localhost",
            "url": "http://localhost:4200",
            "webRoot": "${workspaceFolder}/.."
        }
    ]
}
```

- A continuación, se pulsa en el botón verde de Play (dentro de la pestaña Bug) ubicado en la parte superior izquierda de la ventana de Visual Studio Code. La aplicación de Angular debe estar ejecutándose con ng serve.

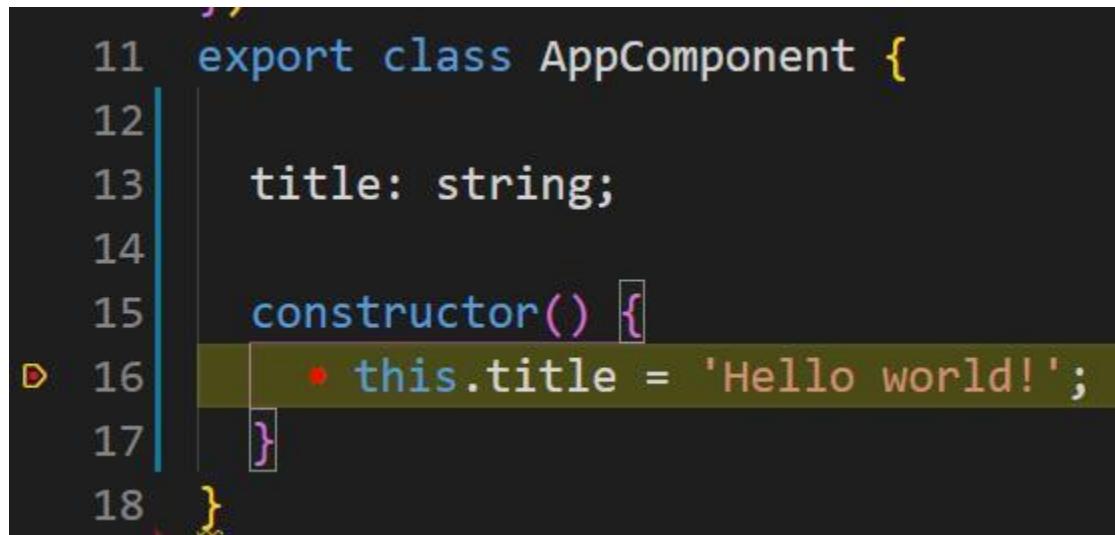


- Una ventana de Chrome se abrirá y ejecutará la aplicación de Angular accediendo al puerto 4200.
- Puede accederse a un archivo de TypeScript y marcar una línea del margen izquierda para añadir un punto de interrupción o checkpoint en el código.

```

11 ✓export class AppComponent {
12
13     title: string;
14
15 ✓   constructor() {
16     this.title = 'Hello world!';
17   }
18 }
```

- Los botones ubicados en la parte superior central permiten controlar la depuración. Al pulsar el botón de actualizar, se reinicia la página y el programa se detienen en el checkpoint.



```
11 export class AppComponent {  
12     title: string;  
13  
14     constructor() {  
D 16         this.title = 'Hello world!';  
17     }  
18 }
```

- Otra forma de depurar es con las *source maps* que genera Angular. Es posible crear *breakpoints* a nivel del navegador y automáticamente aparecerá el archivo de TypeScript o la plantilla HTML donde se pretende parar la ejecución de código.
- También aparecen los archivos de TypeScript, CSS y HTML directamente en la sección de *webpack://*, dentro de la pestaña *Sources* de las herramientas del desarrollador de Chrome.

Tu carrera digital ~

# Módulo 3

## Java básico

Introducción a Java



# Introducción a Java

---

- ◆ Introducción al lenguaje Java
- ◆ Características de Java
- ◆ Plataformas Java
- ◆ Software para Java
- ◆ Instalación del JDK y Eclipse
- ◆ Primer programa en Java

## Introducción al lenguaje Java

- ◆ Java es un lenguaje de programación creado por la empresa Sun Microsystems en 1990.
- ◆ Posee una sintaxis similar a los lenguajes C o C++. Surgió de la necesidad de crear software para la electrónica doméstica.
- ◆ El objetivo fue el de crear una lenguaje de programación para desarrollar programas muy pequeños, veloces, confiables y transportables.



## Características de Java

# Orientado a Objetos

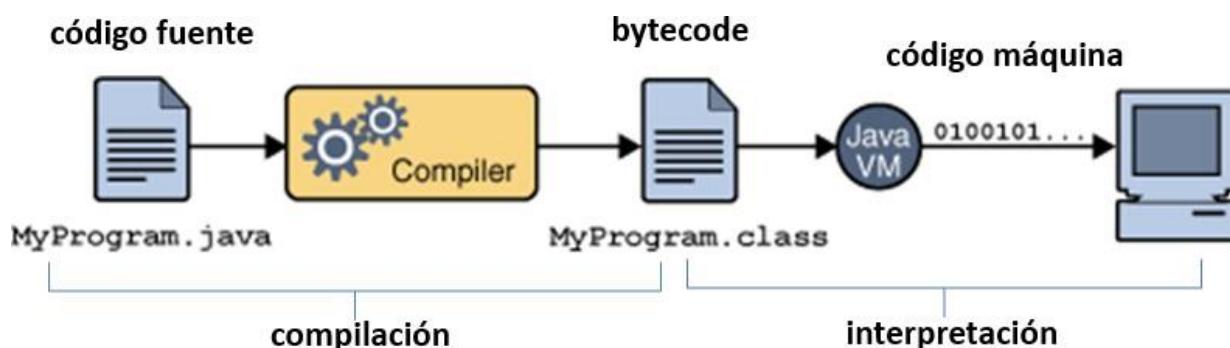
# Compilado

## Interpretado

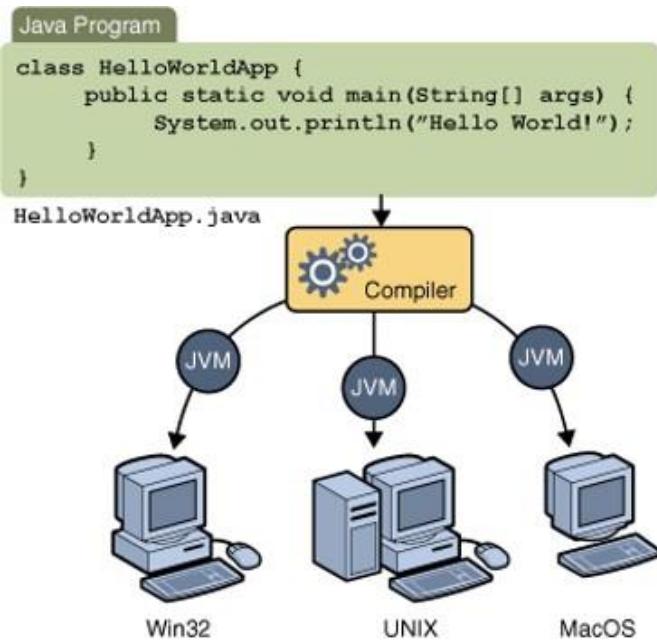
## Independiente de la plataforma

## Códigos de bytes

- Compilación e interpretación: es una traducción que transforma un lenguaje de programación (llamado código fuente) a lenguaje máquina (ceros y unos).
    - Antes de generar el código en lenguaje máquina, el compilador crea un código intermedio (llamado bytecode o código de bytes). El código bytecode suelen ser interpretado por un intérprete de bytecode denominado máquina virtual de Java (JVM).

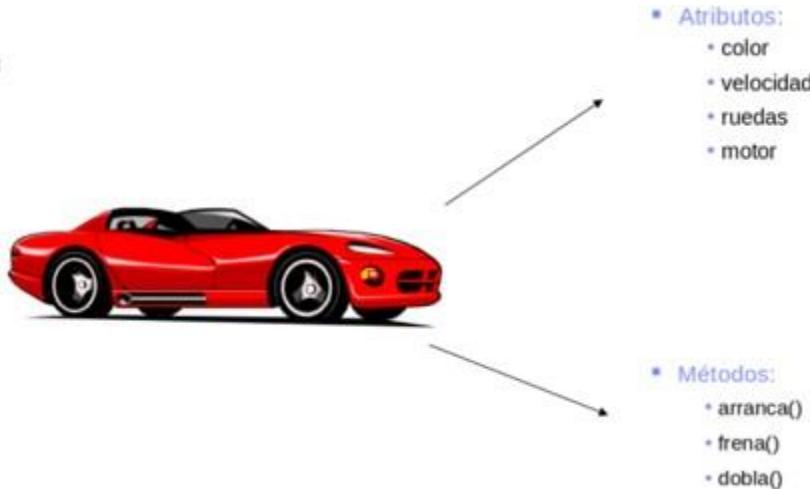


- Independiente de la plataforma: el código bytecode generado por el compilador puede ser interpretado en casi cualquier sistema operativo, dispositivo o arquitectura de computadores(JVM).



*La filosofía de Java es:  
“escribe una vez,  
ejecuta en cualquier  
lado” (WORA).*

- Orientado a objetos: es un paradigma de programación que organiza el código utilizando objetos, que son elementos individuales que contienen un estado (atributos) y un comportamiento (métodos). La programación orientada a objetos está basada en varias técnicas: herencia, abstracción, polimorfismo, encapsulamiento e interfaces.



- Otras ventajas significativas de Java:
  - Java es rápido: las últimas versiones de la JVM han mejorado sustancialmente la ejecución de las aplicaciones.
  - Java es seguro: gestiona automáticamente la memoria, reduce la posibilidad de vulnerabilidades y proporciona mecanismos avanzados para transmitir datos de forma cifrada.
  - Java posee un amplio conjunto de bibliotecas agrupadas en paquetes con diversas funcionalidades.
  - Java es utilizado en el desarrollo de aplicaciones móviles con Android SDK o en aplicaciones web con Servlets, Struts, JSPs o Spring.

## Plataformas Java

- Las plataformas Java definen el conjunto de tecnologías y el entorno necesario para ejecutar aplicaciones desarrolladas mediante el lenguaje Java.
- Existen varios tipos de plataformas Java, en función del tipo de sistema operativo y dispositivo donde se ejecutarán las aplicaciones:
  - Java SE (Standard Edition): para aplicaciones de escritorio y de propósito general.
  - Jakarta EE (Enterprise Edition): orientada a aplicaciones empresariales.
  - Java ME (Micro Edition): librerías para dispositivos con capacidades limitadas de almacenamiento, visualización y potencia. JavaFX: destinada al desarrollo de aplicaciones multimedia interactivas.
  - Java Card: permite ejecutar pequeñas aplicaciones Java en tarjetas inteligentes.



- A continuación se presentan las versiones de Java SE

|Versión de Java SE|Año| |Se inicia el proyecto del lenguaje Java|1991| |JDK 1.0|1996| |JDK 1.1|1997| |J2SE 1.2|1998| |J2SE 1.3|2000| |J2SE 1.4|2002| |J2SE 5.0|2004| |Java SE 6|2006| |Java SE 7|2011| |Java SE 8|2011| |Java SE 9|2014| |Java SE 10|2018| |Java SE 11 (LTS)|2018| |Java SE 12|2019| |Java SE 13|2019| |Java SE 14|2020| |Java SE 15|2020| |Java SE 16|2021| |Java SE 17 (LTS)|2021| |Java SE 18|2022|

- A continuación se presentan las versiones de Java EE / Jakarta EE

|Versión de Java EE / Jakarta EE|Año| |J2EE 1.2|1999| |J2EE 1.3|2001| |J2EE 1.4|2003| |Java EE 5|2006| |Java EE 6|2009| |Java EE 7|2013| |Java EE 8|2017| |Jakarta EE 8|2019| |Jakarta EE 9|2020| |Jakarta EE 9.1|2021|

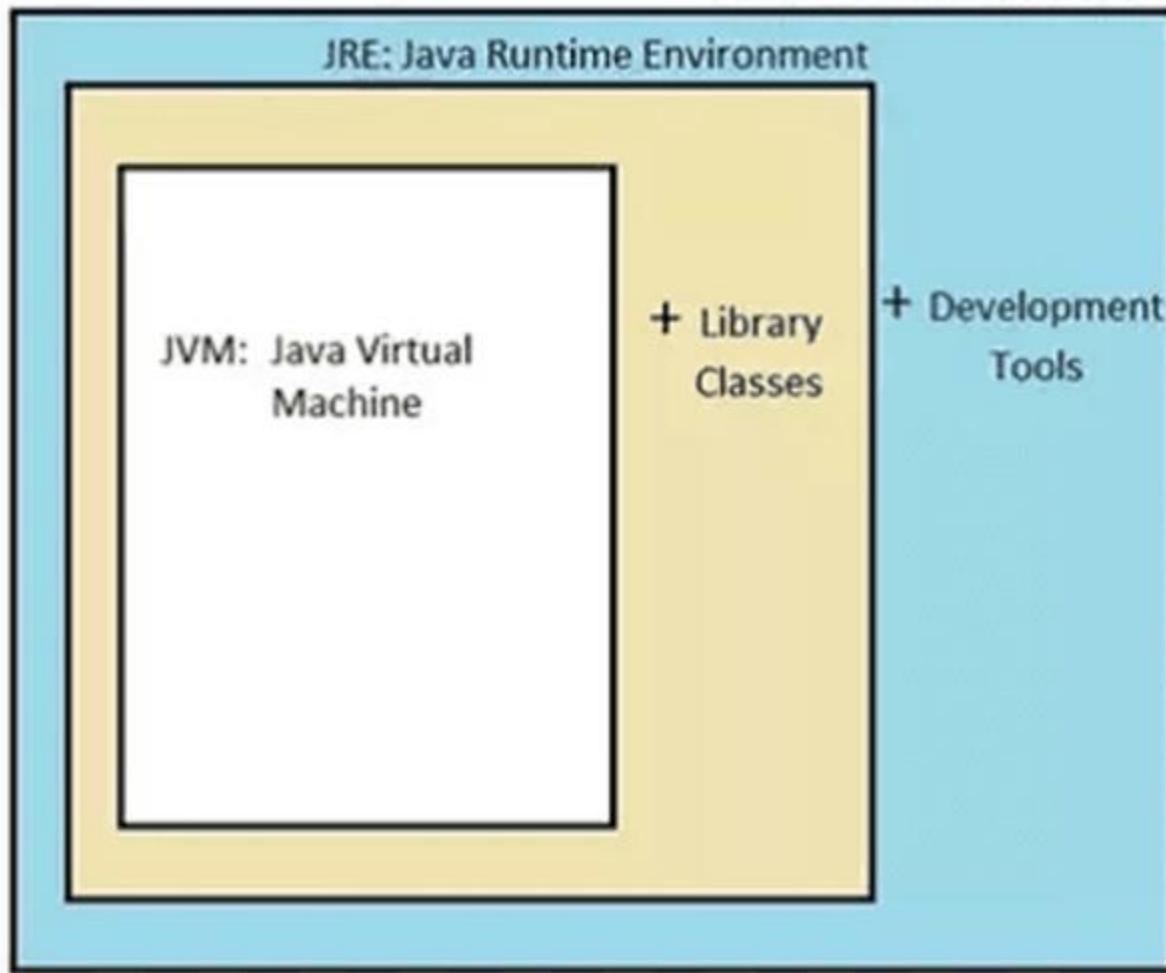
- Desde el 24 de abril de 2018, Java EE es gestionado por la Fundación Eclipse. La Fundación Eclipse fue forzada a eliminar la palabra Java del nombre debido a que Oracle posee el registro de la marca Java. El 26 de febrero de 2018 se anunció que el nuevo nombre de Java EE sería Jakarta EE (compatible con Java EE).

## Software para Java

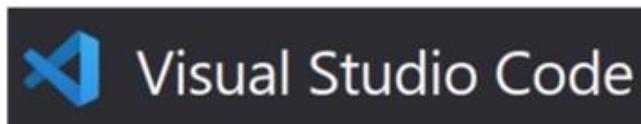
- El software de Java para la compilación y ejecución de aplicaciones está dividido en tres paquetes:
  - JVM (Java Virtual Machine): es una máquina virtual capaz de interpretar y ejecutar instrucciones expresadas en el bytecode Java.

- JRE (Java Runtime Environment): proporciona los requisitos mínimos para ejecutar una aplicación Java. Incluye la JVM, clases y archivos auxiliares.
- JDK (Java Development Kit): es un entorno de desarrollo software utilizado para desarrollar aplicaciones Java. Incluye el JRE, un compilador (javac) otras herramientas necesarias para el desarrollo de Java. Una alternativa es OpenJDK y el gestor sdkman.

### JDK: Java Development Kit

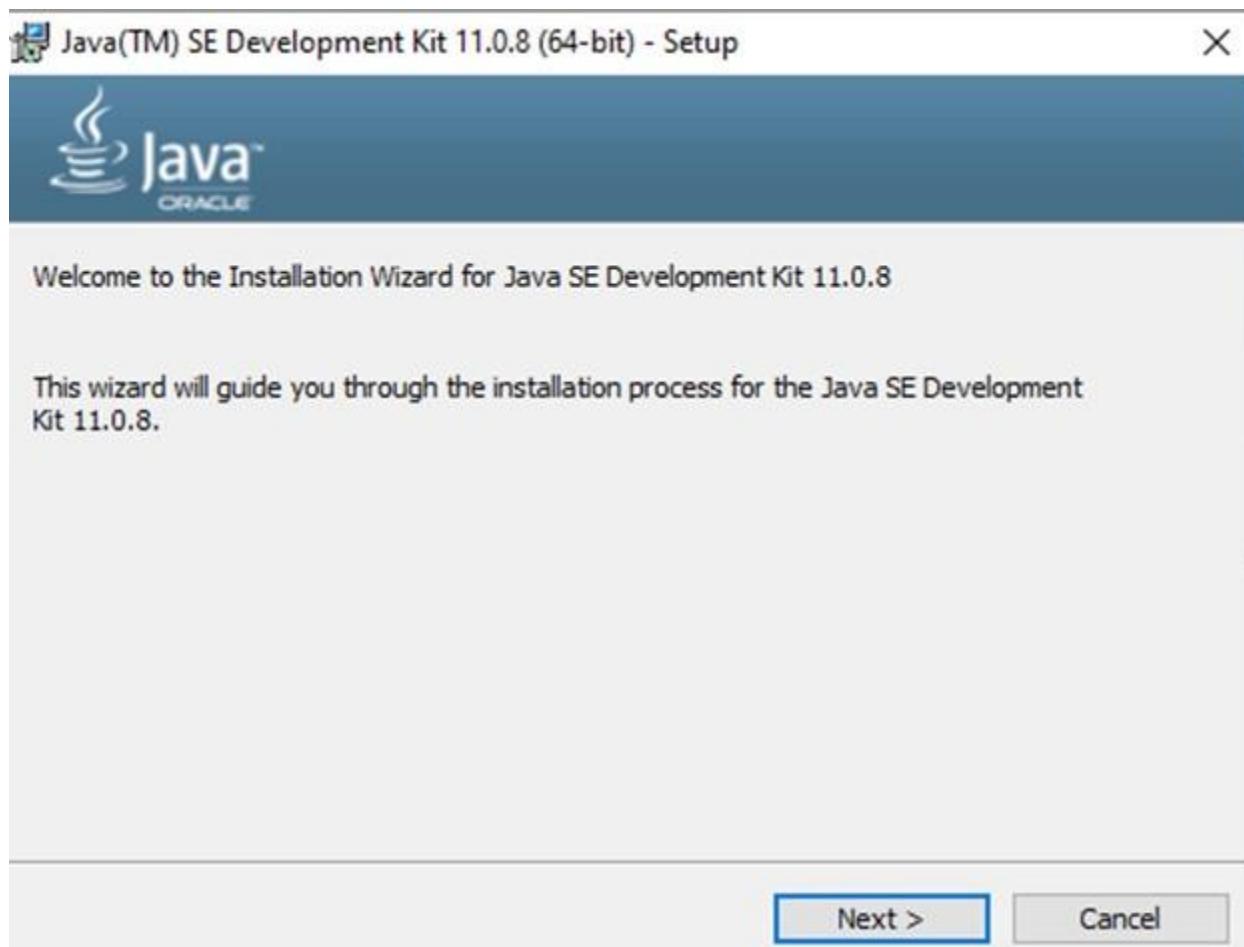


- ◆ Para escribir programas en Java existen diferentes entornos de desarrollo integrados (IDE).
- ◆ Quizás el más importante por su madurez y antigüedad es Eclipse, aunque IntelliJ también es una excelente opción.
- ◆ Eclipse forma parte de la Fundación Eclipse (propietaria de la marca Jakarta EE).

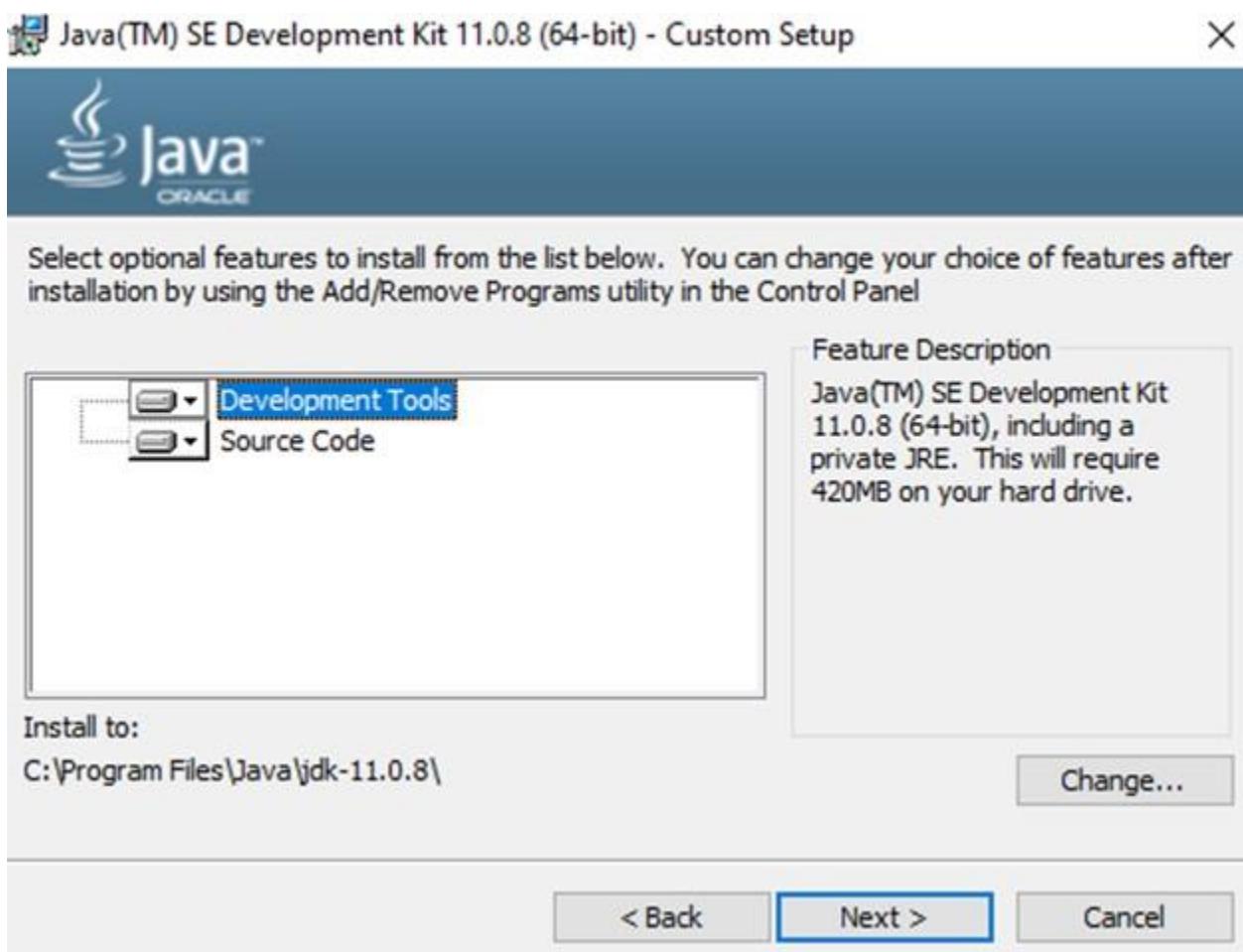


## Instalación del JDK y Eclipse

- En primer lugar es necesario acceder a la página web oficial del [JDK 11 de Java](#).
  - Puede utilizarse la cuenta [cursojavacore@gmail.com](mailto:cursojavacore@gmail.com) con contraseña cursojava12A para descargar el JDK (versión x64\_bin\_exe) para Windows (antes que hay que aceptar la licencia).
- Ejecutar el instalador del JDK (jdk-X\_windows-x64\_bin.exe).



- Instalar Development Tools y Source Code.



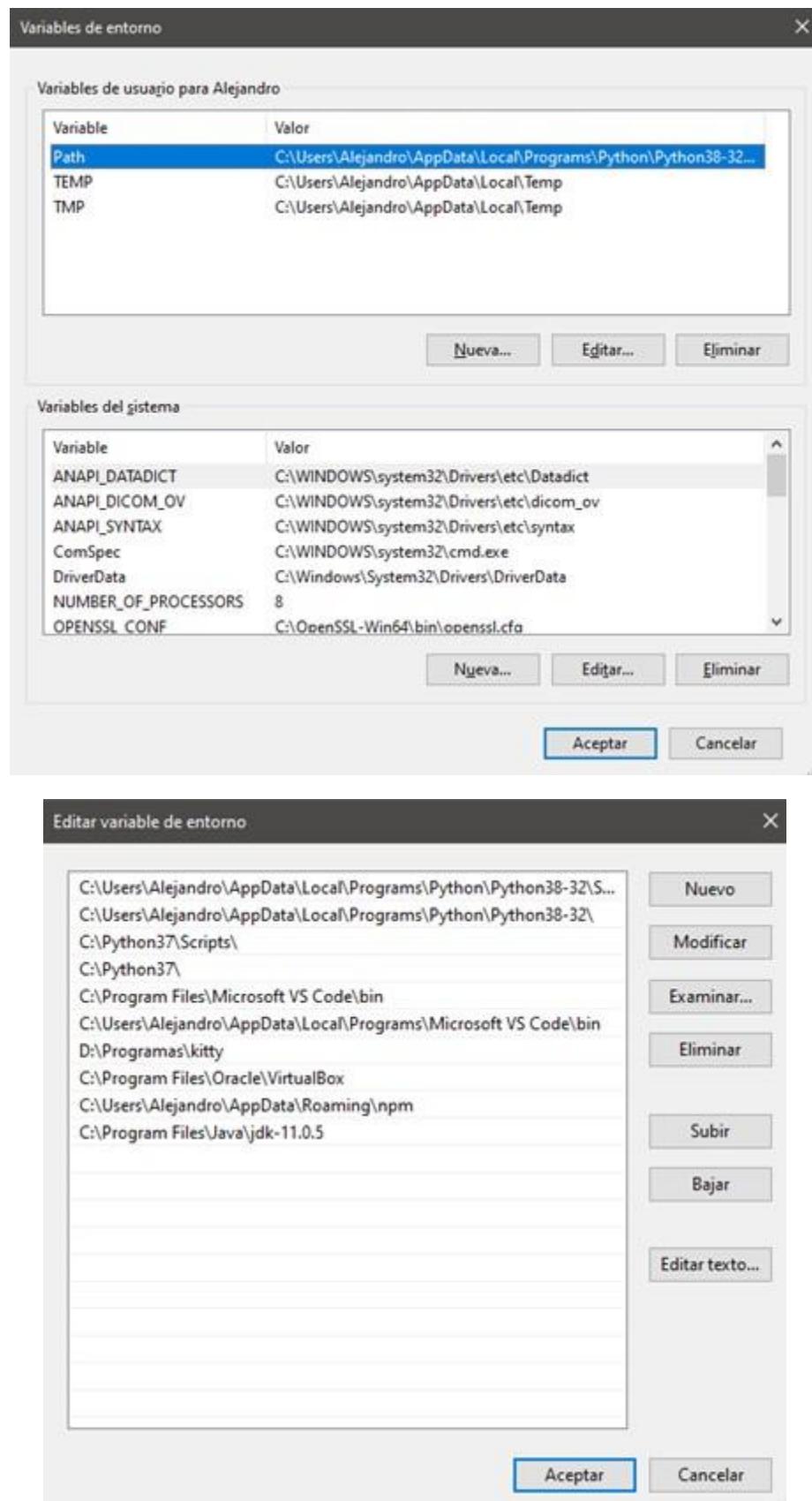
- A continuación comenzará el proceso de instalación.



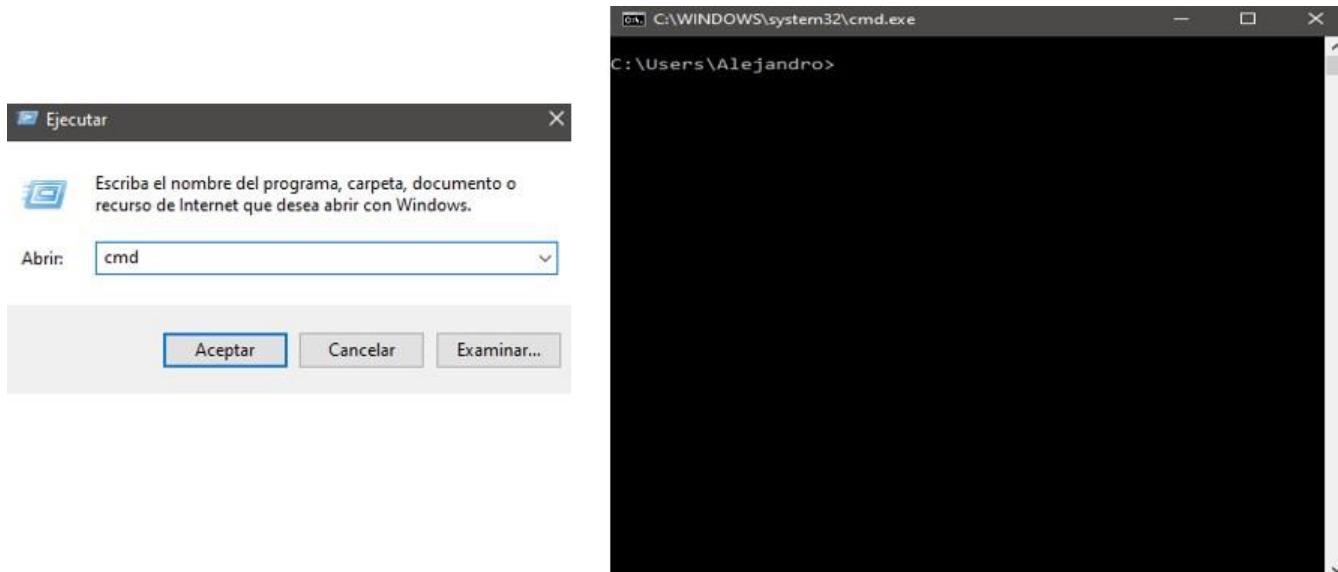
- Al finalizar, cerrar el asistente de instalación.



- Configurar la variable de entorno PATH para apuntar al directorio bin del JDK:
  - Panel de control -> Sistema -> Configuración avanzada del sistema -> Variables de entorno
  - Pulsar en Path en el recuadro superior de "Variables de usuario" y a continuación, en Editar.
  - Pulsar en Nuevo y añadir el directorio bin del JDK (generalmente será C:\Program Files\Java\jdk-11.0.8\bin o similar).
  - Pulsar en Aceptar en todas las ventanas.



- Comprobar que la instalación se ha efectuado correctamente. Para ello, es necesario abrir una consola de terminal de comandos en Windows.



- Ejecutar *javac -version* para comprobar la versión del compilador *javac* instalado. Este comando transforma el código fuente de Java en bytecode.

```
C:\Users\master>javac -version
javac 11.0.8
```

- Ejecutar *java -version* para comprobar la versión del comando *java* instalado. Este comando permite la ejecución de *bytecode*.

```
C:\Users\master>java -version
java version "11.0.8" 2020-07-14 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.8+10-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.8+10-LTS, mixed mode)
```

- También puede ejecutarse el comando *jshell* (una consola de Java disponible a partir de la versión 9).

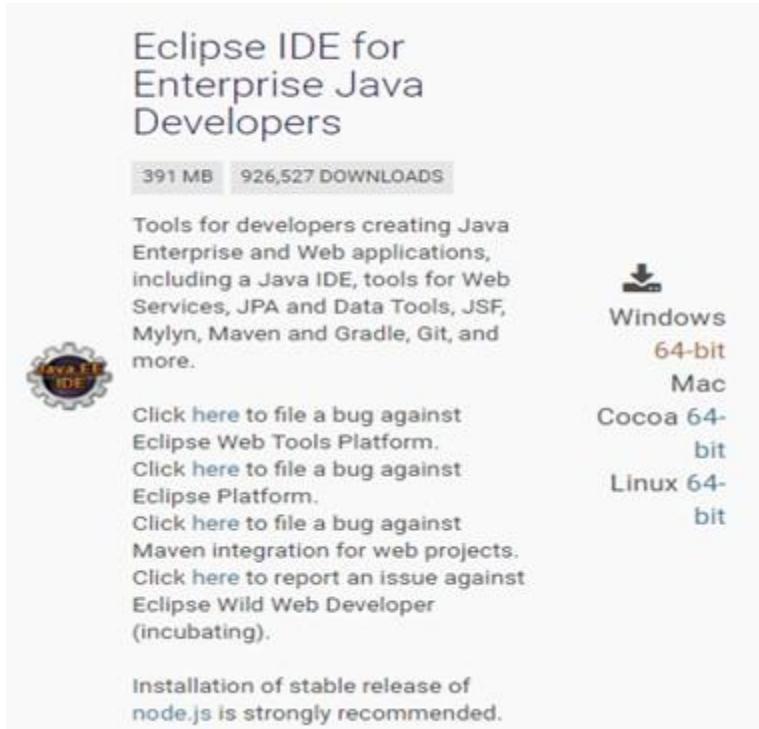
```
C:\Users\master>jshell
| Welcome to JShell -- Version 11.0.8
| For an introduction type: /help intro

jshell> 1+2
$1 ==> 3

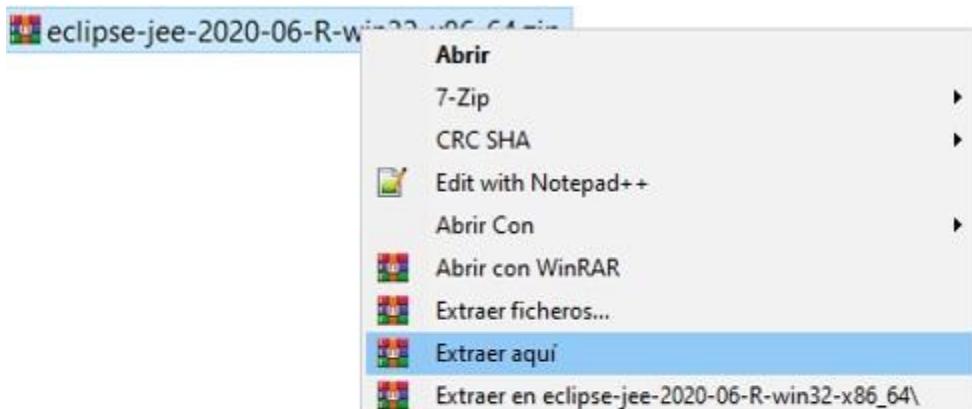
jshell> 25*3
$2 ==> 75

jshell> /exit
| Goodbye
```

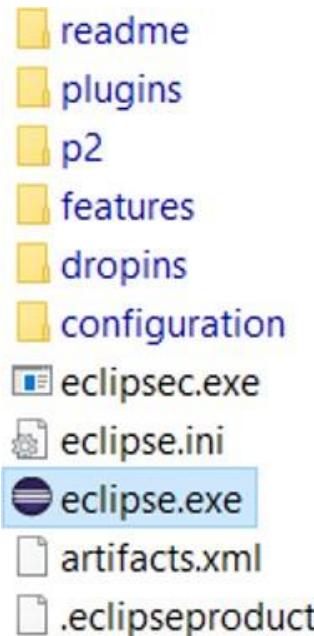
- Eclipse puede descargarse desde su [página web oficial]  
[\(https://www.eclipse.org/downloads/packages/\)](https://www.eclipse.org/downloads/packages/)
- Concretamente es necesario descargar Eclipse IDE for Enterprise Java Developers.



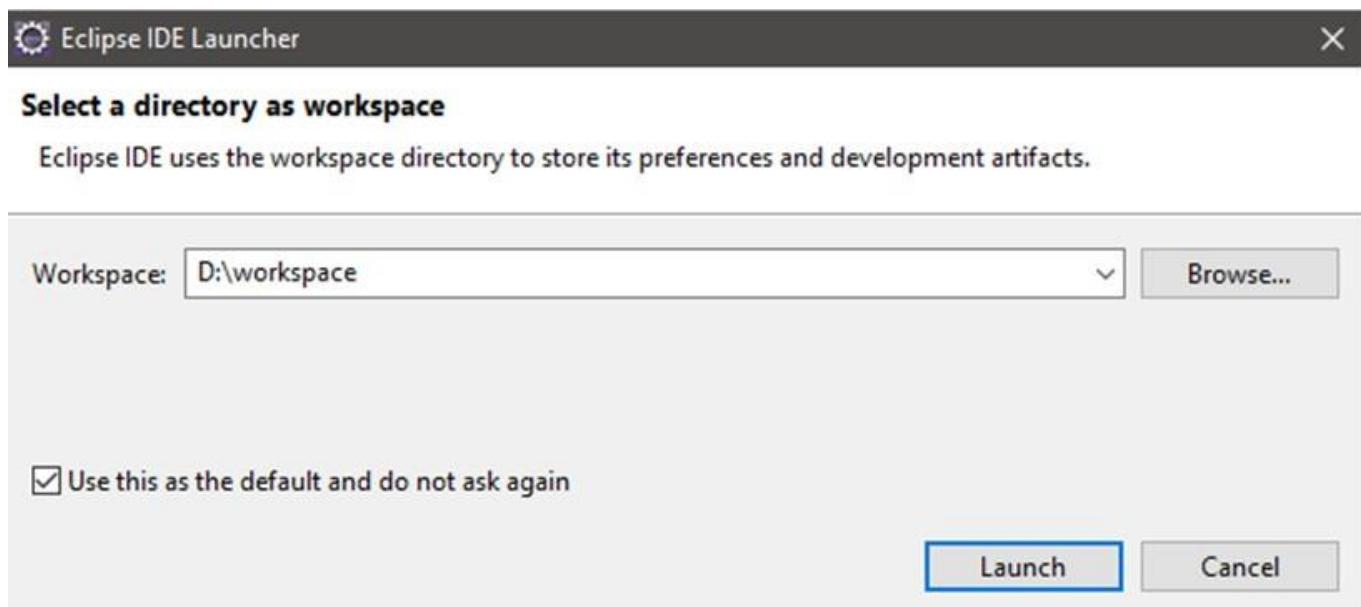
- A continuación se descomprime el archivo descargado.



- Acceder al directorio eclipse y ejecutar el archivo eclipse.exe.



- Seleccionar el directorio de trabajo (donde se almacenará todo el código fuente de los programas). Debería de ser un directorio vacío.

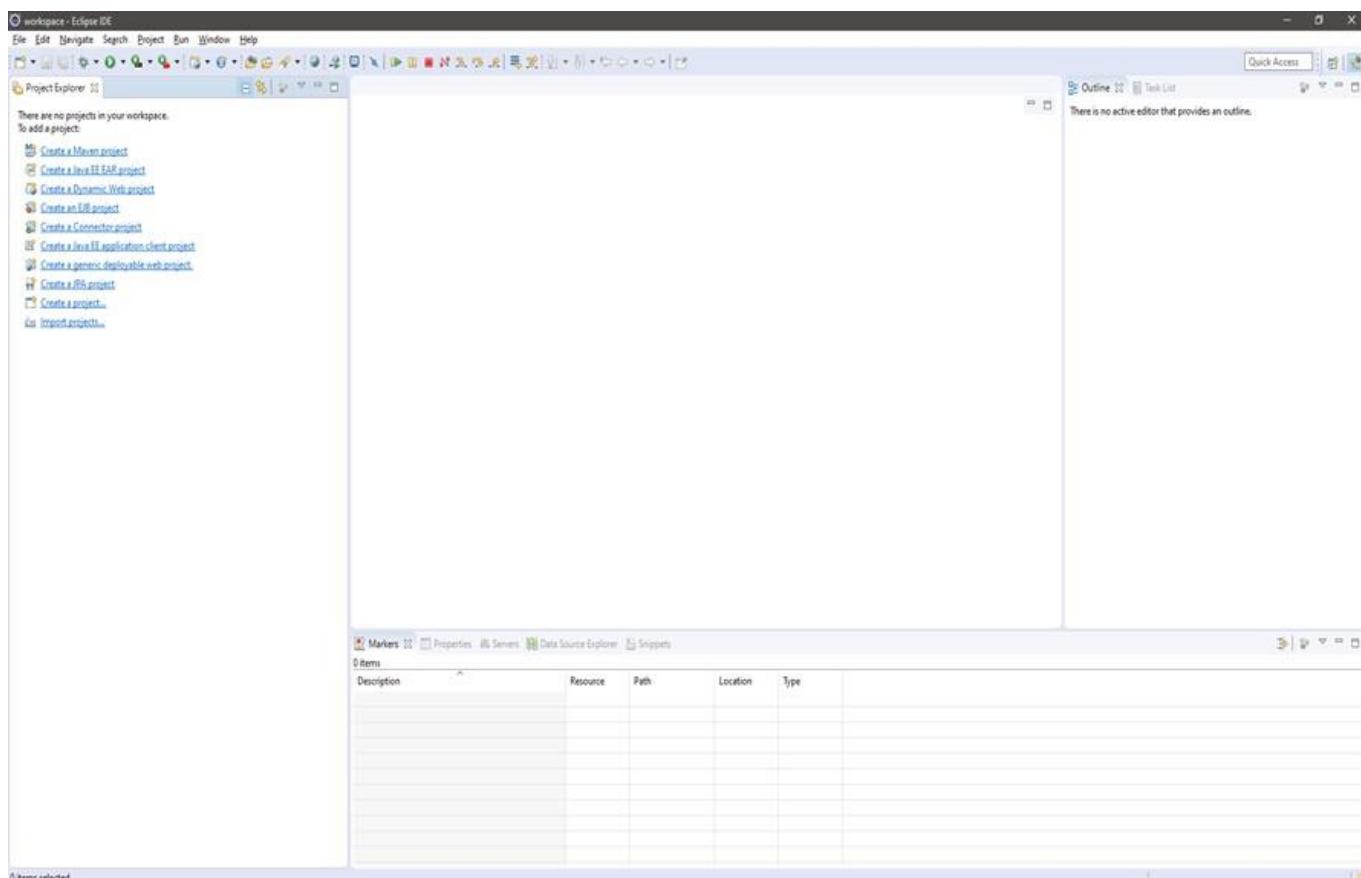
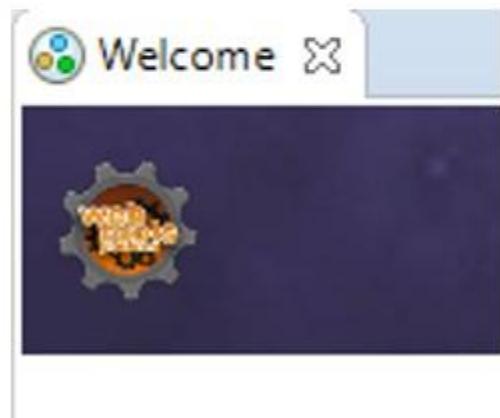


- Eclipse EE IDE debe abrirse y presentar una interfaz similar a la siguiente:



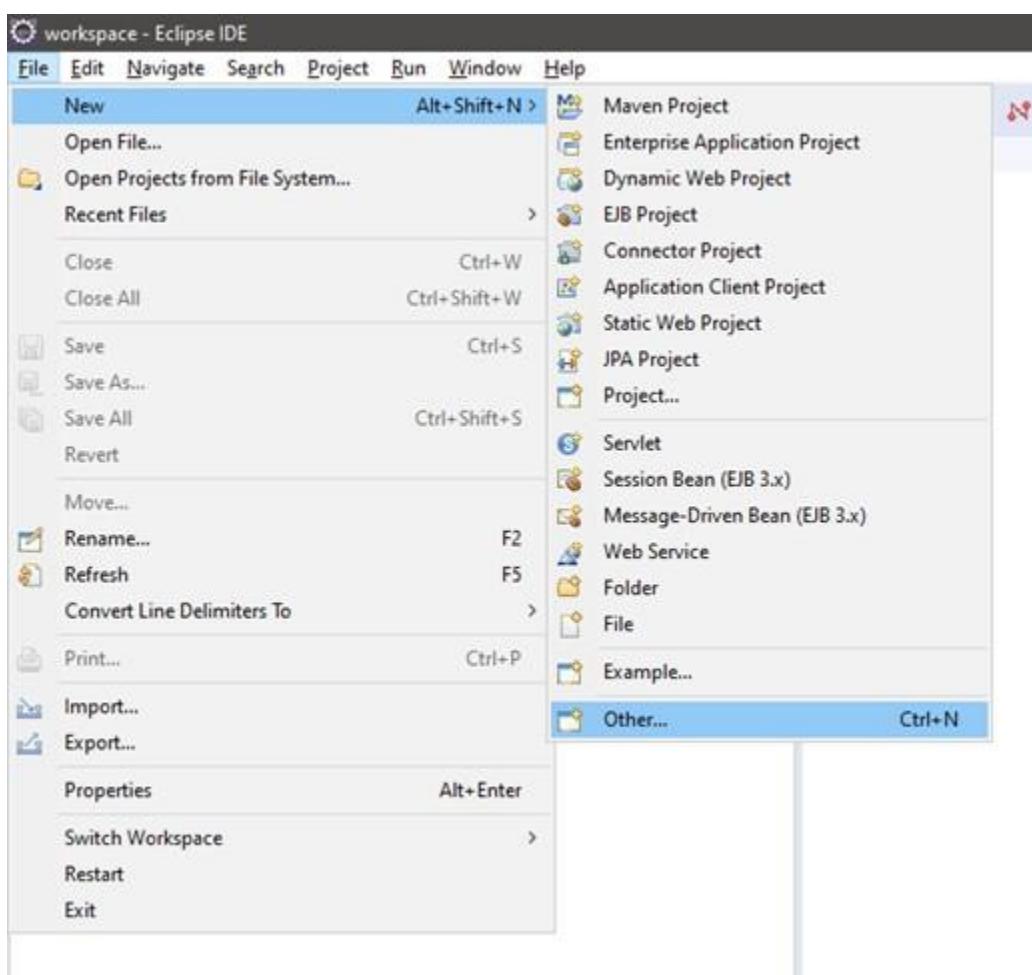
- Desactivar la casilla "Always show Welcome at start up" (en la parte inferior derecha de la pantalla) y pulsar la cruz de Welcome para cerrar la pestaña de bienvenida de Eclipse.

## Always show Welcome at start up

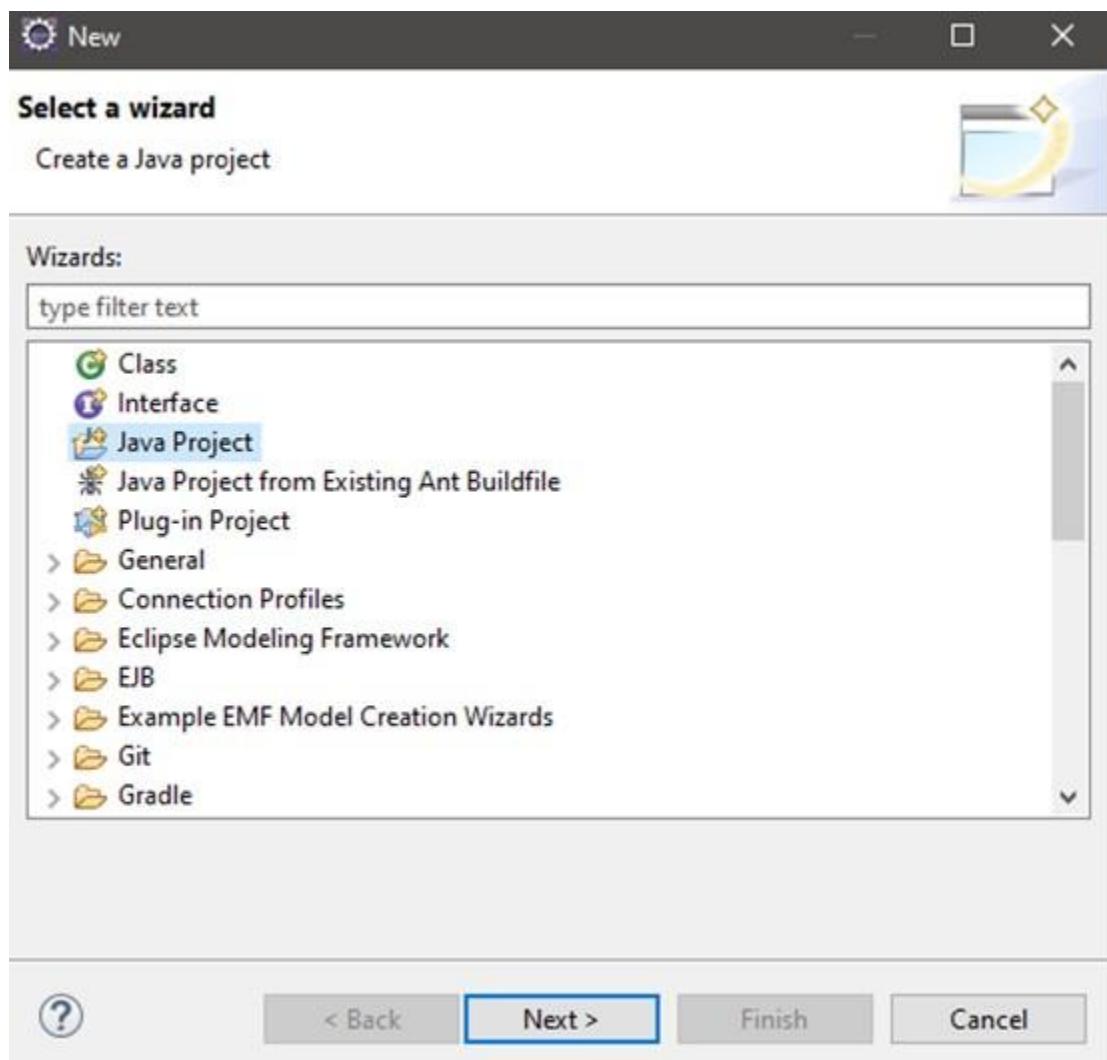


## Primer programa en Java

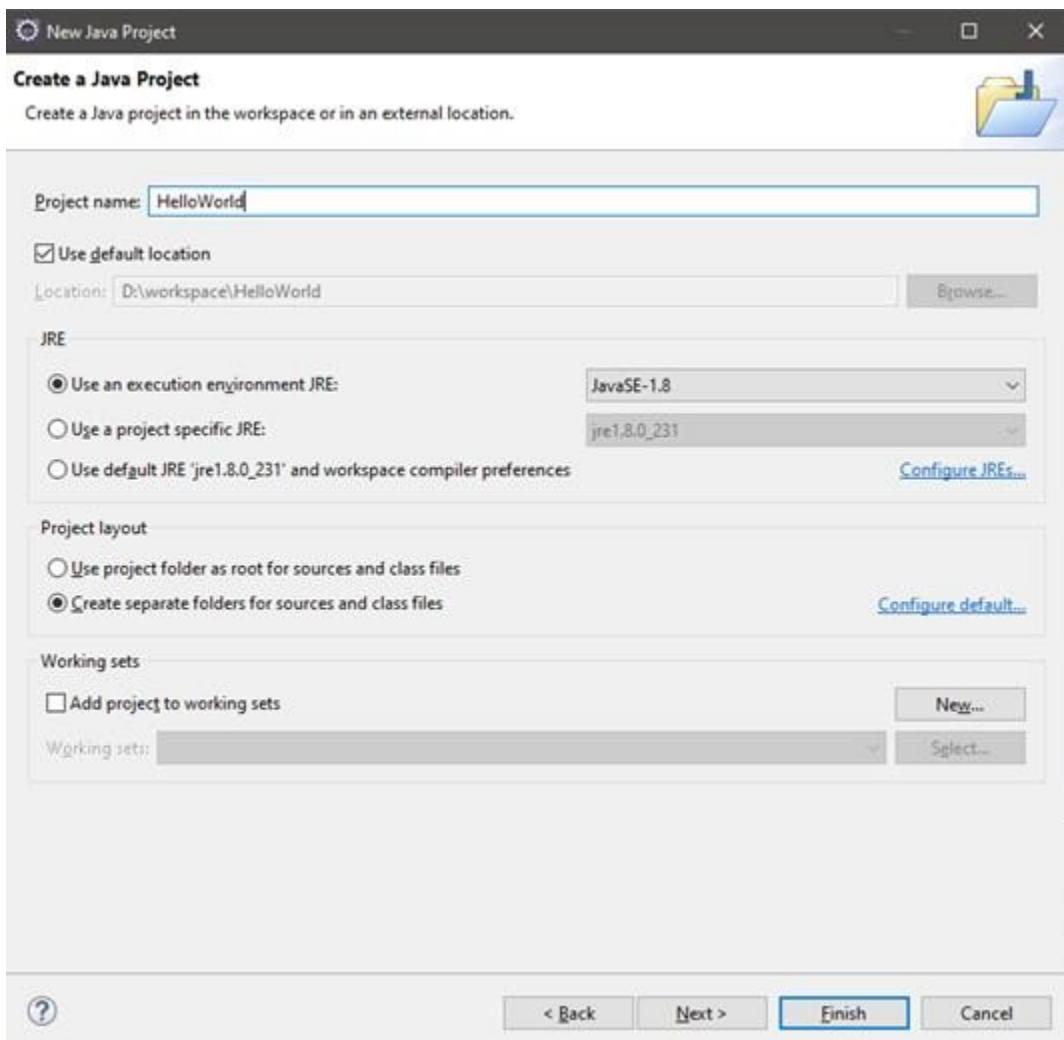
- Crear un nuevo Proyecto básico de Java.
  - *File -> New -> Others*



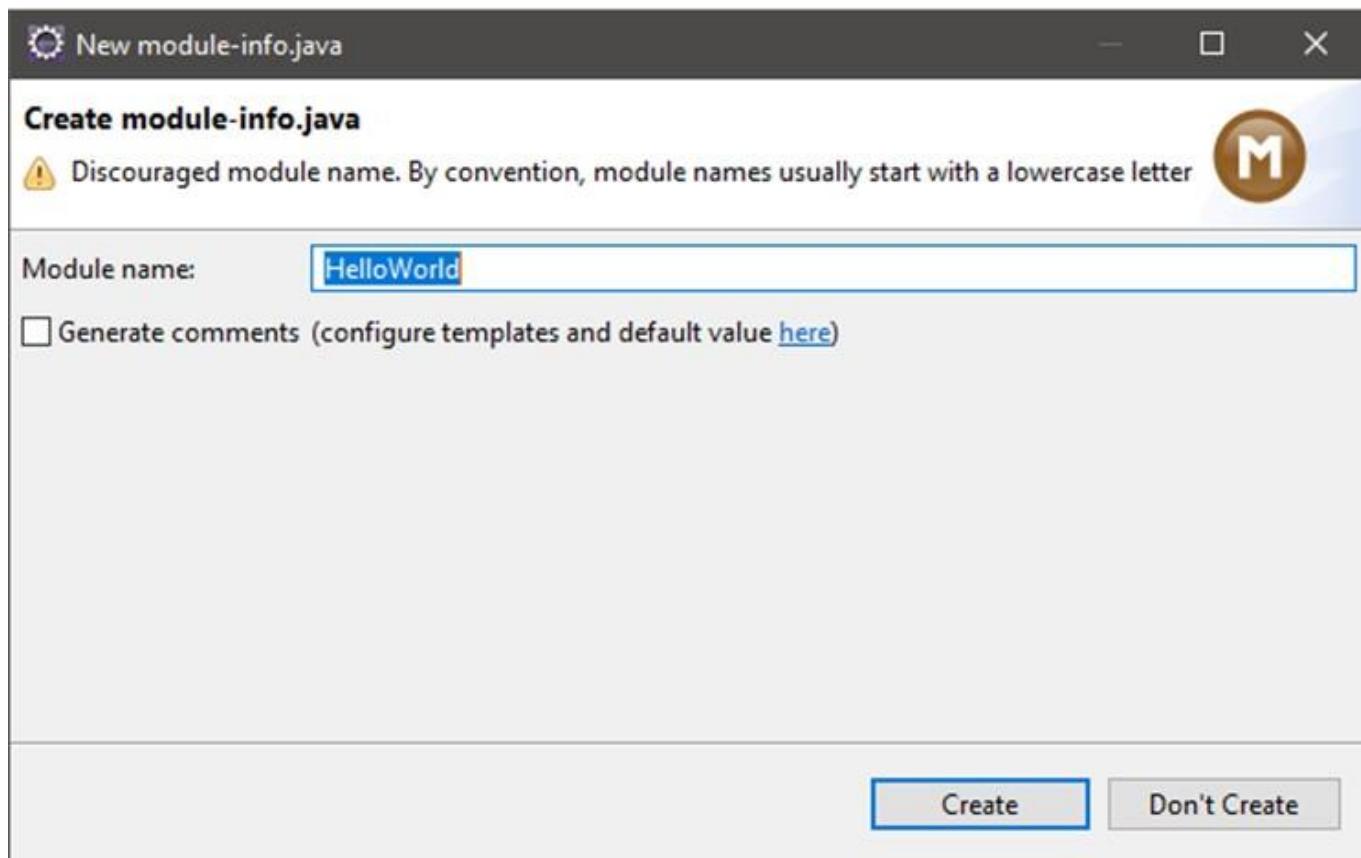
- Seleccionar Java Project.



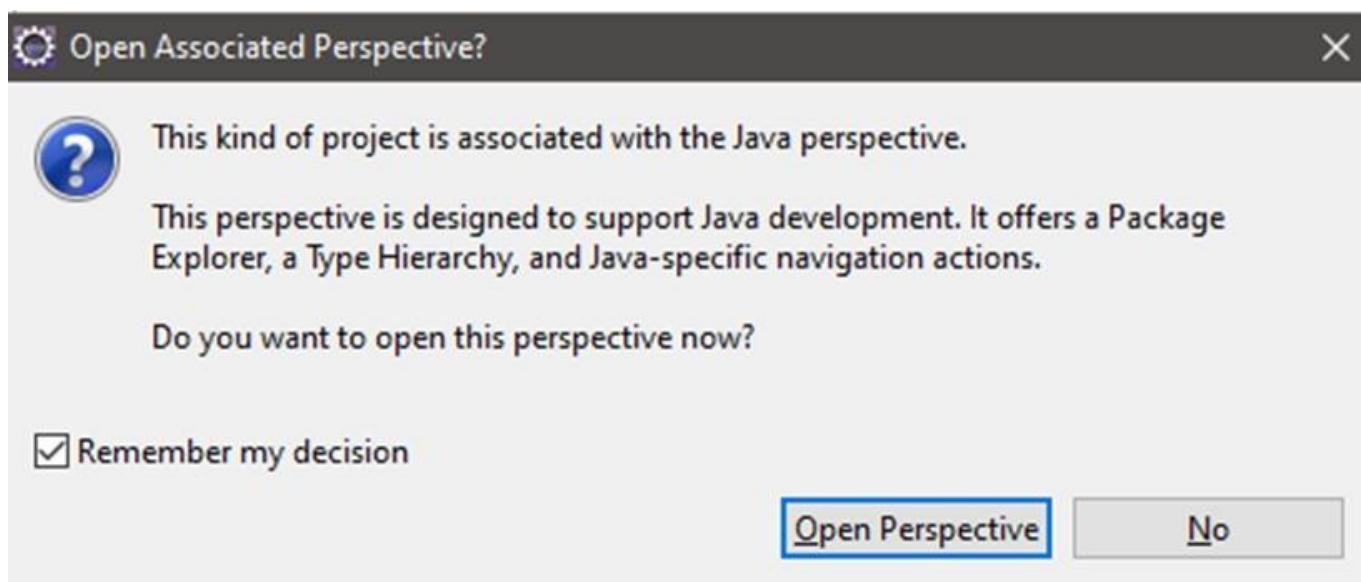
- ♦ Escribir el nombre del proyecto y finalizar.



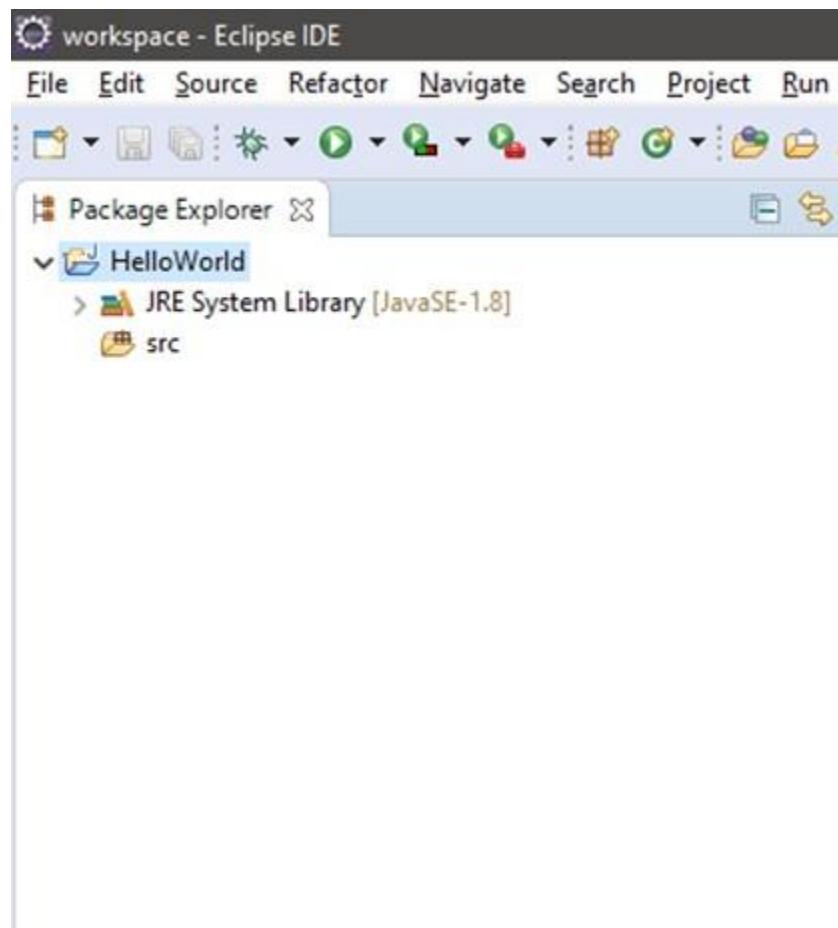
- Aparecerá una ventana para preguntar acerca de la creación de un módulo. Pulsar en *Don't Create*



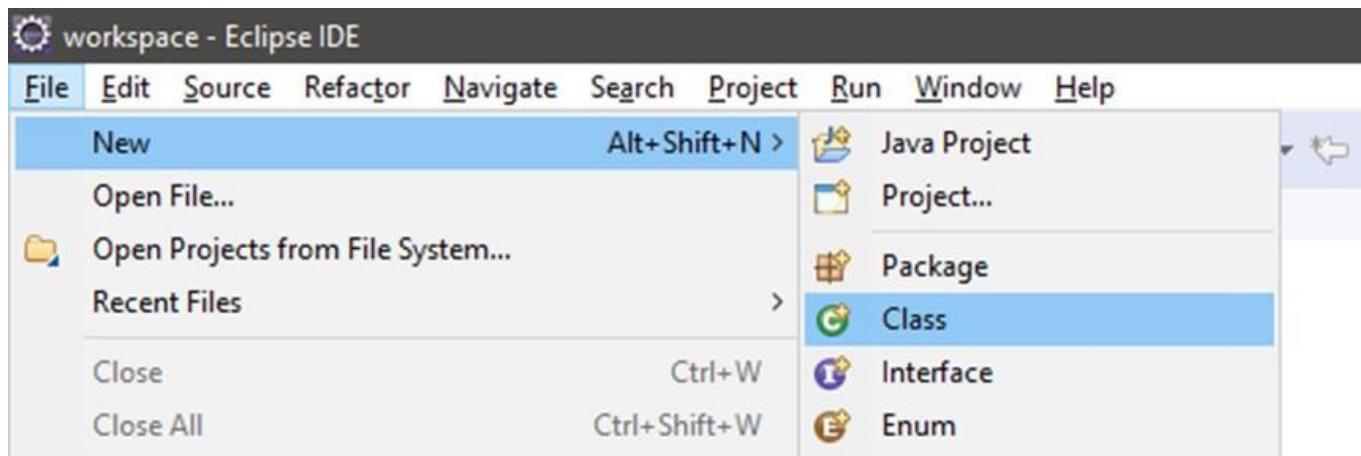
- Una nueva ventana aparecerá de nuevo. Hay que pulsar en "Open Perspective" (por defecto la perspectiva es la de Java EE).



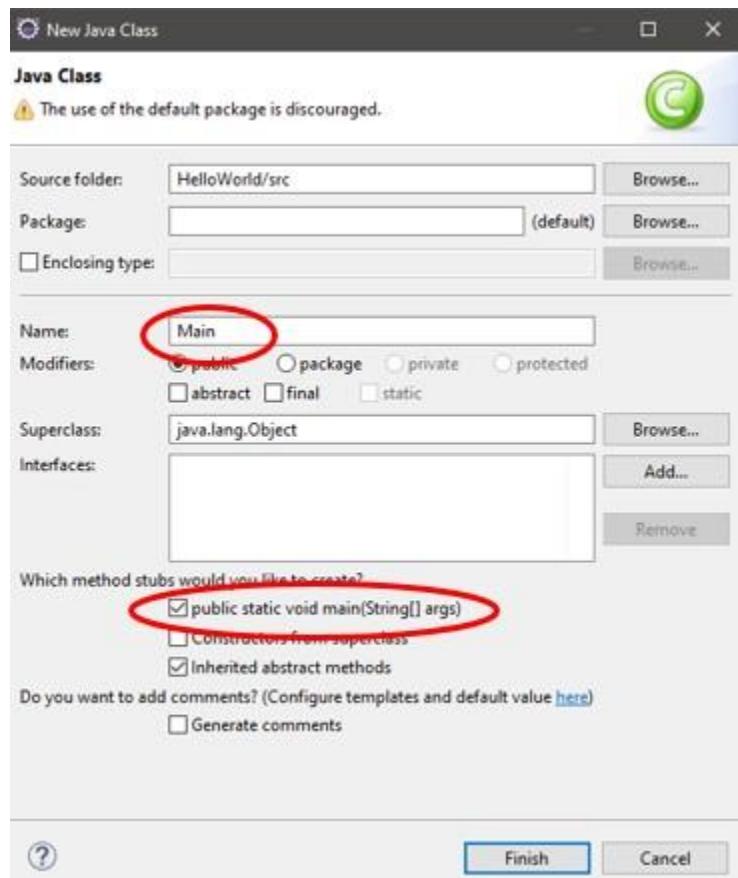
- En el explorador de paquetes aparece el nuevo proyecto. El directorio src contendrá todo los archivos de código fuente Java.



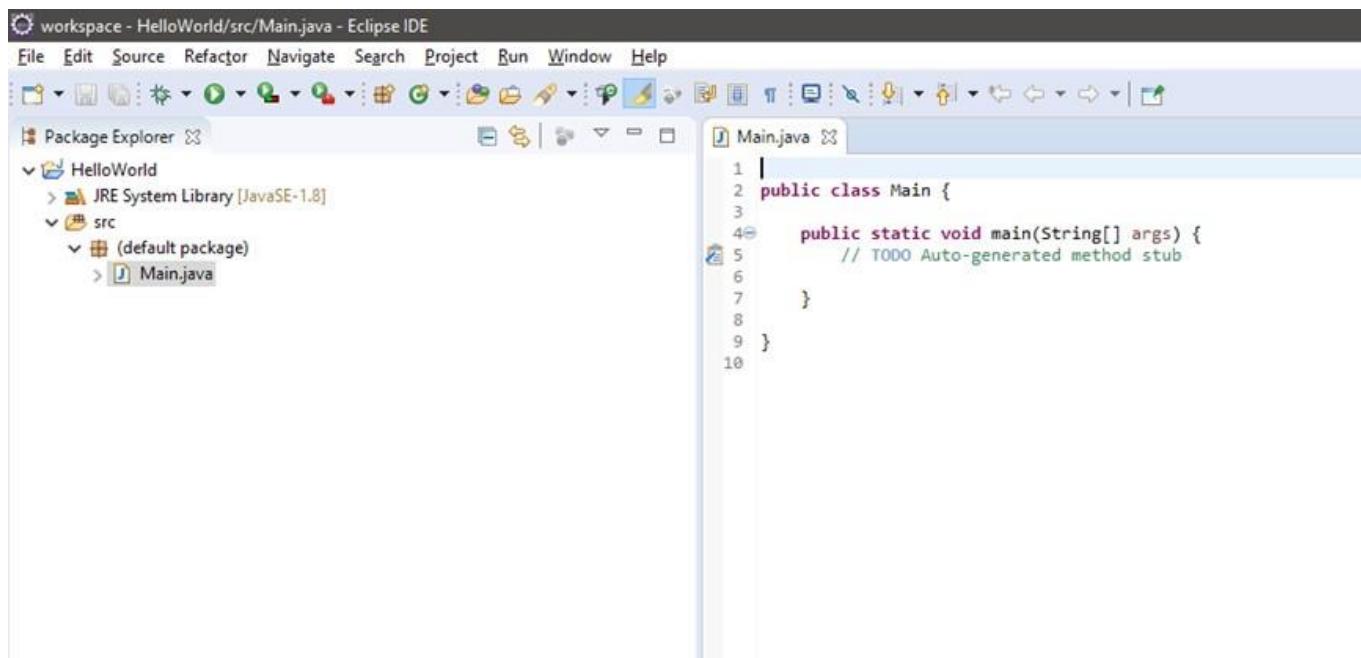
- Crear un nuevo archivo Java.
  - New -> Class



- Escribir un nombre de la clase (que será también el nombre del archivo).
  - Marcar la opción `public static void main (String[] args)`.



- El archivo nuevo creado se abrirá con un código mínimo de ejecución.



- Windows -> Preferences
- General -> Appearance -> Color and Fonts
- En el recuadro de la derecha -> Basic -> Text Font
- Botón Edit

- Escribir una línea de código con la instrucción System.out.println para imprimir por pantalla un texto y guardar el archivo (CTRL + S).

```
public class Main {
```

```
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("¡Hola mundo!");  
    }
```

```
}
```

- Compilar y ejecutar el archivo fuente:
  - Botón derecho en el archivo java
  - *Run As -> Java Application*
- El resultado aparecerá en la consola de Eclipse, en la parte inferior de la pantalla

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The console window displays the output of a Java application named 'Main'. The output text is: '<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0\_231\bin\javaw.exe (14 nov. 2019 19:29:00)  
¡Hola mundo!'. The tabs at the top of the console are 'Problems', 'Javadoc', 'Declaration', and 'Console'.

- También se puede compilar y ejecutar el programa mediante línea de comandos.

```
D:\workspace\HelloWorld\src>javac Main.java

D:\workspace\HelloWorld\src>dir
El volumen de la unidad D es Datos
El número de serie del volumen es: 93A2-51A0

Directorio de D:\workspace\HelloWorld\src

14/11/2019  19:30      <DIR>          .
14/11/2019  19:30      <DIR>          ..
14/11/2019  19:31                415 Main.class
14/11/2019  19:29                158 Main.java
                           2 archivos           573 bytes
                           2 dirs   103.299.911.680 bytes libres

D:\workspace\HelloWorld\src>java Main
¡Hola mundo!

D:\workspace\HelloWorld\src>
```

Tu carrera digital ~

# Módulo 3

## Java básico

### Fundamentos



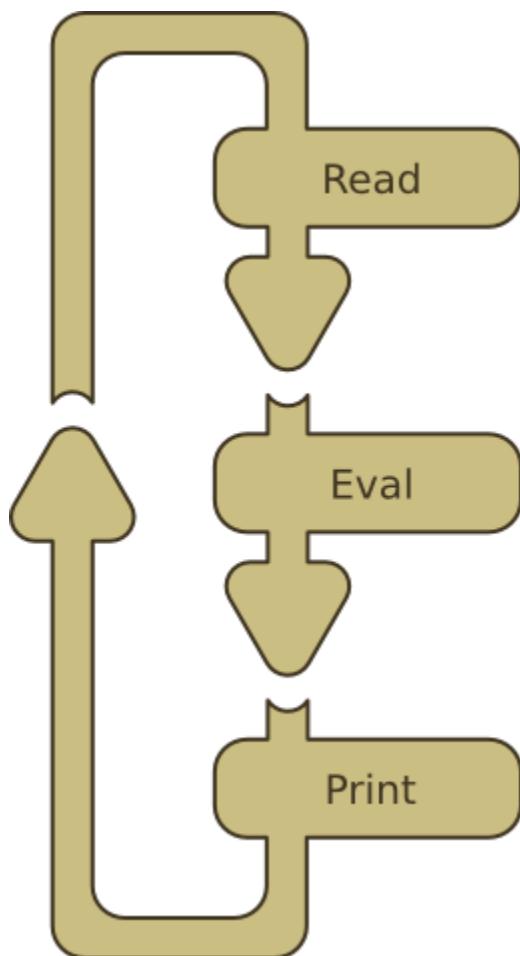
# Fundamentos

---

- ◆ JShell
- ◆ Operadores
- ◆ System.out.println
- ◆ Espacios en blanco
- ◆ Case sensitive
- ◆ Secuencias de escape
- ◆ System.out.printf
- ◆ Comentarios
- ◆ Variables
- ◆ Tipos de variable
- ◆ Conversión de tipos

## JShell

- ◆ JShell es una herramienta de programación que provee una interfaz REPL. El término REPL está referido a:
  - R: significa lectura (lee el código de Java).
  - E: significa evaluación (evalúa el código de Java).
  - P: significa imprimir (muestra el resultado por pantalla).
  - L: significa bucle o lazo (finaliza y vuelve de nuevo al inicio para esperar nuevo código Java).



- JShell está disponible a partir de la versión 9 de Java, pero puede probarse también online en [TryJShell](#).

```
| Welcome to JShell -- Version 14-ea
| For an introduction type: /help intro

jshell> /set editor /usr/bin/vim
| Editor set to: /usr/bin/vim

jshell> 
```

- Sin embargo, es preferible ejecutarlo desde el propio equipo.

```
C:\Users\master>jshell
| Welcome to JShell -- Version 11.0.5
| For an introduction type: /help intro

jshell>
```

- Una vez ejecutado, JShell se comporta como un intérprete de comandos para código Java. Puede consultarse la ayuda de JShell escribiendo /help.

```
jshell> /help
Type a Java language expression, statement, or declaration.
Or type one of the following commands:
/list [<name or id>|-all|-start]
    list the source you have typed
/edit <name or id>
    edit a source entry
/drop <name or id>
    delete a source entry
/save [-all|-history|-start] <file>
    Save snippet source to a file
/open <file>
    open a file as source input
/vars [<name or id>|-all|-start]
    list the declared variables and their values
/methods [<name or id>|-all|-start]
    list the declared methods and their signatures
/types [<name or id>|-all|-start]
    list the type declarations
/imports
    list the imported items
/exit [<integer-expression-snippet>]
    exit the jshell tool
```

- JShell puede interpretar expresiones Java válidas.

```
C:\Users\master>jsshell
| Welcome to JShell -- Version 11.0.5
| For an introduction type: /help intro

jshell> 5*2
$1 ==> 10

jshell> 3/2
$2 ==> 1

jshell> 5+2
$3 ==> 7

jshell> 1-5
$4 ==> -4

jshell> -
```

- ◆ Para cada una de las expresiones anteriores, JShell evalúa, imprime el resultado y vuelve a esperar la introducción de nuevas expresiones.
- ◆ \$1, \$2, \$3, \$4, ... son variables especiales asignadas al resultado de cada una de las expresiones ejecutadas.
  - Este tipo de variables se asimilan en funcionamiento a la variable Ans de las calculadoras.
- ◆ Una variable de este tipo (\$) se creará cada vez que se ejecute una nueva expresión en JShell, incrementando su valor en cada ejecución.
- ◆ Puede accederse a cada una de estas variables de forma directa escribiendo el nombre en la consola de JShell.
  - Al acceder directamente a estas variables especiales, se incrementa el valor asociado a \$, pero no se crea la variable como tal.

```
jshell> 5*2
$1 ==> 10

jshell> 3/2
$2 ==> 1

jshell> 5+2
$3 ==> 7

jshell> 1-5
$4 ==> -4

jshell> $1
$1 ==> 10

jshell> $2
$2 ==> 1

jshell> 3*2
$7 ==> 6

jshell> $6
| Error:
| cannot find symbol
|   symbol:   variable $6
| $6
| ^^^

jshell> $7
$7 ==> 6

jshell> -
```

- Puede ejecutarse la instrucción /reset para inicializar el valor asociado a \$.

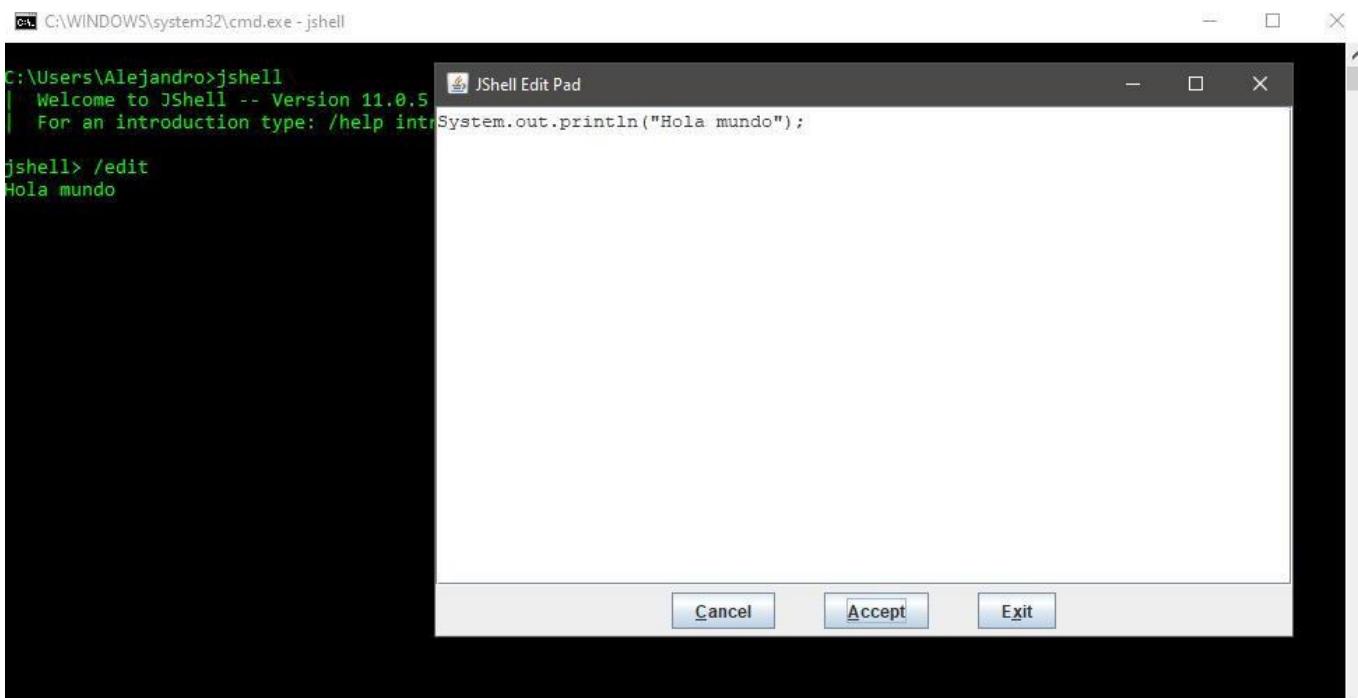
```
jshell> $7
$7 ==> 6

jshell> /reset
| Resetting state.

jshell> 5+2
$1 ==> 7

jshell>
```

- JShell también incorpora un modo edición al que puede accederse con /edit.



- La instrucción /history permite listar todos las sentencias ejecutadas en JShell.

```
C:\Users\Alejandro>jshell
| Welcome to JShell -- Version 11.0.5
| For an introduction type: /help intro

jshell> 2 + 1
$1 ==> 3

jshell> 2 * 5
$2 ==> 10

jshell> /history

2 + 1
2 * 5
/history

jshell>
```

- Para salir de JShell hay que pulsar la combinación de teclas CTRL+D o escribir /exit.

## Operadores

- Las expresiones anteriores están compuestas por operandos y por operador. Por ejemplo, en la expresión  $5*3$ :
  - 5 y 3 son operandos.
  - El símbolo \* es un operador que representa la operación de multiplicación.
- Hay diferentes tipos de operadores. Los más importantes son listados a continuación.

### Operador Significado

+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo

- La consola JShell generará un error al escribir un operador que no existe.

```
jshell> 2$3
| Error:
| ';' expected
| 2$3
| ^
|
| Error:
| missing return statement
| 2$3
| ^-^
```

- Pueden utilizarse varios operadores para conformar expresiones más complejas, aunque siempre teniendo en cuenta que los operadores de multiplicación y división tienen preferencia con respecto a las suma y la resta.

```
jshell> 2+4*3+2
$16 ==> 16

jshell> 2+4*3/2
$17 ==> 8
```

- Los paréntesis tienen preferencia sobre todos los operadores.

```
jshell> (2+4)*3/2
$18 ==> 9
```

- Los operandos vistos hasta este momento son todos valores de tipo entero (sin parte decimal). Sin embargo, también se pueden realizar operaciones matemáticas con operandos de tipo flotante (con parte decimal).
  - Las operaciones con números enteros producirán siempre resultados de números enteros.
  - Las operaciones con al menos un número de tipo flotante producirá un resultado de número flotante.

```
jshell> 1/2
$19 ==> 0

jshell> 1/2.0
$20 ==> 0.5

jshell> 1.0/2
$21 ==> 0.5

jshell> 2.0/1
$22 ==> 2.0
```

Ejercicio: escribe una expresión para calcular la cantidad de minutos en un día.

Ejercicio: escribe una expresión para calcular la cantidad de segundos en un día.

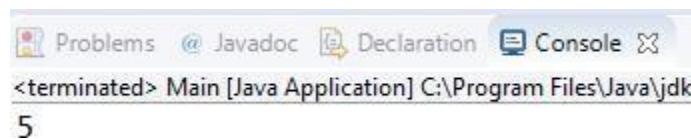
## System.out.println

- Todas las expresiones vistas anteriormente son válidas en la consola JShell. Sin embargo, no son totalmente correctas a nivel de código de programación en Java.
- Todo el código que se ejecuta en un programa Java se encuentra encerrado entre las llaves ({ }) abiertas por la sentencia "public static void main(String[] args)".

```
1
2 public class Main {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         3+2
7     }
8
9 }
10
```

- A nivel de programación en Java, la instrucción que muestra por pantalla datos es `System.out.println`.
  - El dato a mostrar se encierra entre dos paréntesis (uno de apertura y otro de cierre).
  - La sentencia debe finalizar con un punto y coma para que sea sintácticamente válida.

```
public class Main {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.print(3+2);  
    }  
  
}
```



- La instrucción `System.out.println` también puede ejecutarse en la consola JShell (en este caso no sería necesario el punto y coma final).

```
jshell> System.out.println(3*2)  
6
```

- La instrucción `System.out.println` posee una sintaxis muy estricta que es necesario respetar.

```
jshell> System.out.println3*4)
|   Error:
|     ';' expected
|     System.out.println3*4)
|                           ^
|
|   Error:
|     cannot find symbol
|       symbol:   variable println3
|     System.out.println3*4)
|   ^-----^

jshell> System.out.println 3*4
|   Error:
|     ';' expected
|     System.out.println 3*4
|           ^
|
|   Error:
|     cannot find symbol
|       symbol:   variable println
|     System.out.println 3*4
|   ^-----^

jshell> System.out.println( 3 * 4 )
12
```

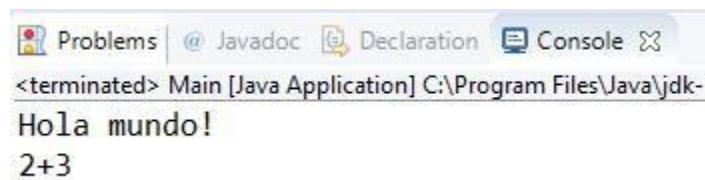
- `System.out.println` también permite imprimir por pantalla texto encerrado entre comillas dobles.

```
jshell> System.out.println("Hola mundo!")
Hola mundo!

jshell> System.out.println("2+3")
2+3
```

- El ejemplo anterior sería correcto a nivel de programación Java si se añade el punto y coma al final.

```
public class Main {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("Hola mundo!");  
        System.out.println("2+3");  
    }  
}
```



Ejercicio: escribe un programa en Java que realice las siguientes tareas:

1. Mostrar por pantalla Hello world.
2. Mostrar por pantalla la expresión 2\*3 como texto.
3. Mostrar por pantalla el resultado de la expresión 2\*3.

## Espacios en blanco

- Los espacios en blanco afectan a la salida por pantalla cuando se encuentran entre comillas dobles.

```
jshell> System.out.println("Hello           world");  
Hello           world
```

- Sin embargo, no sucede lo mismo cuando se ejecuta una expresión matemática.

```
jshell> 1+2  
$1 ==> 3  
  
jshell> 1+      2  
$2 ==> 3  
  
jshell> 1 + 2  
$3 ==> 3  
  
jshell> System.out.println(3      * 2)  
6
```

## Case sensitive

- Java es case sensitive, es decir, es sensible a minúsculas y mayúsculas. Por ejemplo, la instrucción System.out.println() debe escribirse tal cual (con S mayúscula y con el resto de caracteres en minúsculas). Java generará un error si no se respeta esta sintaxis.

```
jshell> system.out.println("Hello world");
| Error:
| package system does not exist
| system.out.println("Hello world");
| ^-----^

jshell> System.Out.println("Hello world");
| Error:
| cannot find symbol
|   symbol:   variable Out
| System.Out.println("Hello world");
| ^-----^
```

- No hay ningún problema en utilizar mayúsculas y minúsculas en un texto encerrado entre comillas dobles.

```
jshell> System.out.println("HELLo WoRlD");
HElLo WoRlD
```

## Secuencias de escape

- Una secuencia de escape es una combinación de caracteres que tiene un significado diferente a los caracteres literales contenidos en ella.
- En Java, las secuencias de escape comienzan con el carácter de barra invertida \.
- Si se pretende imprimir por pantalla el carácter de comilla doble (""), es necesario escaparlo previamente con \.
  - Si no se escapa, Java interpreta al carácter " como el final de la cadena de texto, mostrando un error.

```
jshell> System.out.println("esto " es una comilla")
| Error:
| ')' expected
| System.out.println("esto " es una comilla")
| ^
```

- La combinación de \ con distintos caracteres da lugar a las secuencias de escape.

**Secuencia de escape      Comportamiento**

\t	Tabulación
\n	Salto de línea
\\	Imprime \
\"	Imprime "
'	Imprime '

```
jshell> System.out.println("12345\t67890")
12345    67890

jshell> System.out.println("12345\n67890")
12345
67890

jshell> System.out.println("12345\\67890")
12345\67890

jshell> System.out.println("12345\"67890")
12345"67890

jshell> System.out.println("12345\'67890")
12345'67890
```

## System.out.printf

- Con System.out.println pueden existir problemas cuando se pretende combinar la visualización de valores numéricos con cadenas de texto.

```
jshell> System.out.println("5*2=" 5*2)
| Error:
|   ')' expected
|   System.out.println("5*2=" 5*2)
|
```

- Para mezclar varios tipos de datos (texto con números) puede utilizarse System.out.printf combinado con println.
- System.out.printf permitir definir caracteres de formato especiales dentro de las comillas dobles.

- El carácter de formato especial más importante es %d, que denota el espacio en el texto que debe ser sustituido por un número entero.
  - El número entero a sustituir se define a continuación del cierre de la cadena de texto y tras una coma.

```
jshell> System.out.printf("5*2=%d",5*2).println()
5*2=10
```

- Pueden establecerse tantos caracteres de formato especial como se quiera. Serán sustituidos por los valores numéricos establecidos a continuación en estricto orden.

```
jshell> System.out.printf("%d %d %d", 5, 7, 5).println()
5 7 5
```

```
jshell> System.out.printf("%d %d %d", 5, 7, 5*7).println()
5 7 35
```

```
jshell> System.out.printf("%d * %d = %d", 5, 7, 5*7).println()
5 * 7 = 35
```

Ejercicio: imprime por pantalla  $5 + 6 + 7 = 18$ , siendo:

- $5 + 6 + 7$  = una cadena de texto literal.
- 18 el resultado de realizar la operación matemática anterior.
- Si se establecen más caracteres de formato especial que números separados por comas, entonces se generará un error.

```
jshell> System.out.printf("%d + %d + %d = %d", 5, 6, 7).println()
5 + 6 + 7 = | Exception java.util.MissingFormatArgumentException:
MissingFormatArgumentException: Format specifier '%d'
```

- En cambio, si se establecen menos, entonces el resto serán omitidos.

```
jshell> System.out.printf("%d + %d", 5, 6, 7).println()
5 + 6
```

- No puede utilizarse el carácter de formato especial %d para valores de tipo flotante (con parte decimal). En este caso debe utilizarse %f.

```
jshell> System.out.printf("%f + %f + %f", 5.5, 6.5, 7.5).println()
5,500000 + 6,500000 + 7,500000
```

## Comentarios

- Un comentario es una anotación legible incrustada en el código de programación y sirve facilitar al desarrollador la comprensión del programa.
- Los comentarios son ignorados por el compilador de Java.
- Los comentarios en Java pueden ser de dos tipos:
  - Comentarios de una línea: comienzan con dos barras (//)
  - Comentarios multilínea: comienzan con /\* y finaliza con \*/

```
public class Main {  
  
    public static void main(String[] args) {  
  
        // la siguiente línea muestra por pantalla Hello world  
        System.out.println("Hello world");  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        /* la  
         * siguiente  
         * línea  
         * muestra  
         * por pantalla Hello world  
         */  
        System.out.println("Hello world");  
    }  
}
```

- Un comentario preciso y conciso es más apropiado que uno largo, repetitivo y complicado.

## Variables

- Las variables son espacios de memoria en el ordenador donde se almacena un dato.
- El dato almacenado puede ser de diferente tipo: entero, flotante, booleano u otros más complejos.
- La declaración de una variable consta de las siguientes partes:
  - Tipo de variable: int (número sin parte decimal), float (número con parte decimal), boolean (solo acepta dos valores: true o false), etc.
  - Nombre de la variable: puede establecerse cualquier cadena alfanumérica.

- El Carácter igual (=) indica que se le va asignar un valor a una variable.
- Valor de la variable: debe ser un valor del tipo declarado previamente.
- Carácter ;

```
/*
Tipo de variable: int (número sin parte decimal)
Nombre de la variable: a
Valor de la variable: 10
*/
int a = 10;

// el valor de la variable puede mostrarse mediante System.out.println
System.out.println(a);
```

- ◆ Los nombres de las variables deben comenzar por una letra y deben evitarse caracteres especiales (aunque algunos como \_ y \$ están permitidos). No pueden declararse dos variables con el mismo nombre.

```
// variables declaradas correctamente
int E378a = 3;
int A3d = 3;
int asDas_d$ = 4

// error porque ya se ha definido una variable con este nombre
// int E378a = 3;

// error porque no se puede utilizar el carácter - en el nombre de una variable
// int a-3 = 3;

// error porque el nombre de una variable debe comenzar por una letra
// int 4a = 3;

// error porque el nombre de la variable no puede coincidir con palabras reservadas del lenguaje Java (en este caso, int)
// int int a;
```

- ◆ Para escribir bien código Java es conveniente seguir buenas prácticas y recomendaciones. Con respecto al nombre de variables:
  - Debería comenzar siempre por una letra minúscula.
  - Debería utilizarse la notación lower case para nombres de variables formadas por palabras compuestas:
    - En minúscula la primera letra de la primera palabra.
    - En mayúscula la primera letra de las palabras sucesivas.
  - El resto de letras deberían escribirse en minúsculas.

```
// underscore  
int edad_de_persona = 2;  
  
// upper camel case  
int EdadDePersona = 3;  
  
// lower camel case (opción recomendada en Java)  
int edadDePersona = 4;
```

- ◆ El valor de una variable puede ser modificado posteriormente.

```
int c = 10;  
System.out.println(c);  
  
// el valor de la variable c cambia a 11  
c = 11;  
System.out.println(c);
```

- ◆ El valor de una variable puede ser asignado a otra variable.
  - Las dos variables deben ser del mismo tipo.
  - El valor de una variable es copiado a la dirección de memoria del ordenador donde se encuentra la otra variable.

```
int d = 5;  
int e = d;  
int f = d + e;  
System.out.printf("%d + %d = %d", d, e, f).println();
```

- ◆ Pueden declararse también varias variables en una única línea de código.

```
// las tres variables declaradas son de tipo entero  
int a = 1, b = 2, c = 1;
```

Ejercicio: escribe un programa que declare tres variables de nombre a, b y c, con valores de tipo entero. A continuación:

1. Escribe una sentencia que muestre por pantalla la suma de las tres variables utilizando System.out.println.
2. Cambia el valor de la variable c.
3. Escribe de nuevo una sentencia que muestre por pantalla la suma de las tres variables utilizando System.out.printf.

## Tipos de variable

- Java es un lenguaje de programación fuertemente tipado. Esto significa:
  - Cada variable debe ser declarada con un tipo (entero, flotante, booleano, ...).
  - Los valores asignados a una variable pueden cambiar a lo largo del programa, pero deben ser siempre del mismo tipo que el establecido en la declaración de la variable.

```
// error porque el valor de la variable h debe ser de tipo entero (sin parte decimal)
// int h = 5.5
```

```
// declaración correcta de una variable de tipo entero
int i = 4;
```

```
// error porque la variable i es de tipo entero y se está estableciendo como valor una
// cadena de texto
// i = "hola";
```

- Los tipos más básicos que existen en Java se denominan tipos primitivos. Se diferencia de otros tipos más complejos en que comienzan siempre con letra minúscula.

<b>Tipo primitivo</b>	<b>Tipo de valores</b>	<b>Tamaño (bits)</b>	<b>Rango de valores</b>	<b>Ejemplo</b>
byte	Valores enteros	8	-128 a 127	byte b = 5;
short	Valores enteros	16	-32,768 a 32,767	short s = 128;
int	Valores enteros	32	-2,147,483,648 a 2,147,483,647	int i = 40000;
long	Valores enteros	64	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	long l = 2222222222L;
float	Valores de tipo flotante	32	±3.40282347E+38F aproximadamente	float f = 4.0f;
double	Valores de tipo flotante	64	±1.79769313486231570E+308 aproximadamente	double d = 67.0;
char	Valores de tipo carácter	16	'\u0000 a '\uffff'	char c = 'A';
boolean	Valores de tipo booleano	1	true o false	boolean esTrue = true;

- La única diferencia entre los tipos enteros (byte, short, int y long) es el espacio que ocupan en memoria.
- Cuando se declara una variable de tipo long es necesario añadir la letra *l* o *L* al final del valor.

```
byte b = 5;  
short s = 128;  
int i = 40000;
```

```
// en variables de tipo long hay que añadir el carácter L al final del número  
long l = 222222222L;
```

- Java genera un error cuando se establece un valor superior al soportado por el tipo.

```
// declaración correcta  
int j = 2147483647;
```

```
// error porque las variables de tipo int no puede almacenar un número de esa magnitud  
// int k = 2147483648;
```

```
// long sí admite valores de esta magnitud  
long l = 2147483648L;
```

- Cuando una variable sobrepasa el máximo de los valores soportados por un tipo, se le asigna el mínimo valor soportado.

```
int j = 2147483647;
```

```
// el valor de j es incrementado  
j = j + 1;
```

```
// el valor de j es ahora -2147483648  
System.out.println(j);
```

- double es el tipo por defecto para valores flotantes y es la recomendada para cálculos científicos o financieros donde se requiere más precisión.
- Al igual que con las variables de tipo long, es necesario utilizar la letra *f* o *F* al final del valor cuando se declara una variable de tipo float.

```
double m = 67.0;
```

```
// error porque las variables de tipo float terminan con la letra f o F (una de las dos)
```

```
// float n = 4.0;;
// declaración correcta de una variable de tipo float
float o = 4.0f;
```

- Una variable de tipo char almacena un único carácter encerrado entre comillas simples.

```
// declaración correcta de una variable de tipo char
char c = 'A';

// error porque char solamente soporta un carácter
// char cab = 'AB';

// error porque el carácter debe ir encerrado entre comillas simples
// char cab = A;
```

- El concepto de tipo booleano se basa en la lógica matemática. Este tipo de dato puede almacenar solamente dos valores posibles: true o false (sin comillas simples o dobles y en minúsculas).

```
boolean esVerdadero = true;
boolean esFalso = false;
```

Ejercicio: elige el tipo de variable adecuada para almacenar los siguientes datos:

1. El número de goles en un partido de fútbol.
2. El número de personas que viven en el mundo.
3. La letra del DNI.
4. Si una bombilla está encendida o apagada.
5. El precio de una barra de pan.
6. La distancia existente entre planetas.

- Pueden realizar operaciones matemáticas entre diferentes tipos de variables numéricas.

```
byte numeroByte = 5;
short numeroShort = 128;
int numeroInt = 40000;
long numeroLong = 2222222222L;

float numeroFloat = 2.3f;
double numeroDouble = 2.5;

boolean variableBooleana = false;
```

```
// 133
System.out.println(numeroByte + numeroShort);

// 2222262222
System.out.println(numeroInt + numeroLong);

// 7.3
System.out.println(numeroByte + numeroFloat);

// 40002.3
System.out.println(numeroInt + numeroFloat);

// 4.799999952316284
System.out.println(numeroFloat + numeroDouble);

// error porque no se puede sumar un variable numérica con una variable booleana
// System.out.println(numeroByte + variableBooleana);
```

- El operador de asignación compuesto combina un operador numérico con el símbolo igual (=).

```
int b = 4;

// b = b + 5;
b += 5;

// b = b - 2;
b -= 2;

// b = b * 3;
b *= 3;

// b = b / 2;
b /= 2;
```

- Hay dos sintaxis para aumentar (o decrementar) una variable numérica.
  - variable++ significa post-incremento.
  - ++variable significa pre-incremento.

```
int a = 4;

// a = a + 1;
a++;

// 5
System.out.println(a);
```

// 5 (post-incremento: primero se visualiza y luego se incrementa)  
System.out.println(a++);

// 6  
System.out.println(a);

// 7 (pre-incremento: primero se incrementa y luego se visualiza)  
System.out.println(++a);

// 7  
System.out.println(a);

## Conversión de tipos

- Cuando se asigna el valor de un tipo a otro distinto, los tipos pueden no ser compatibles entre sí.
  - Si los tipos de datos son compatibles, entonces se realiza una conversión automática.
  - Si los tipos de datos no son compatibles, entonces debe realizarse un casting o conversión explícita.
- Dos tipos son compatibles cuando se asigna el valor de un tipo de datos más pequeño a un tipo de datos más grande, teniendo en cuenta que: byte < short < int < long < float < double.

```
int i = 100;  
  
// long > int (conversión automática)  
long l = i;  
  
// float > long (conversión automática)  
float f = l;  
  
// 100  
System.out.println(i);  
  
// 100  
System.out.println(l);  
  
// 100.0  
System.out.println(f);
```

- El casting o conversión de tipo explícito se realiza cuando se asigna un valor de un tipo de dato más grande a un tipo de dato más pequeño.

```
double d = 100;  
  
// short < double (casting)  
short s = (short) d;  
  
// byte < double (casting)  
byte b = (byte) d;  
  
// 100.0  
System.out.println(d);  
  
// 100  
System.out.println(s);  
  
// 100  
System.out.println(b);
```

- Es necesario también utilizar el casting para convertir una variable de tipo char a tipo numérico, y viceversa (considerando su correspondencia ASCII).

```
short n1 = 88;  
char c1 = (char) n1;  
  
// X  
System.out.println(c1);
```

```
char c2 = 'a';  
short n2 = (short) c2;  
  
// 97  
System.out.println(n2);
```

Tu carrera digital ~

# Módulo 3

## Java básico

Control del flujo



# Control de flujo

- ◆ Operadores de comparación
- ◆ Condicionales
- ◆ Visibilidad de las variables
- ◆ Operador lógico de negación
- ◆ Operador ternario
- ◆ Introducción de datos por consola
- ◆ Bucles
- ◆ Manejo de excepciones
- ◆ Métodos

## Operadores de comparación

- ◆ Los operadores de comparación evalúan dos variables y producen un valor booleano: true si la evaluación es correcta o false si no lo es.
- ◆ A continuación se listan los operadores de comparación más importantes.

Operador de comparación	Significado
==	Igual que
<	Menor que
<=	Menor o igual que
>	Mayor
>=	Mayor o igual que
!=	Distinto que

```
boolean evaluacion = 5 > 6;
```

```
// false  
System.out.println(evaluacion);
```

```
// true  
System.out.println(5 == 5);
```

```
// false  
System.out.println(5 != 5);
```

```
// true  
System.out.println(5 >= 5);
```

Ejercicio: probar si las siguientes evaluaciones devuelven true o false.

4.3 >= 4

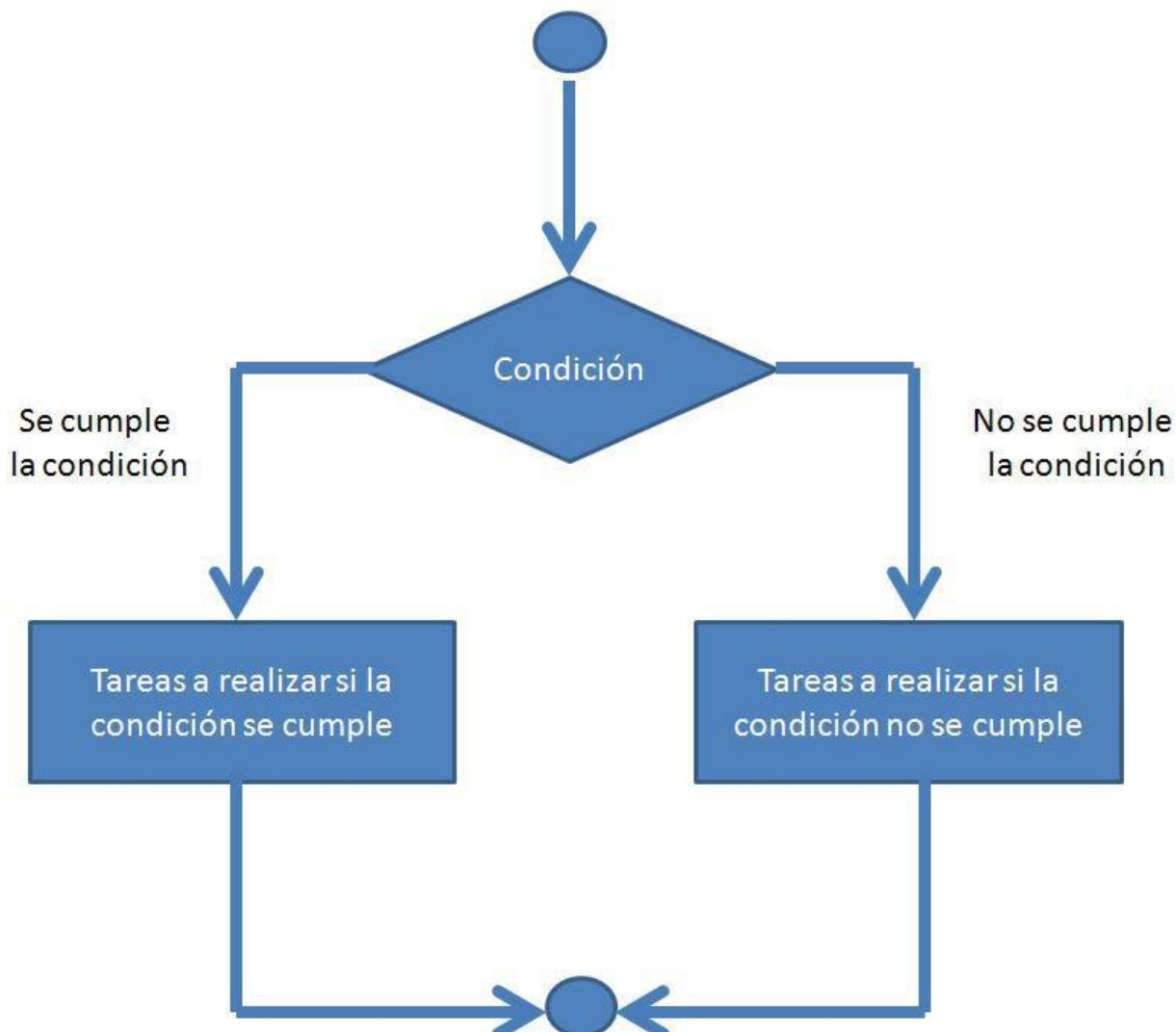
1 == 2

4 < 4

2 != 5

## Condicionales

- Los condicionales permiten ejecutar un conjunto de instrucciones en función del valor de una condición.
- La evaluación de la condición sólo puede arrojar un resultado de true o false.



- Los condicionales se implementan en Java con if. Su sintaxis es:
  - if.
  - Evaluación rodeada de paréntesis.

- Apertura de llaves (dentro de las mismas se encuentra el conjunto de líneas de código que se ejecutarán si se cumple la condición).

```
int edad = 18;  
  
// si edad es igual a 18  
if (edad == 18) {  
    System.out.println("Tengo 18 años");  
}  
  
System.out.println("Programa finalizado");
```

- ◆ Las llaves del condicional pueden omitirse si en el interior del condicional solamente existe una línea de código (aunque es recomendable utilizar siempre llaves).

```
int edad = 18;  
  
// si edad es igual a 18  
if (edad == 18)  
    System.out.println("Tengo 18 años");  
  
System.out.println("Programa finalizado");
```

Ejercicio: escribe un programa con cuatro variables de tipo entero (a, b, c y d) y un condicional que imprima por pantalla si la suma de a y b es mayor que la suma de c y d.

Ejercicio: escribe un programa que almacene tres ángulos de un triángulo en variables de tipo entero (angulo1, angulo2 y angulo3). Crea un condicional que compruebe si esos tres ángulos juntos pueden formar un triángulo (los ángulos de un triángulo suman siempre 180 grados).

Ejercicio: escribe un programa con una variable de tipo entero (a) y un condicional que evalúe si el entero es par o impar utilizando el operador %.

- ◆ Con una secuencia de instrucciones if-else se puede ejecutar un bloque de código si se cumple una condición y otro bloque de código distinto si no se cumple.

```
int edad = 18;  
  
// si edad es igual o mayor que 18  
if (edad >= 18) {  
    System.out.println("Soy mayor de edad");  
}  
// si no se cumple la condición anterior  
else {
```

```
System.out.println("No soy mayor de edad");
}

System.out.println("Programa finalizado");
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int i = 26;

if(i == 25) {
    System.out.println("a");
}
else {
    System.out.println("b");
}
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int i = 25;

if(i == 25) {
    System.out.println("a");
}
if(i == 24) {
    System.out.println("b");
}
else {
    System.out.println("c");
}
```

- Con una secuencia de instrucciones if-else if pueden evaluarse un conjunto de condiciones de forma ordenada. Si se cumple una de ellas, entonces el bloque de código asociado es ejecutado y las siguientes condiciones no son evaluadas.

```
int edad = 18;

// si edad es mayor que 18
if (edad > 18) {
    System.out.println("Soy mayor de edad");
}
// si edad es igual a 18
else if (edad == 18) {
    System.out.println("Tengo 18 años");
```

```
}
```

// si edad es menor que 0

```
else if (edad < 0) {
```

    System.out.println("No he nacido");

```
}
```

System.out.println("Programa finalizado");

- ◆ A la secuencia de instrucciones if-else if se le puede añadir un else final cuyo bloque de código se ejecutará si ninguna de las condiciones anteriores se ha cumplido.

```
int edad = 18;
```

// si edad es mayor o igual que 18

```
if (edad >= 18) {
```

    System.out.println("Soy mayor de edad");

```
}
```

// si edad es menor que 0

```
else if (edad < 0) {
```

    System.out.println("No he nacido");

```
}
```

// si no se cumple ninguna de las condiciones anteriores

```
else {
```

    System.out.println("No soy mayor de edad");

```
}
```

System.out.println("Programa finalizado");

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int i = 24;
```

```
if(i == 25) {
```

    System.out.println("a");

```
}
```

```
else if(i == 24) {
```

    System.out.println("b");

```
}
```

```
else {
```

    System.out.println("c");

```
}
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```

int i = 25;

if(i == 25) {
    System.out.println("a");
}
else if(i == 24) {
    System.out.println("b");
}
else {
    System.out.println("c");
}
if(i == 24) {
    System.out.println("d");
}
else {
    System.out.println("e");
}
if(i == 22) {
    System.out.println("f");
}
else if(i == 25) {
    System.out.println("g");
}
else {
    System.out.println("h");
}
    
```

- Cuando se pretende evaluar varias condiciones al mismo tiempo es necesario utilizar los operadores lógicos.

<b>Operador lógico</b>	<b>Descripción</b>	<b>Ejemplo</b>	<b>Descripción del ejemplo</b>
&&	AND	( (a == 5) && (b == 2) )	La evaluación se cumple si a = 5 y b = 2
	OR	( (a == 5)    (b == 2) )	La evaluación se cumple si a = 5 o b = 2

```

int edad = 18;

// si edad es mayor o igual que 18, o menor que 0
if ((edad >= 18) || (edad < 0)) {
    System.out.println("Soy mayor de edad o no he nacido");
}
// si no se cumple la condición anteriorz
else {
    
```

```
System.out.println("No soy mayor de edad");
}
```

```
int edad = 18;

// si edad es menor que 18, y mayor o igual que 0
if ((edad < 18) && (edad >= 0)) {
    System.out.println("No soy mayor de edad");
}
// si no se cumple la condición anterior
else {
    System.out.println("Soy mayor de edad o no he nacido");
}
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int i = 25;

if((i < 25) && (i >= 25)) {
    System.out.println("a");
}
if((i < 25) || (i >= 25)) {
    System.out.println("b");
}
```

Ejercicio: escribe un programa que dado tres números imprima por pantalla cuál es el mayor.

## Visibilidad de las variables

- Una variable puede ser declarada y no inicializada con un valor.

```
// declaración de variables que no son inicializadas
int a;
int b;

// error porque antes de utilizar la variable es necesario que tome algún valor
// System.out.println(a);

b = 5;
// correcto porque a la variable b se le ha asignado previamente un valor
System.out.println(b);
```

- Las variables pueden ser utilizadas dentro del bloque en el que fueron declaradas.
  - Un bloque es un conjunto de líneas que se encuentra rodeado de llaves {}.
  - Los bloques de código definen la visibilidad o ámbito de las variables (su alcance, es decir, dónde pueden ser utilizadas).

```
public static void main(String[] args) {  
  
    // la variable a es declarada dentro del bloque del método main  
    int a = 5;  
  
    // el bloque if se encuentra dentro del bloque del método main  
    if (a == 5) {  
  
        // la variable a puede ser utilizada dentro del bloque if  
        System.out.println(a);  
    }  
}
```

- Una variable declarada dentro de un bloque no puede ser utilizada fuera del mismo porque su ámbito pertenece al bloque donde fue declarada.

```
public static void main(String[] args) {  
  
    int a = 5;  
  
    if (a == 5) {  
  
        // la variable b es declarada dentro del bloque if  
        int b = 5;  
  
        // la variable b puede ser utilizada dentro del bloque if  
        System.out.println(b);  
    }  
  
    // ERROR --> la variable b no puede ser utilizada fuera del bloque if  
    // System.out.println(b);  
}
```

- Las variables declaradas sin valor e inicializadas dentro un bloque pueden generar errores posteriormente si no se garantiza que ese bloque de inicialización se ejecuta en todos los casos.

```
public static void main(String[] args) {  
  
    // la variable a es declarada dentro del bloque del método main
```

```
int a = 5;

// la variable b es declarada sin ningún valor dentro del bloque del método main
int b;

if (a == 5) {
    // la variable b toma el valor 3 (pero solamente cuando a == 5)
    b = 3;
}

// ERROR --> no se puede garantizar que la variable b tenga un valor en este punto del
// código
// System.out.println(b);
}
```

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

    // la variable a es declarada dentro del bloque del método main
    int a = 5;

    // la variable b es declarada sin ningún valor dentro del bloque del método main
    int b;

    // b tomará un valor siempre, independientemente del valor de a
    if (a == 5) {
        b = 3;
    }
    else {
        b = 4;
    }

    // ahora ya no se producirá ningún error porque b tendrá valor siempre en este punto
    // del código
    System.out.println(b);
}
```

- No se pueden declarar dos variables con el mismo nombre, excepto si son declaradas en distintos ámbitos.

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

    int a = 5;

    if (a == 5) {

        // la variable b es declarada dentro del bloque if y desaparece cuando termina este
        // bloque
    }
}
```

```
int b = 5;
System.out.println(b);
}

// esta variable b es distinta a la variable b anterior (que no existe en este punto del
// código)
int b = 11;
System.out.println(b);
}
```

## Operador lógico de negación

- El operador lógico de negación representa a la operación lógica NOT e invierte el valor de una variable boolean

```
boolean a = true;

// true
System.out.println(a);

a = !a;

// false
System.out.println(a);
```

- Este operador también puede utilizarse en los condicionales cuando se pretende evaluar una condición o variable booleana como falsa, en lugar de como verdadera.

```
boolean a = true;

if (a == true) {
    System.out.println("el valor de a es true");
}

if (a == false) {
    System.out.println("el valor de a es false");
}

if (a != true) {
    System.out.println("el valor de a es false");
}

if (a != false) {
    System.out.println("el valor de a es true");
}

// si a == true
```

```
if (a) {  
    System.out.println("el valor de a es true");  
}  
  
// si a == false  
if (!a) {  
    System.out.println("el valor de a es false");  
}
```

## Operador ternario

- Es muy habitual en programación asignar un valor u otro en función de una determinada condición.

```
int a = 3;  
int b;  
if (a == 3) {  
    b = 2;  
}  
else {  
    b = 4;  
}
```

- El operador ternario (?) puede utilizarse para simplificar el código anterior. Posee la siguiente sintaxis:
  - Condición.
  - Operador ternario ?
  - Valor que se asigna si se cumple la condición.
  - Carácter dos puntos :
  - Valor que se asigna si no se cumple la condición.

```
int a = 3;  
int b = (a == 3) ? 2 : 4;
```

## Introducción de datos por consola

- La clase *Scanner* de Java 5 facilita la lectura de datos en los programas.
- *Scanner* se encuentra dentro del paquete *java.util* y permite obtener la entrada de tipos primitivos de datos (*int*, *double*, *short*, ...) y también *String*.
- El funcionamiento de *Scanner* para leer datos introducidos por el usuario a través de la consola es el siguiente:

1. Crear una instancia de la clase *Scanner*, pasando al constructor el objeto *System.in*, que representa el flujo de entrada estándar.
2. Para leer valores numéricos de un determinado tipo de datos XYZ, la función que se utilizará es *nextXYZ()*. Por ejemplo, para leer un valor de tipo *int*, se utiliza el método *nextInt()*. Para leer strings se utiliza *next()*
3. Cerrar la instancia de la clase *Scanner* mediante el método *close()*

```
// crear una instancia de la clase Scanner pasando System.in al constructor  
Scanner keyboard = new Scanner(System.in);  
  
// el programa se detiene en este punto hasta que el usuario introduza un número y  
pulse ENTER  
int numero = keyboard.nextInt();  
  
// cerrar la instancia de la clase Scanner  
keyboard.close();
```

- ◆ La introducción de un tipo de dato no esperado generará una excepción de tipo *InputMismatchException*.

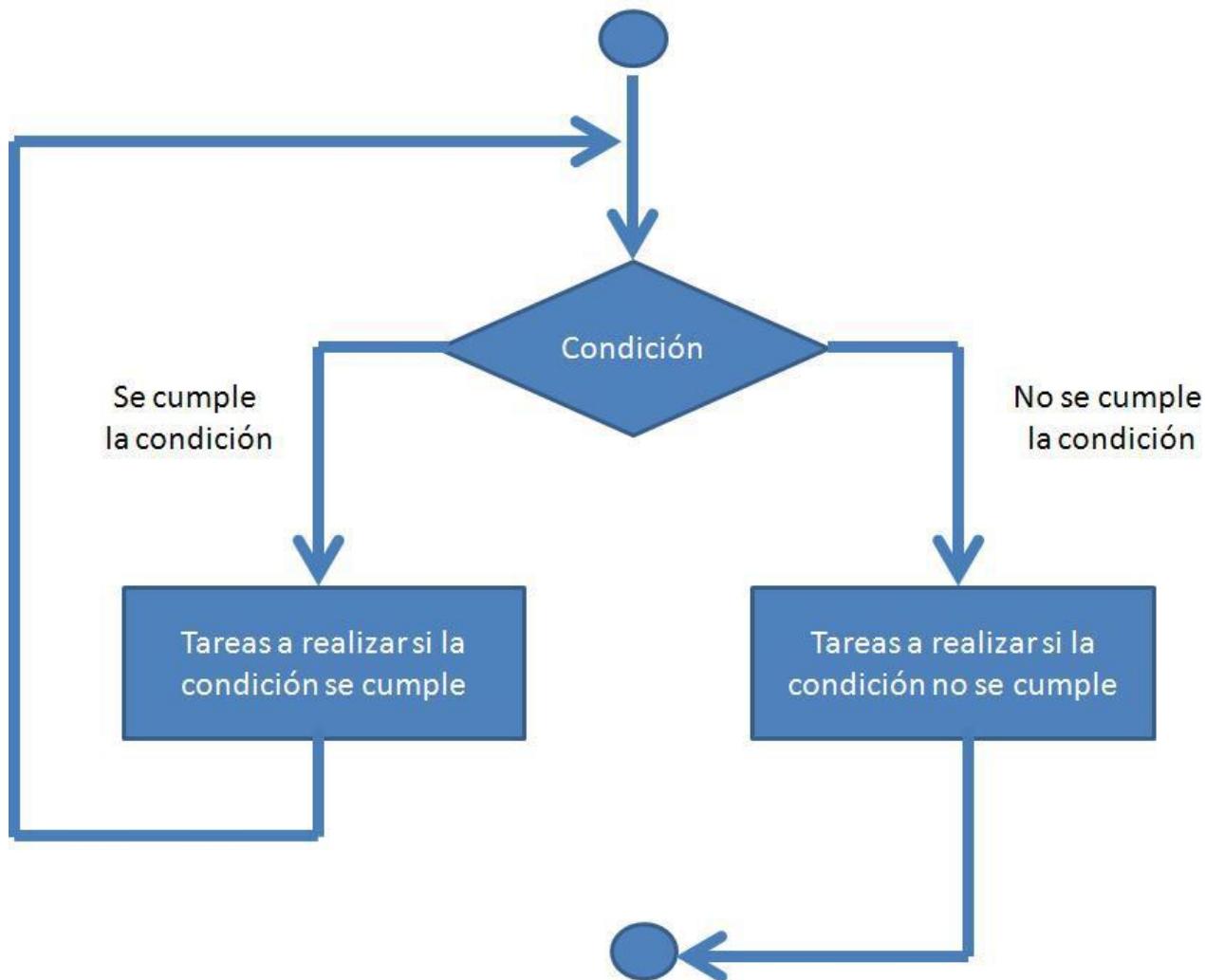
Ejercicio proyecto (Main1): implementa un programa que solicite al usuario que introduzca un número por pantalla y en función del número introducido se muestre el siguiente mensaje para cada número:

- 1 --> Gestores
- 2 --> Clientes
- 3 --> Transferencias
- 4 --> Mensajes
- 5 --> Préstamos
- 6 -> Salir

Para el resto de números no se mostrará ningún mensaje por pantalla.

## Bucles

- ◆ Un bucle es una sentencia que se ejecuta repetidas veces (iteraciones) hasta que la condición asignada a dicho bucle deja de cumplirse.



- Los bucles se implementan en Java con while o for.
- Los bucles con while tienen la siguiente sintaxis:
  - while.
  - Evaluación rodeada con paréntesis.
  - Apertura de llaves (dentro de las mismas se encuentra el conjunto de líneas que se ejecutarán hasta que deje de cumplirse la condición).

```

int edad = 1;

// mientras que edad sea menor que 18
while (edad < 18) {
    System.out.printf("%d", edad).println();
    edad++;
}

System.out.printf("Programa finalizado --> edad: %d", edad).println();
  
```

- El bucle while no terminará nunca cuando dentro del mismo no se produzca una modificación de la variable evaluada. El bucle infinito puede detenerse en Eclipse pulsando el botón "Terminate" (cuadrado rojo).

```
int edad = 1;  
  
// mientras que edad sea menor que 18  
while (edad < 18) {  
    System.out.printf("%d", edad).println();  
}  
  
System.out.printf("Programa finalizado --> edad: %d", edad).println();
```

- También se puede utilizar la sintaxis en do-while. En este caso el código contenido en el bucle se ejecutará al menos una vez.
  - Es obligatorio añadir punto y coma después de la condición del while.

```
int edad = 25;  
  
do {  
    System.out.printf("%d", edad).println();  
    edad++;  
}while (edad < 18); // obligatorio el punto y coma
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int i = 25;  
  
while(i != 0) {  
    i--;  
}  
  
System.out.println(i);
```

Ejercicio proyecto (Main2): reutiliza el programa desarrollado anteriormente para solicitar continuamente un número al usuario hasta que éste sea 6. Ejercicio proyecto (Main3): reutiliza el programa desarrollado anteriormente para mostrar el siguiente menú antes de solicitar el número por pantalla al usuario:

```
---  
1. Gestores  
2. Clientes
```

3. Transferencias

4. Mensajes

5. Préstamos

6. Salir

Introduzca un número:

- ◆ Los bucles con for son más complejos que los bucles while. Su sintaxis está dividida en tres partes:
  - for
  - Tres expresiones rodeadas por paréntesis y separadas por punto y coma:
    - Inicialización: en esta expresión se inicializa una variable (generalmente denominada i) en la primera iteración y solamente accesible dentro del bucle.
    - Condición de evaluación: condición que se evalúa en cada iteración y que debe devolver un valor booleano. Cuando la condición devuelve false, el bucle termina.
    - Condición de incremento/decremento: es una expresión que incrementa o decrementa en cada iteración la variable inicializada.

```
int edad = 1;  
  
// comenzando con i=edad, mientras que i sea menor que 18 y con un incremento de i  
// de uno en uno  
for (int i=edad; i<18; i++) {  
    System.out.printf("%d", i).println();  
}  
  
System.out.printf("Programa finalizado --> edad: %d", edad).println();
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
for(int i=0; i<100; i--) {  
    System.out.println("Hola");  
}  
System.out.println("Adiós");
```

- ◆ Un bucle puede contener internamente condicionales (y viceversa).

```
for(int i=0; i<100; i++) {  
    if (i % 2 == 0) {  
        System.out.printf("%d es un número par", i).println();  
    }  
}
```

Ejercicio: escribe un programa que imprima toda la tabla de multiplicar del 5 (desde 0 hasta 10).

Ejercicio: escribe un programa que imprima todas las tablas de multiplicar del 1 al 9 (desde 0 hasta 10).

Ejercicio: escribe un programa que dado un número, sume dicho número con todos los anteriores. Por ejemplo, para el número 5 el resultado debería ser 15 ( $5 + 4 + 3 + 2 + 1$ ).

Ejercicio: escribe un programa que dado un número, calcule si es primo o no. Un número primo es un número natural mayor que 1 que tiene únicamente dos divisores distintos: él mismo y el 1.

- ♦ Un bucle (for o while) puede finalizar de forma abrupta con una instrucción break.

```
int a = 3;  
  
// bucle infinito  
while(true) {  
  
    a++;  
  
    if (a == 6000) {  
        // finaliza el while  
        break;  
    }  
}
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int a = 3;  
  
while(true) {  
  
    while(true) {  
  
        a++;  
  
        if (a == 6000) {  
            break;  
        }  
    }  
}
```

Ejercicio: ¿cuál es el resultado de ejecutar el siguiente código?

```
int a = 3;  
  
while(true) {  
  
    while(true) {  
  
        a++;  
  
        if (a == 6000) {  
            break;  
        }  
    }  
  
    break;  
}
```

## Manejo de excepciones

- Una vez que el código Java es sintácticamente válido, el compilador puede convertir el código fuente a bytecode para que pueda ser interpretado.
- Sin embargo, durante la ejecución del programa pueden producirse errores. Java arroja una excepción si esto sucede.

```
// se generación una excepción porque en la sexta iteración del bucle se realiza la  
operación matemática 5/0 (infinito) y ese valor no puede ser almacenado en una  
variable de tipo int. Se arroja una excepción de tipo ArithmeticException y el programa  
se detiene en ese punto  
for(int i = -5; i < 5; i++) {  
    int c = 5 / i;  
    System.out.println(c);  
}  
  
/*  
Exception in thread "main" java.lang.ArithmeticsException: / by zero  
at Main.main(Main.java:9)  
*/
```

- Las excepciones se pueden capturar mediante un bloque de código try-catch.
  - El try envuelve entre llaves el código que es susceptible de producir errores.
  - El catch permite capturar excepciones específicas (como la excepción ArithmeticException) y/o excepciones genéricas (excepción Exception). El código encerrado entre las llaves se ejecuta cuando se produce la excepción establecida entre los paréntesis del catch.

- La ejecución del programa no se detiene tras capturar la excepción y continúa después del bloque try-catch, a diferencia del código anterior sin el try-catch donde el programa se detenía en la línea de código donde se generaba la excepción.

```
// trata de ejecutar el código que se encuentra entre las llaves
try {
    for(int i = -5; i < 5; i++) {
        int c = 5 / i;
        System.out.println(c);
    }
}
// captura la excepción de tipo ArithmeticException
catch (ArithmaticException e) {
    System.out.println("Excepción aritmética");
}
// captura cualquier otra excepción que pueda producirse
catch (Exception e) {
    System.out.println("Otro error desconocido");
}

System.out.println("Programa finalizado");
```

```
// trata de ejecutar el código que se encuentra entre las llaves
try {
    for(int i = -5; i<5; i++) {
        int c = 5 / i;
        System.out.println(c);
    }
}
// captura todas las excepciones que puedan producirse
catch (Exception e) {
    System.out.println("Otro tipo de error desconocido");
}
```

- Puede capturarse con catch todas las excepciones que se quieran, pero si se incluye la excepción genérica Exception, debe colocarse ésta en el último catch del bloque try-catch.
- También se puede arrojar una excepción de forma explícita mediante throw new, seguido del nombre de la excepción y paréntesis.
  - Arrojar excepciones de forma explícita con throw es habitual en excepciones personalizadas, es decir, aquellas que son creadas por el usuario.

```
// trata de ejecutar el código que se encuentra entre las llaves
try {
    for (int i = -5; i < 5; i++) {
```

```
if (i == 0) {  
    // arroja explícitamente la excepción ArithmeticException, que es capturada por el  
    catch  
    throw new ArithmeticException();  
}  
  
int c = 5 / i;  
System.out.println(c);  
}  
}  
// captura la excepción de tipo ArithmeticException  
catch (ArithmeticException e) {  
    System.out.println("Excepción aritmética");  
}
```

- ♦ Puede añadirse un bloque adicional (finally) al bloque try-catch.
  - ◊ Este bloque finally es ejecutado siempre, con independencia de que se produzca una excepción o no en el bloque del try, e incluso también si se produce una excepción dentro de los bloques de código dentro de los catch.
  - ◊ Normalmente el código del bloque finally se utiliza para liberar recursos (cerrar archivos abiertos, conexiones a bases de datos o servidores, entre otros).

```
// trata de ejecutar el código que se encuentra entre las llaves  
try {  
    for(int i = -5; i < 5; i++) {  
        int c = 5 / i;  
        System.out.println(c);  
    }  
}  
// captura todas las excepciones que puedan producirse  
catch (Exception e) {  
  
    // arroja la excepción ArithmeticException, que no es capturada  
    System.out.println(5 / 0);  
}  
// este código se ejecuta siempre, se produzca excepción (dentro del bloque try o catch)  
// o no  
finally {  
    System.out.println("Este código se ejecuta siempre");  
}  
  
// esta línea no se ejecuta porque se produce una excepción no capturada dentro del  
// catch  
System.out.println("Programa finalizado");
```

- Es habitual utilizar la instrucción `e.printStackTrace()` dentro del catch para mostrar información sobre el tipo de excepción, la línea donde sucedió la excepción y otra información relevante. Seguidamente el programa continúa ejecutándose a partir del bloque try-catch.

```

try {
    for(int i = -5; i < 5; i++) {
        int c = 5 / i;
        System.out.println(c);
    }
}
// captura cualquier excepción que pueda producirse
catch (Exception e) {
    e.printStackTrace();
}

System.out.println("Programa finalizado");
    
```

- El concepto de error y excepción no es exactamente el mismo, aunque en determinados contextos pueden ser sinónimos:
  - Error: se refiere a errores graves en la máquina virtual de Java que generalmente no pueden ser tratados o capturados en el código. Ejemplos: fallos al enlazar con alguna librería.
  - Excepción: representa errores que no son críticos y por lo tanto pueden ser capturados, tratados y continuar con la ejecución del programa. Ejemplos: dividir entre cero.

## Métodos

- Un método es un conjunto de líneas de código que realiza una tarea específica.
  - Puede aceptar uno o más valores.
  - Puede retornar un valor.
  - En otros lenguajes de programación se le conoce también como función.



- ◆ Para definir y utilizar un método es necesario:
  1. Declararlo
  2. Invocarlo
- ◆ En Java, los métodos se declaran dentro de las clases (encerrado entre sus llaves). Su sintaxis es la siguiente:
  - Ámbito de visibilidad (este concepto se verá posteriormente en programación orientada a objetos): es opcional y sus posibles valores son public, protected o private. El valor por defecto es public.
  - Tipo de método (este concepto se verá posteriormente en programación orientada a objetos): es opcional y su único posible valor es static. Si no se establece el valor de static, entonces el método es no estático (valor por defecto).
  - Tipo de dato que es devuelto por el método: int, float, void (nulo o ninguno) etc.
  - Nombre del método: puede establecerse un nombre arbitrario, aunque no pueden existir dos métodos con el mismo nombre dentro de la misma clase.
  - Variables que acepta el método y tipos: tipo y nombre de la variable. Es necesario separarlas con comas si hay varias variables. No pueden existir dos variables con el mismo nombre.
  - Cuerpo del método: código de programación del cuerpo, encerrado entre llaves: {}
- ◆ Dentro del método puede retornarse un valor o no. En caso de retornarse, se utiliza la expresión return.

```
public class Main {  
  
    /*  
     * Ámbito de visibilidad: public  
     * Tipo de método: static  
     * Tipo de dato que es devuelto por el método: void (ninguno)  
     * Nombre del método: main  
     * Variables que acepta el método y tipos: array de tipo String y de nombre args  
     */  
    public static void main(String[] args) {  
        // el método no retorna ningún valor (void)  
    }  
  
    /*  
     * Ámbito de visibilidad: public  
     * Tipo de método: static  
     * Tipo de dato que es devuelto por el método: int  
     * Nombre del método: suma  
     * Variables que acepta el método y tipos: dos variables de tipo entero con nombre x e y  
     */  
    public static int suma(int x, int y) {  
        // el método retorna el resultado de la operación x + y  
        return x + y;  
    }
```

```
/*
Ámbito de visibilidad: public (valor por defecto)
Tipo de método: no estático (valor por defecto)
Tipo de dato que es devuelto por el método: int
Nombre del método: resta
Variables que acepta el método y tipos: dos enteros de nombre x e y
*/
int resta(int x, int y) {
    // el método retorna el resultado de la operación x - y
    return x - y;
}
```

- Una vez declarados, los métodos estáticos pueden ser invocados desde otras partes del código (por ejemplo, desde el método principal main).
- La invocación a un método estático está compuesto por las siguientes partes:
  - El nombre de la clase donde se ubica el método.
  - El carácter .
  - El nombre del método.
  - Paréntesis de apertura y cierre. En el interior de los paréntesis deben escribirse los valores de las variables que se pasan al método (si hubiera alguna declarada en la definición del método), separadas por coma. Esto se llama "paso de argumentos".
  - El carácter ;

```
/*
Nombre de la clase: Main
Nombre del método: suma
Paso de argumentos: 2 y 3 (valores de tipo entero)
*/
Main.suma(2, 3);
```

- Si el método retorna un valor, entonces a la invocación se le puede asignar una variable donde se almacenará el resultado del método invocado.

```
int a = Main.suma(2, 3);
```

- A continuación se expone un ejemplo completo de declaración de un método llamado suma y dos invocaciones desde el método main.

```
public class Main {
```

```
public static void main(String[] args) {  
  
    // invocación al método suma  
    int a = Main.suma(2, 3);  
  
    // imprime 5  
    System.out.println(a);  
  
    // imprime 5  
    System.out.println(Main.suma(1, 4));  
}  
  
static int suma(int x, int y) {  
    return x + y;  
}
```

- ◆ Desde la invocación es importante que los valores suministrados sean coincidentes en número y en tipo con los declarados por el método (aunque la conversión de tipos automático o el casting están permitidos).

```
// error porque el método suma devuelve un valor de tipo entero y se está asignando  
// ese valor a una variable de tipo boolean  
// boolean d = Main.suma(6, 2);
```

```
// error porque el método suma espera dos variables de tipo entero y se está pasando  
// una variable de tipo boolean y otra de tipo entero  
// int e = Main.suma(true, 2);
```

```
// error porque el método suma espera dos variables como entrada y no tres  
// int f = Main.suma(1, 2, 3);
```

- ◆ Java proporciona gran cantidad de clases con métodos que realizan diversas funcionalidades. Por ejemplo, la clase Math declara métodos para realizar cálculos y operaciones matemáticas.

```
// el método random de la clase Math retorna un valor aleatorio entre 0 y 1 y no requiere de argumentos
```

```
Math.random();
```

```
// el método min de la clase Math retorna el valor mínimo entre dos números que son pasados por argumentos
```

```
Math.min(1,2);
```

// el método max de la clase Math retorna el valor máximo entre dos números que son pasados por argumentos

```
Math.max(1,2);
```

Ejercicio proyecto (Main4): reutiliza el programa desarrollado para mostrar los mensajes (Gestores, Clientes, Transferencias, Mensajes, Préstamos) en métodos (un método por cada mensaje).

Tu carrera digital ~

# Módulo 3

## Java básico

Orientación a objetos



# Orientación a objetos

- ◆ [Introducción](#)
- ◆ [Clases y objetos](#)
- ◆ [Constructores y objeto this](#)
- ◆ [Encapsulación](#)
- ◆ [Métodos y atributos estáticos](#)
- ◆ [Sobrecarga](#)
- ◆ [Herencia](#)
  - [Constructores](#)
  - [Métodos](#)
  - [Objetos](#)
  - [Sobrecarga y reescritura](#)
  - [Herencia múltiple](#)
  - [Librerías de Java](#)
- ◆ [Polimorfismo](#)
- ◆ [Clases abstractas](#)
- ◆ [Interfaces](#)
- ◆ [Paquetes](#)
- ◆ [Referencias a objetos](#)
- ◆ [Convenciones en los nombres](#)
  - [Paquete](#)
  - [Clases e interfaces](#)
- ◆ [Constantes](#)

## Introducción

- ◆ La programación orientada a objetos (POO) es un paradigma que trata de estructurar el código en base a la creación de objetos que realizan acciones de forma colaborativa.
- ◆ Es un paradigma que puede aplicarse a casi cualquier lenguaje de programación (Java es uno de ellos).
- ◆ Algunas de las ventajas que ofrece la POO son:
  - Permite desarrollar programas más rápidamente.
  - Promueve la reutilización del código.
  - Es fácil de mantener.
  - Facilita el trabajo en equipo.
  - Permite descomponer problemas complejos reales en unidades computacionales simples, facilitando la abstracción.
- ◆ Un objeto en el POO es una representación computacional de un objeto del mundo real: reloj, avión, coche, etc.

- Un objeto en el POO es una representación computacional de un objeto del mundo real: reloj, avión, coche, etc.
  - ◆ Un objeto del mundo real posee características comunes con otros objetos del mismo tipo.
  - ◆ Por ejemplo, existen muchos tipos de coches, pero todos tienen características comunes: color, ruedas, asientos, motor, marca, modelo, etc.
  - ◆ Un objeto puede descomponerse en dos tipos de abstracciones: una abstracción de datos y una abstracción funcional.

<b>Abstracción de datos</b>	<b>Abstracción funcional</b>
velocidad	parar
caballos	acelerar
asientos	desacelerar
orientación de las ruedas	girar a la izquierda
	girar a la derecha

## Clases y objetos

- La abstracción de datos representa el estado de un objeto y son definidas en programación con atributos.
- La abstracción funcional representa el comportamiento de un objeto y son definidas en programación con métodos.

### Abstracción de datos



Estado

### Abstracción funcional



Comportamiento

### Atributos

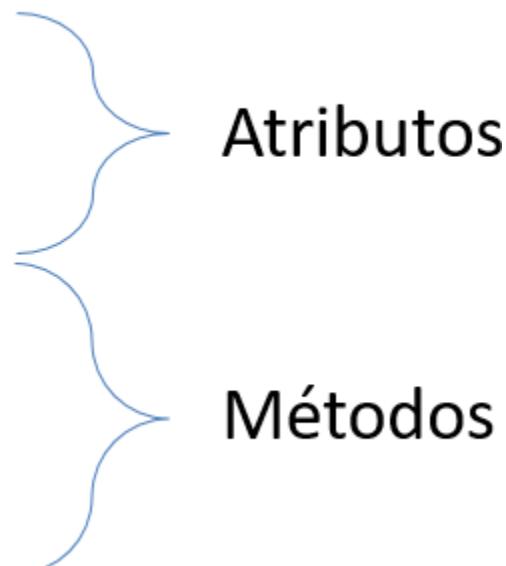


### Métodos

- La naturaleza de un objeto se define en una clase. La clase puede considerarse como un prototipo, plantilla o molde de objetos.

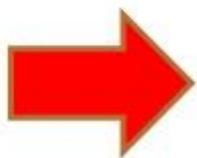
- En la clase es donde se definen los atributos (estado) y métodos (comportamiento) que tendrán los objetos.

Clase Coche	
velocidad	
caballos	
asientos	
orientación de las ruedas	
parar	
acelerar	
desacelerar	
girar a la izquierda	
girar a la derecha	



- En Java se definen las clases en archivos .java (una clase por archivo).
- Para definir la clase dentro del archivo se utiliza la partícula class seguido del nombre de la clase, que debe ser el mismo que el nombre del archivo.
- Entre las llaves de la clase se definen los atributos sin inicializar (en forma de variables) y los métodos.

Fichero Coche.java



```
public class Coche {
    // atributos
    float velocidad;
    float caballos;
    int asientos;
    int orientacionRuedas;

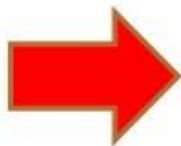
    // métodos
    void parar() { }
    void acelerar() { }
    void desacelerar() { }
    void girarIzquierda() { }
    void girarDerecha() { }
}
```

- Por tanto, una clase es una plantilla para definir objetos.



- A partir de la clase pueden crearse objetos como si fueran variables mediante la partícula new seguido del nombre de la clase y paréntesis de apertura y cierre.

Fichero Main.java



```
public class Main {
    public static void main(String[] args) {
        Coche audi = new Coche();
        Coche nissan = new Coche();
        Coche volvo = new Coche();
    }
}
```

Ejercicio: Crea la clase Estudiante con los siguientes atributos y métodos: Atributo id  
Atributo edad Atributo telefono Atributo numeroDeNotas Atributo sumaDeNotas Atributo  
notaMedia Método mostrarInfo Método agregarNuevaNota

Clase Estudiante	
	id
	edad
	telefono
	numeroDeNotas
	sumaDeNotas
	notaMedia
	mostrarInfo
	agregarNuevaNota

## Constructores y objeto this

- Al inicializar un objeto se crea una instancia de la clase con sus correspondientes atributos y métodos.

- Por defecto, el valor de los atributos para cada objeto es 0 para números enteros, 0.0 para números flotantes y false para booleanos.

Objeto audi	Objeto volvo	Objeto nissan
velocidad = 0.0 caballos = 0.0 orientación de las ruedas = 0 marcha = 0	velocidad = 0.0 caballos = 0.0 orientación de las ruedas = 0 marcha = 0	velocidad = 0.0 caballos = 0.0 orientación de las ruedas = 0 marcha = 0
parar acelerar desacelerar girar a la izquierda girar a la derecha	parar acelerar desacelerar girar a la izquierda girar a la derecha	parar acelerar desacelerar girar a la izquierda girar a la derecha

- Estos valores por defecto de los atributos no suelen representar el estado real de los objetos.
- Cada objeto debería estar definido por distintos valores para cada atributo.

Objeto audi	Objeto volvo	Objeto nissan
velocidad = 0.0 km/h caballos = 350.0 CV asientos = 2 orientación de las ruedas = 0º	velocidad = 0.0 km/h caballos = 90.0 CV asientos = 4 orientación de las ruedas = 0º	velocidad = 12.0 km/h caballos = 75.0 CV asientos = 4 orientación de las ruedas = 9º
parar acelerar desacelerar girar a la izquierda girar a la derecha	parar acelerar desacelerar girar a la izquierda girar a la derecha	parar acelerar desacelerar girar a la izquierda girar a la derecha

- Para inicializar un objeto con unos valores concretos hay que definir un constructor en la clase.
- Un constructor es un método especial de la clase que: Es público y tiene el mismo nombre que el de la clase.
  - No retorna ningún tipo de valor (ni siquiera void).
  - Recibe como argumento el valor de los atributos que se pretenden inicializar.
- La asignación se realiza entre las variables pasadas por argumento y los atributos del objeto, donde this referencia al objeto de la clase actual.

```
public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {  
    this.velocidad = velocidad;  
    this.caballos = caballos;  
    this.asientos = asientos;  
    this.orientacionRuedas = orientacionRuedas;  
}
```

- Una vez definido el constructor, en la creación de objetos mediante new puede suministrarse los valores de los atributos para cada objeto, en el orden en el que son definidos en el constructor.

```
public class Coche {  
  
    public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {  
        this.velocidad = velocidad;  
        this.caballos = caballos;  
        this.asientos = asientos;  
        this.orientacionRuedas = orientacionRuedas;  
    }  
  
    private float velocidad;  
    private float caballos;  
    private int asientos;  
    private int orientacionRuedas;  
  
    void parar() {}  
    void acelerar() {}  
    void desacelerar() {}  
    void girarIzquierda() {}  
    void girarDerecha() {}  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Coche audi = new Coche(0.0f, 350.0f, 2, 0);  
        Coche nissan = new Coche(0.0f, 90.0f, 4, 0);  
        Coche volvo = new Coche(12.0f, 75.0f, 5, 5);  
    }  
}
```

- No es obligatorio que una clase defina un constructor, aunque sí es recomendable que exista al menos uno.
- Un constructor que no recibe parámetros se llama constructor por defecto.
- Un constructor que recibe al menos un parámetro se llama constructor parametrizado.
- Pueden definirse todos los constructores que se quieran.

```

    // constructor por defecto
    public Coche() {
        this.velocidad = 5.0f;
        this.caballos = 0.0f;
        this.asientos = 4;
        this.orientacionRuedas = 0;
    }

    // constructor parametrizado
    public Coche(int asientos) {
        this.velocidad = 5.0f;
        this.caballos = 0.0f;
        this.asientos = asientos;
        this.orientacionRuedas = 0;
    }

    // constructor parametrizado
    public Coche(float velocidad, int caballos) {
        this.velocidad = velocidad;
        this.caballos = caballos;
        this.asientos = 2;
        this.orientacionRuedas = 0;
    }
}

```

- No se puede crear un objeto sin argumentos si se define un único constructor parametrizado en la clase.

```

public class Coche {

    public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {
        this.velocidad = velocidad;
        this.caballos = caballos;
        this.asientos = asientos;
        this.orientacionRuedas = orientacionRuedas;
    }

    private float velocidad;
    private float caballos;
    private int asientos;
    private int orientacionRuedas;

    void parar() { }
    void acelerar() { }
    void desacelerar() { }
    void girarIzquierda() { }
    void girarDerecha() { }
}

```

```

public class Main {
    public static void main(String[] args) {
        Coche audi = new Coche();
    }
}

```

Error

Ejercicio: Crea un constructor por defecto y tres parametrizados (uno que incluya todos los atributos de la clase y asigne sus respectivos valores pasados por argumentos). Crea varias instancias de la clase a partir de los diferentes constructores.

Clase Estudiante
id
edad
telefono
numeroDeNotas
sumaDeNotas
notaMedia
mostrarInfo
agregarNuevaNota

## Encapsulación

- Una vez creado el objeto se puede acceder a los atributos y a los métodos del mismo. La sintaxis es:
  - Nombre del objeto.
  - Punto.
  - Nombre del atributo o del método.

```
Coche audi = new Coche(0.0f, 350.0f, 2, 0);
```

```
// visualización de los atributos del objeto audi
System.out.println(audi.asientos);
System.out.println(audi.velocidad);
```

```
// invocación de los métodos del objeto audi
audi.acelerar();
audi.parar();
```

- Sin embargo, no es habitual acceder directamente a los atributos de un objeto.
  - En el ejemplo anterior del coche, el atributo velocidad no se debería poder modificar directamente. Su valor debería cambiar solamente cuando se invocan a los métodos acelerar, desacelerar y/o parar.
- La encapsulación es la definición de la parte visible de una clase (interfaz pública) y de la parte oculta (privada).
- En general:
  - Los métodos son públicos.

- Los atributos son privados.
- Pueden existir métodos privados.
- Es mala práctica utilizar atributos públicos.
- ◆ Para declarar un atributo o método como público o privado se utilizan los modificadores `public` y `private` respectivamente antes de establecer el tipo.
- ◆ Un atributo o método es público por defecto si no se le agrega el modificador `public` o `private`.

```
// atributos
private float velocidad;
private float caballos;
private int asientos;
private int orientacionRuedas;

// métodos
public void parar() { }
public void acelerar() { }
public void desacelerar() { }
public void girarIzquierda() { }
public void girarDerecha() { }
```

- ◆ Una vez establecidos los atributos como privados (`private`), ya no es posible acceder a ellos directamente desde el objeto.

Error

```
Coche audi = new Coche(0.0f, 350.0f, 2, 0);
```

```
// visualización de los atributos del objeto audi
System.out.println(audi.asientos);
System.out.println(audi.velocidad);

audi.acelerar();
audi.parar();
```

- ◆ A veces puede interesar obtener o modificar los atributos privados de un objeto. Para ello pueden definirse métodos públicos `get` (para obtener) y `set` (para modificar).
  - Un método `get` retorna el valor de un atributo del objeto mediante `this` y `return`.
  - Un método `set` asigna un nuevo valor pasado por argumentos a un atributo del objeto mediante `this`.

```
public float getCaballos() {  
    return caballos;  
}  
  
public void setCaballos(float caballos) {  
    this.caballos = caballos;  
}
```

- Una vez implementados los métodos get y set para un atributo, puede obtener o modificarse el valor del atributo asociado al objeto.

```
Coche audi = new Coche(0.0f, 350.0f, 2, 0);  
  
// se obtiene el valor actual del atributo caballos y se muestra (350.0)  
System.out.println(audi.getCaballos());  
  
// se modifica el valor actual del atributo caballos a 250  
audi.setCaballos(250.0f);  
  
// se obtiene el valor actual del atributo caballos y se muestra (250.0)  
System.out.println(audi.getCaballos());
```

- Los métodos get y set deben declararse siempre como públicos.

Ejercicio: Añade los diferentes modificadores (private y public) a los atributos y métodos de la clase Estudiante. Crea los diferentes métodos get y set para cada uno de los atributos de la clase Estudiante.

### Clase Estudiante

```
id  
edad  
telefono  
numeroDeNotas  
sumaDeNotas  
notaMedia  
  
mostrarInfo  
agregarNuevaNota
```

## Métodos y atributos estáticos

- A veces puede interesar crear un método al que se pueda acceder sin necesidad de crear una instancia de la clase. Son los llamados métodos estáticos.

- Para declarar un atributo o método estático hay que utilizar la partícula *static*.

```
// método que convierte los cv (caballos) a kw (kilowatios)
public static float convertirCvKw(float caballos) {
    return caballos / 1.35f;
}

// método que convierte los kw (kilowatios) a cv (caballos)
public static float convertirKwCv(float kilowatios) {
    return kilowatios * 1.35f;
}
```

- Los métodos estáticos de una clase pueden invocarse desde fuera de la clase a partir de su nombre, sin necesidad de crear un objeto.

```
// invocación de los métodos estáticos
float kw = Coche.convertirCvKw(350.0f);
float cv = Coche.convertirKwCv(kw);
```

```
System.out.println(kw);
System.out.println(cv);
```

- También se puede acceder al método estático desde un objeto, pero esto no es recomendable y aparece un warning o advertencia notificándolo.

```
Coche audi = new Coche(0.0f, 350.0f, 2, 0);

audi.convertirCvKw(350.0f);
```

- Al igual que los métodos estáticos, también pueden declararse atributos estáticos.
- Generalmente este tipo de atributos sí suelen ser declarados como públicos.

```
// atributo estático
public static float factorConversionCvKw = 1.35f;
```

- Un atributo estático público puede ser accedido desde fuera de la clase.
- Un atributo estático público puede ser accedido desde fuera de la clase.
- Cuando el acceso a un miembro (atributo o método) estático de la clase se realiza desde dentro de la propia clase, se puede omitir el nombre de la misma.

```
// atributo estático
public static float factorConversionCvKw = 1.35f;

// método que convierte los cv (caballos) a kw (kilowatios)
public static float convertirCvKw(float caballos) {
    return caballos / factorConversionCvKw;
}

// método que convierte los kw (kilowatios) a cv (caballos)
public static float convertirKwCv(float kilowatios) {
    return kilowatios * factorConversionCvKw;
}
```

- Desde fuera de la clase se puede modificar el valor de un atributo estático si es público.

```
public class Main {

    public static void main(String... args) {
        Coche.factorConversionCvKw = 2.3f;
    }
}
```

- Puede añadirse la partícula final en la declaración del atributo para evitar que un atributo pueda ser modificado.

```
// atributo estático cuyo valor no puede ser cambiado (constante)
public static final float factorConversionCvKw = 1.35f;
```

- Una vez declarado un atributo como final, su valor no puede ser cambiado ni desde dentro de la clase, ni desde fuera. Se convierte en una constante.

Fichero Coche.java



```
public static final float factorConversionCvKw = 1.35f;

public static float convertirCvKw(float caballos) {
    factorConversionCvKw = 2.3f;
    return caballos / factorConversionCvKw;
}
```

Fichero Main.java



```
public class Main {

    public static void main(String... args) {
        Coche.factorConversionCvKw = 2.3f;
    }
}
```

- Dentro de un método estático no puede utilizarse this.

Ejercicio: Crea un método estático público (de nombre crearEstudiante) que devuelva un objeto de tipo Estudiante inicializado con los valores asignados en el constructor por defecto.



## Sobrecarga

- Anteriormente se comprobó que podían existir distintos constructores (con el mismo nombre) si los argumentos eran distintos.
- Igualmente, pueden definirse en una clase dos métodos con el mismo nombre si los argumentos son distintos. A esto se le llama sobrecarga de un método.

```
// método acelerar sin argumentos
public void acelerar() {
    this.velocidad += 5;
}
```

```
// método acelerar con un argumento
public void acelerar(float incrementoVelocidad) {
    this.velocidad += incrementoVelocidad;
}
```

- Dos métodos pueden tener el mismo nombre y el mismo número de argumentos, siempre y cuando los tipos sean distintos.

```
// método desacelerar sin argumentos
public void desacelerar() {
    this.velocidad -= 5;
}

// método desacelerar con un argumento de tipo float
public void desacelerar(float decrementoVelocidad) {
    this.velocidad -= decrementoVelocidad;
}

// método desacelerar con un argumento de tipo booleano
public void desacelerar(boolean marchaAtras) {
    if(marchaAtras) this.velocidad = -5.0f;
    else desacelerar();
}
```

- Los métodos sobrecargados pueden ser accedidos desde fuera de la clase, a través de los objetos.
- El método ejecutado dependerá del tipo de dato pasado por argumentos en la invocación.

```
public class Main {

    public static void main(String[] args) {

        Coche audi = new Coche(0.0f, 350.0f, 2, 0);
        audi.desacelerar();
        audi.desacelerar(2.0f);
        audi.desacelerar(true);
    }
}
```

- También pueden crearse métodos estáticos sobrecargados.

Ejercicio: Implementa tres métodos sobrecargados para agregarNota: Primer método: no recibe parámetros. Segundo método: recibe como parámetro una nota (valor de tipo flotante). Tercer método: recibe como parámetro una nota y una variable booleana para reiniciar a cero los atributos numeroDeNotas, sumaDeNotas y notaMedia. El funcionamiento general del método agregarNota es: Aumentar un valor el atributo numeroDeNotas. Sumar la nueva nota al atributo sumaDeNotas. Obtener el nuevo valor de notaMedia diviendo sumaDeNotas por numeroDeNotas.

## Herencia

- A veces pueden existir clases que comparten características comunes.
- La herencia permite definir una clase a partir de otra relacionada.
- Es un mecanismo para la reutilización de software.

Clase Coche
velocidad
caballos
asientos
orientación de las ruedas
parar
acelerar
desacelerar
girar a la izquierda
girar a la derecha

Clase Avión
velocidad
tren de aterrizaje
asientos
orientación de las ruedas
volar
aterrizar
acelerar
desacelerar
girar a la izquierda
girar a la derecha

Clase Bicicleta
velocidad
cambios
asientos
orientación de las ruedas
parar
acelerar
desacelerar
girar a la izquierda
girar a la derecha

Las clases Coche, Avión y Bicicleta son especializaciones de la clase Vehículo

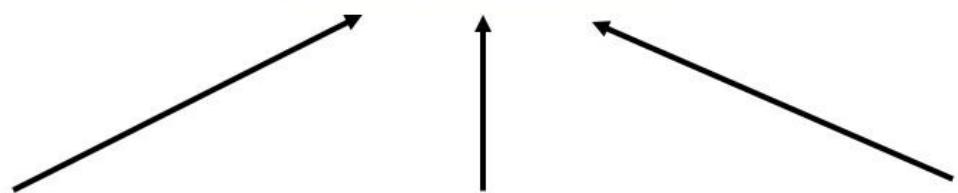
Clase Vehículo
velocidad
asientos
orientación de las ruedas
parar
acelerar
desacelerar
girar a la izquierda
girar a la derecha

La clase Vehículo es una generalización de las clases Coche, Avión y Bicicleta.

Clase Coche
caballos

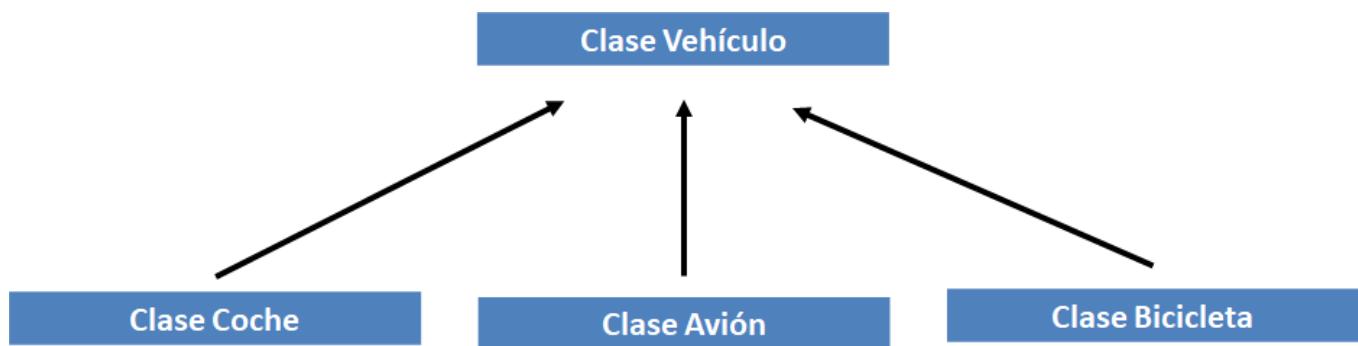
Clase Avión
tren de aterrizaje
volar
aterrizar

Clase Bicicleta
cambios

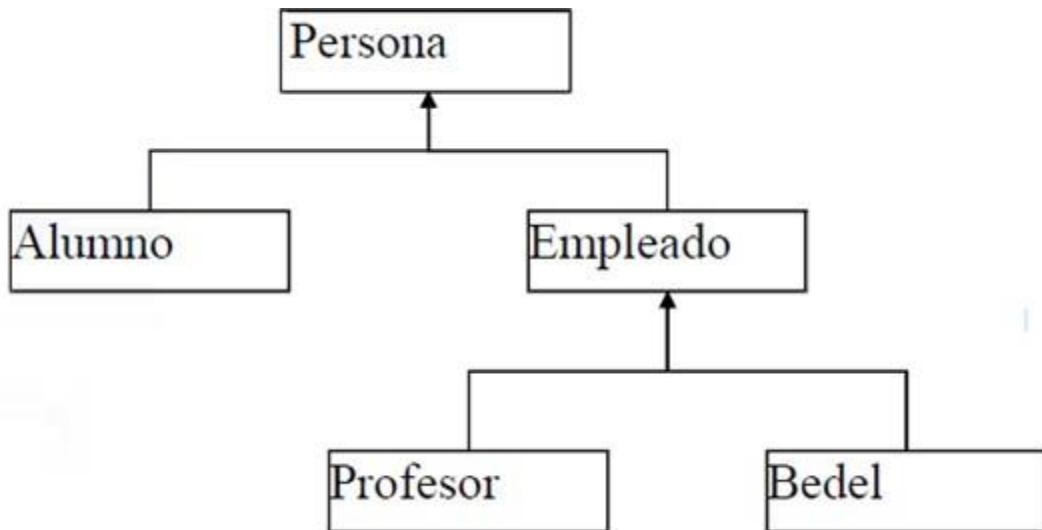


- La terminología utilizada en la herencia es la siguiente:
  - Coche, Avión y Bicicleta heredan los atributos y métodos de la clase Vehículo.
  - Coche, Avión y Bicicleta son subclases, clases derivadas o clases hijas de la clase Vehículo.

- Vehículo es una superclase, clase padre o clase base de las clases Coche, Avión y Bicicleta.



- La sintaxis para declarar clases derivadas en Java es:
- **La sintaxis para declarar clases derivadas en Java es:**



- Los atributos en la superclase no suelen ser declarados como privados (private), sino como protegidos (protected), para que puedan ser accedidos desde las clases hijas.
- Los atributos de la superclase declarados como privados solamente podrán ser accedidos desde dentro de la superclase.

## Fichero Vehiculo.java



```
public class Vehiculo {

    // atributos
    protected float velocidad;
    protected int asientos;
    protected int orientacionRuedas;

    // métodos
    public void parar() { }
    public void acelerar() { }
    public void desacelerar() { }
    public void girarIzquierda() { }
    public void girarDerecha() { }
}
```

- Las clases que heredan de una superclase utilizan la partícula `extends`, seguido del nombre de la clase que heredan. En las clases hijas se declaran los atributos y métodos específicos.

## Fichero Coche.java



```
public class Coche extends Vehiculo {

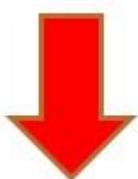
    // atributos específicos del Coche
    private float caballos;

}
```

Clase Coche
velocidad
asientos
orientación de las ruedas
caballos
parar
acelerar
desacelerar
girar a la izquierda
girar a la derecha

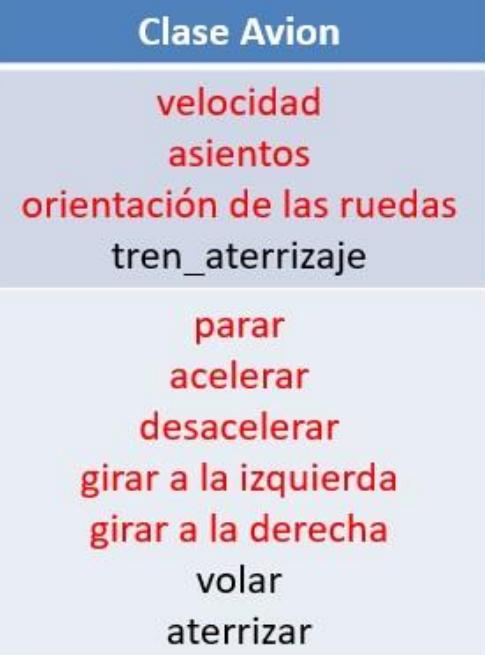
- Una vez declarada una clase hija, ésta hereda todos los atributos y métodos de la superclase.
- Cualquier cambio en el código en la superclase, afectará a la clase hija.

## Fichero Avion.java



```
public class Avion extends Vehiculo {
    // atributos específicos del Avión
    private boolean trenAterrizaje;

    // métodos específicos del avión
    public void volar() { }
    public void aterrizaje() { }
}
```



- Existen diferencias entre los distintos modificadores de visibilidad vistos con anterioridad en función de si la clase hereda (subclase) y también si se encuentra o no en el mismo paquete que la superclase.

-	Sin modificador	private	protected	public
Acceso desde la misma clase	Sí	Sí	Sí	Sí
Acceso desde una subclase del mismo paquete	Sí	No	Sí	Sí
Acceso desde una clase que no es subclase y que se encuentra en el mismo paquete	Sí	No	Sí	Sí
Acceso desde una subclase y que se encuentra en distinto paquete	No	No	Sí	Sí
Acceso desde una clase que no es subclase y que se encuentra en distinto paquete	No	No	No	Sí

## Constructores

- Las clases hijas pueden acceder a los atributos de la superclase mediante this (siempre que los atributos en la superclase hayan sido declarados como protegidos).

```

public class Coche extends Vehiculo {

    // atributos específicos del Coche
    private float caballos;

    public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {
        this.velocidad = velocidad;
        this.asientos = asientos;
        this.orientacionRuedas = orientacionRuedas;

        // inicialización de los atributos específicos del Coche
        this.caballos = caballos;
    }
}

```

- También pueden declararse los atributos de la superclase como privados y acceder a ellos desde las clases hijas mediante métodos get y set públicos o protegidos .
- Sin embargo, es preferible inicializar los atributos de la super clase desde el constructor de la misma.

```

public class Vehiculo {

    // atributos
    protected float velocidad;
    protected int asientos;
    protected int orientacionRuedas;

    public Vehiculo(float velocidad, int asientos, int orientacionRuedas) {
        this.velocidad = velocidad;
        this.asientos = asientos;
        this.orientacionRuedas = orientacionRuedas;
    }
}

```

- Desde el constructor de las clases hijas se invoca al constructor de la superclase mediante el método super, que recibe como argumentos los valores que recibe el constructor de la superclase.
- *super* debe ser la primera línea del constructor de la clase hija.

```

public class Coche extends Vehiculo {

    // atributos específicos del Coche
    private float caballos;

    public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {

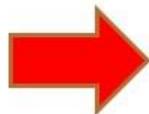
        // invocación al constructor de la clase Vehiculo
        super(velocidad, asientos, orientacionRuedas);

        // inicialización de los atributos específicos del Coche
        this.caballos = caballos;
    }
}

```

- El constructor de la clase hija siempre invoca al constructor por defecto de la clase padre, incluso aunque no se invoque a *super()*.

Fichero Vehiculo.java



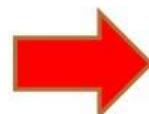
```
public class Vehiculo {
    public Vehiculo() {
        System.out.println("Constructor de la clase padre");
    }
}
```

Fichero Bicicleta.java



```
public class Bicicleta extends Vehiculo {
    public Bicicleta() {
        System.out.println("Constructor de la clase hija");
    }
}
```

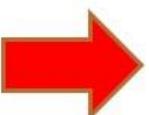
Fichero Main.java



```
public class Main {
    public static void main(String[] args) {
        Bicicleta bicicleta = new Bicicleta();
    }
}
```

- Si no se define ningún constructor parametrizado en la superclase, puede invocarse sin problemas a *super()* desde la clase hija.

Fichero Vehiculo.java



```
public class Vehiculo {
    public Vehiculo() {
        System.out.println("Constructor de la clase padre");
    }
}
```

Fichero Bicicleta.java



```
public class Bicicleta extends Vehiculo {
    public Bicicleta() {
        super();
        System.out.println("Constructor de la clase hija");
    }
}
```

Fichero Main.java



```
public class Main {
    public static void main(String[] args) {
        Bicicleta bicicleta = new Bicicleta();
    }
}
```

- Si existe un constructor parametrizado en la superclase y no existe un constructor por defecto, entonces no se puede invocar al constructor por defecto de la superclase desde la clase hija con *super*.

Fichero Vehiculo.java



```
public class Vehiculo {
    public Vehiculo(float variable) {
        System.out.println("Constructor de la clase padre");
    }
}
```

Error

Fichero Bicicleta.java



```
public class Bicicleta extends Vehiculo {
    public Bicicleta() {
        super();
        System.out.println("Constructor de la clase hija");
    }
}
```

Fichero Main.java



```
public class Main {
    public static void main(String[] args) {
        Bicicleta bicicleta = new Bicicleta();
    }
}
```

## Métodos

- ◆ Con respecto a los métodos de la superclase y su visibilidad, la función de los modificadores es la misma que con los atributos.
  - Método declarado en la superclase como private: solamente accesible desde la propia superclase.
  - Método declarado en la superclase como protected: accesible desde la propia superclase y desde las clases hijas.
  - Método declarado en la superclase como public: accesible desde todas partes.

## Objetos

- ◆ Cuando se crean objetos es preferible instanciarlos a partir de la especialización (Coche, Avión, Bicicleta), antes que de la generalización (Vehículo).

```
// instancia de la clase más genérica
Vehiculo vehiculo = new Vehiculo();
```

```
// instancia de una clase más especializada
Avion avion = new Avion();
```

- ◆ Si existe una mayor especialización es preferible optar por una instancia de la misma.

```
// instancia de una clase aún más especializada
Boeing707 boeing707 = new Boeing707();
```

## Sobrecarga y reescritura

- La reescritura tiene características comunes con la sobrecarga.
- La sobrecarga permite que exista más de una método con el mismo nombre, pero con distintos argumentos.
- Los métodos sobrecargados pueden definirse en la misma clase o en distintas clases de la jerarquía de la herencia.

**Fichero Vehiculo.java**

```
public class Vehiculo {
    protected float velocidad;
    public void parar() { }
}
```

**Fichero Coche.java**

```
public class Coche extends Vehiculo {
    // atributos específicos del Coche
    private float caballos;
    // sobrecarga del método
    public void parar(boolean choque) { }
}
```

**Fichero Main.java**

```
Coche coche = new Coche();
// método de la superclase (Vehiculo)
coche.parar();
// método de la clase hija (Coche)
coche.parar(1, 2);
```

- La reescritura permite que una clase hija pueda reemplazar un atributo (ocultación) o método (redefinición) de la superclase.
- Lo más habitual cuando se produce reescritura es que se reescriba un método.
- Una vez que se reescribe en la clase hija, ya no es posible invocar al método de la superclase desde el objeto.

**Fichero Vehiculo.java**

```
public class Vehiculo {
    protected float velocidad;
    public void parar() { }
}
```

**Fichero Coche.java**

```
public class Coche extends Vehiculo {
    // atributos específicos del Coche
    private float caballos;
    // reescritura del método
    public void parar() { }
}
```

**Fichero Main.java**

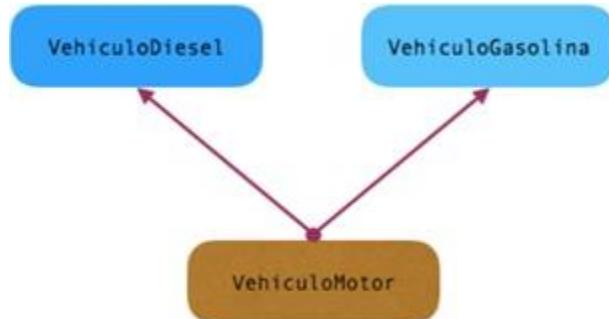
```
Coche coche = new Coche();
// método de la clase hija (Vehiculo)
coche.parar();
```

- Desde la clase hija sí se puede invocar al método reescrito de la superclase utilizando *super*.

```
public class Coche extends Vehiculo {  
  
    // atributos específicos del Coche  
    private float caballos;  
  
    // reescritura del método  
    public void parar() {  
  
        // invocación al método de la superclase  
        super.parar();  
    }  
}
```

## Herencia múltiple

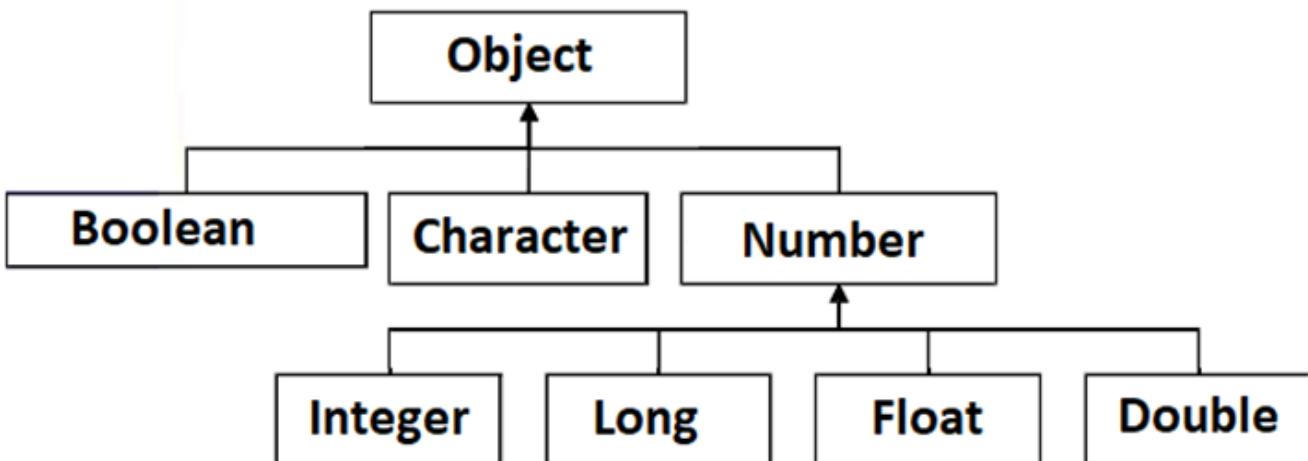
- La herencia múltiple (una clase hereda de varias superclases) no es permitida en Java por tres razones:
  - Ambigüedad en el código.
  - Mayor complejidad.
  - Problemas en la compilación de código.



- Una forma de simular una especie de herencia múltiple en Java es heredando una clase e implementando clases de tipo interfaz.

## Librerías de Java

- Las clases definidas en la librerías de Java también se encuentran agrupadas mediante una jerarquía basada en herencia.
- La mayoría de las clases basadas en tipos tienen como superclase raíz a la clase Object.



Ejercicio: Crea dos nuevas clases (Persona y Profesor) y aplica el concepto de herencia: Persona es la superclase. Profesor y estudiante son clases derivadas. El método mostrarInfo para las clases Estudiante y Profesor debe sobrescribirse para mostrar los valores de los atributos específicos de estas clases.

Clase Persona (superclase)	Clase Estudiante	Clase Profesor
id edad telefono  mostrarInfo	numeroDeNotas sumaDeNotas notaMedia  agregarNuevaNota*3 mostrarInfo+ <i>crearEstudiante</i>	numeroDeEdadesEstudiantes sumaDeEdadesEstudiantes estudiantesEdadMedia  agregarEdadEstudiante mostrarInfo+

## Polimorfismo

- El polimorfismo es la capacidad de un objeto para decidir qué método invocar considerando la clase a la que pertenece. Una invocación de un tipo genérico (clase base) ejecuta la implementación correspondiente del método dependiendo de la clase del objeto especializado.

```

class Vehiculo {
    protected void parar() {
        System.out.println("Método parar del vehículo");
    }
}
  
```

```

public class Coche extends Vehiculo {
    public void parar() {
        System.out.println("Método parar del coche");
    }
}

public class Avion extends Vehiculo {
    public void parar() {
        System.out.println("Método parar del avión");
    }
}

public class Bicicleta extends Vehiculo {
}
  
```

- El método invocado dependerá del tipo de instancia.

```
public static void main(String... args) {  
    Bicicleta bicicleta = new Bicicleta();  
    Coche coche = new Coche();  
    Avion avion = new Avion();  
    Main.invocarParar(bicicleta);  
    Main.invocarParar(coche);  
    Main.invocarParar(avion);  
}  
  
public static void invocarParar(Vehiculo vehiculo) {  
    vehiculo.parar();  
}
```



## Clases abstractas

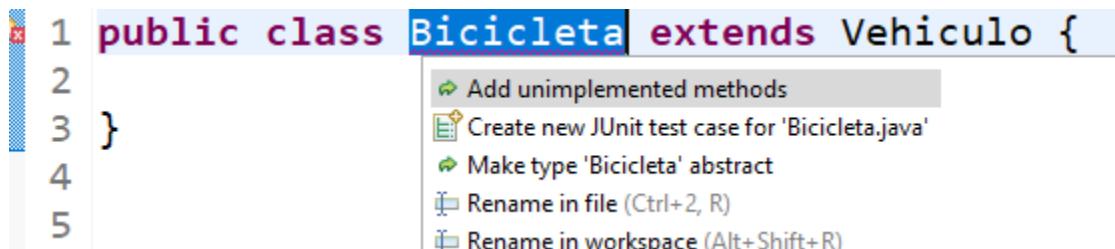
- Las clases abstractas son aquellas que generalmente poseen al menos un método abstracto.
- Un método abstracto es aquel que no tiene implementación (sin código en el cuerpo).
- Las clases abstractas y los métodos abstractos se declaran mediante el modificador abstract.

```
public abstract class Vehiculo {  
    // declaración de método abstracto  
    protected abstract void parar();  
}
```

- Los métodos abstractos no poseen las llaves {} que encierran el cuerpo de su implementación y finalizan siempre con un punto y coma.
- Las clases que heredan clases abstractas están obligadas a implementar el método o métodos abstractos.

```
public class Bicicleta extends Vehiculo {  
}
```

- En Eclipse pueden añadirse los métodos a implementar pulsando sobre la X roja a la izquierda de la línea donde se declara la clase que hereda.



```
public class Bicicleta extends Vehiculo {  
  
    @Override  
    protected void parar() {  
        // TODO Auto-generated method stub  
  
    }  
  
}
```

- El modificador `@Override` se coloca encima de la declaración del método e indica que su implementación es obligatoria. Una clase abstracta no puede ser instanciada.

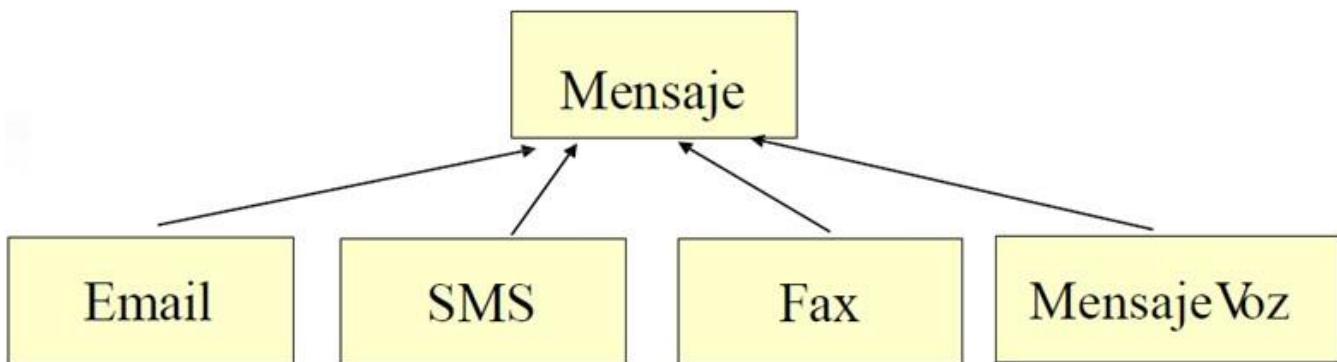
```
Vehiculo vechiculo = new Vehiculo();
```

- En una clase abstracta pueden existir métodos abstractos y métodos no abstractos.

```
public abstract class Coche {  
  
    // declaración de método abstracto  
    protected abstract void parar();  
  
    // declaración de método no abstracto  
    protected void parar(boolean choque) {  
    }  
}
```

- El modificador `abstract` no puede ser colocado en:

- Constructores.
- Métodos estáticos.
- Métodos privados.
- ◆ Las clases abstractas suelen utilizarse para representar clases con implementaciones parciales.
- ◆ El objetivo de las implementaciones parciales es proporcionar una interfaz común a todas las clases derivadas.



Ejercicio: Convierte la clase Persona en una clase abstracta y declara un método abstracto llamado `mostrarID` que debe ser implementado por las clases Estudiante y Profesor. El método `mostrarID` debe mostrar el ID del estudiante o del profesor y, a continuación, un guión y la nota media (en el caso del estudiante) o el número medio de estudiantes (en el caso del profesor).



## Interfaces

- ◆ Las interfaces son clases especiales cuyos métodos son todos abstractos (ninguno se encuentra implementado).
- ◆ Los posibles modificadores que pueden ser utilizados en una interfaz son:
  - Para atributos: `public`, `static` y `final`.
  - Para métodos: solamente `public`.
- ◆ Para declarar una interfaz se utiliza el modificador `interface`, en lugar de `class`.

```
public interface Vehiculo {  
  
    // declaración de un atributo  
    public static final float g = 9.8f;  
  
    // declaración de métodos abstractos  
    public abstract void parar();  
    public abstract void acelerar();  
    public abstract void desacelerar();  
    public abstract void girar_izquierda();  
    public abstract void girar_derecha();  
}
```

- ◆ Las interfaces son implementadas por las clases.
- ◆ Una clase que implementa una interfaz está obligada a implementar todos los métodos abstractos definidos por la interfaz.
- ◆ Para que una clase implemente una interfaz se utiliza la partícula implements a continuación del nombre de la clase.

```
public class Bicicleta implements Vehiculo {  
  
}
```

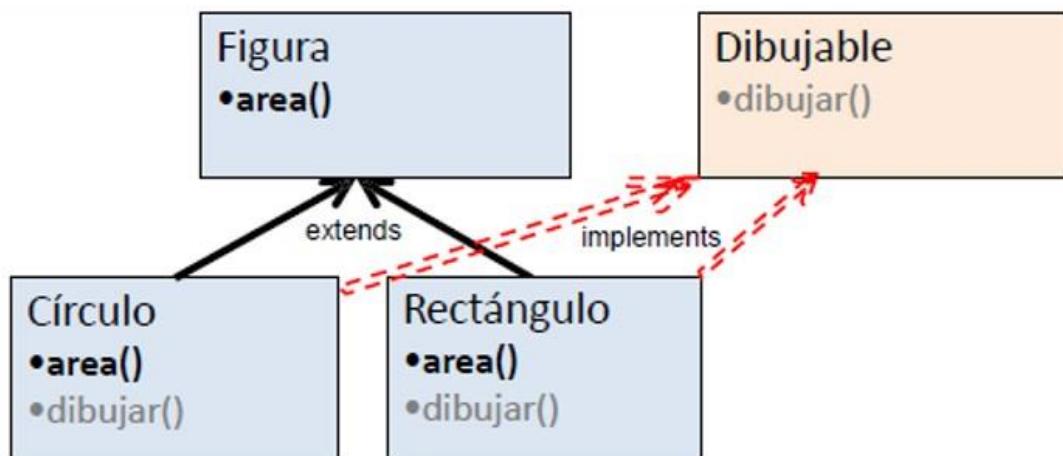
- ◆ Al igual que con las clases abstractas, Eclipse ayuda a añadir los métodos que la clase está obligada a implementar.

```
public class Bicicleta implements Vehiculo {  
  
    @Override  
    public void parar() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void acelerar() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void desacelerar() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void girar_izquierda() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void girar_derecha() {  
        // TODO Auto-generated method stub  
    }  
  
}
```

- ◆ Las interfaces son implementadas por las clases.
- ◆ Una clase que implementa una interfaz está obligada a implementar todos los métodos abstractos de la interfaz.
- ◆ Para que una clase implemente una interfaz se utiliza la partícula `implements` a continuación del nombre de la clase.

```
public class Bicicleta implements Vehiculo {  
  
}
```

- ◆ Al igual que con las clases abstractas, Eclipse ayuda a añadir los métodos que la clase está obligada a implementar.



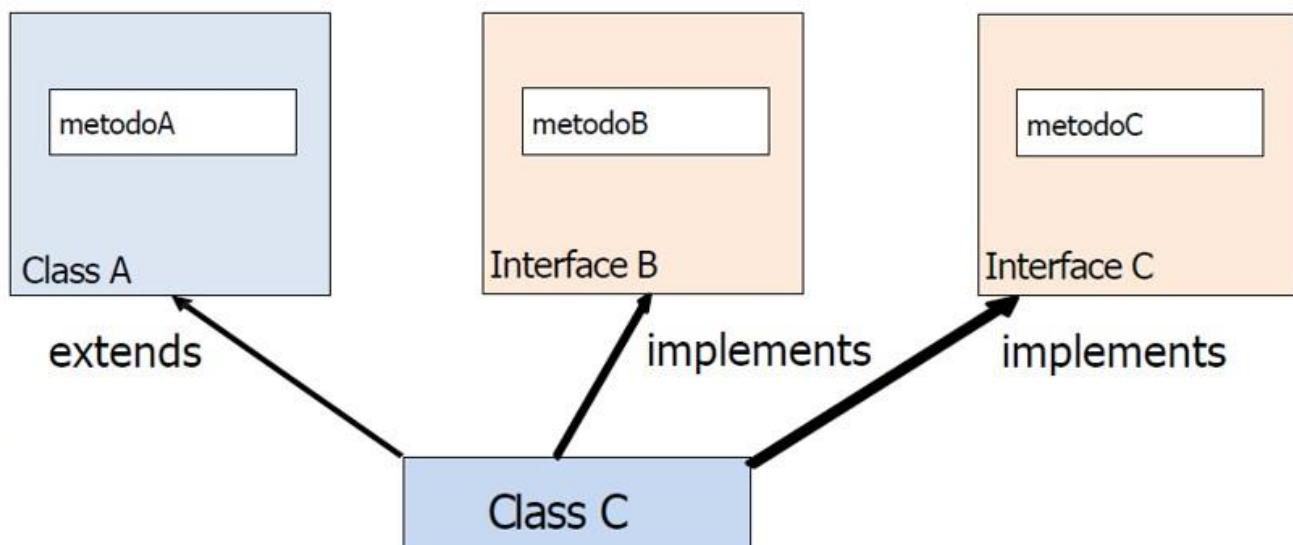
```
public abstract class Figura {...}
```

```
public interface Dibujable {...}
```

```
public class Círculo extends Figura implements Dibujable
```

```
public class Rectángulo extends Figura implements Dibujable
```

- Tal y como se comentó con anterioridad, en Java no se permite heredar varias clases (herencia múltiple).
- Sin embargo, sí se permite implementar varias interfaces al mismo tiempo.



Fichero ClassA.java

```
public class ClassA {
    public void metodoA() {
    }
}
```

Fichero InterfaceB.java

```
public interface InterfaceB {
    public abstract void metodoB();
}
```

Fichero InterfaceC.java

```
public interface InterfaceC {
    public abstract void metodoC();
}
```

Fichero ClassC.java

```
public class ClassC extends ClassA implements InterfaceB, InterfaceC {

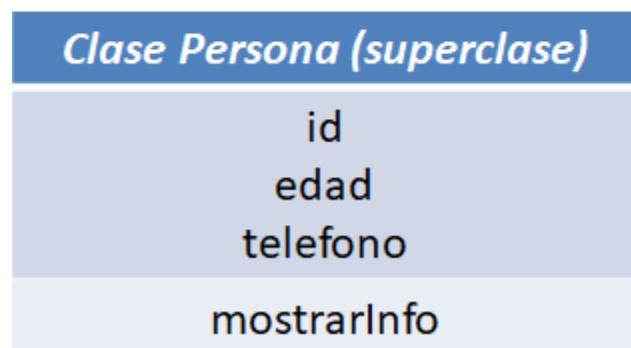
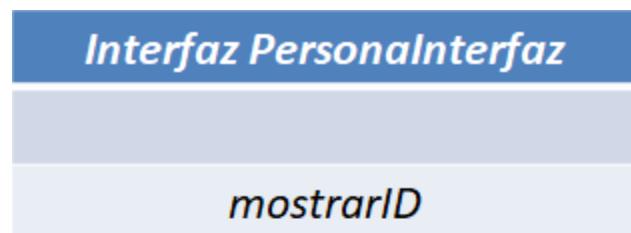
    @Override
    public void metodoC() {
        // TODO Auto-generated method stub
    }

    @Override
    public void metodoB() {
        // TODO Auto-generated method stub
    }
}
```

- En conclusión:

- Clase: todos los métodos se encuentran implementados.
- Clase abstracta: declarada con el modificador `abstract` y generalmente contiene al menos un método no implementado (`abstract`)
- Interfaz: declarada con el modificador `interface` y no debe tener ningún método implementado (todos son `abstractos`).

Ejercicio: Crea una interfaz llamada `PersonaInterfaz` que incluya el método abstracto `mostrarID`. La clase `Persona` debe implementar esta interfaz.



## Paquetes

- Un paquete agrupa a varias clases (e interfaces).
- La jerarquía de un paquete se corresponde con las jerarquías de un directorio.

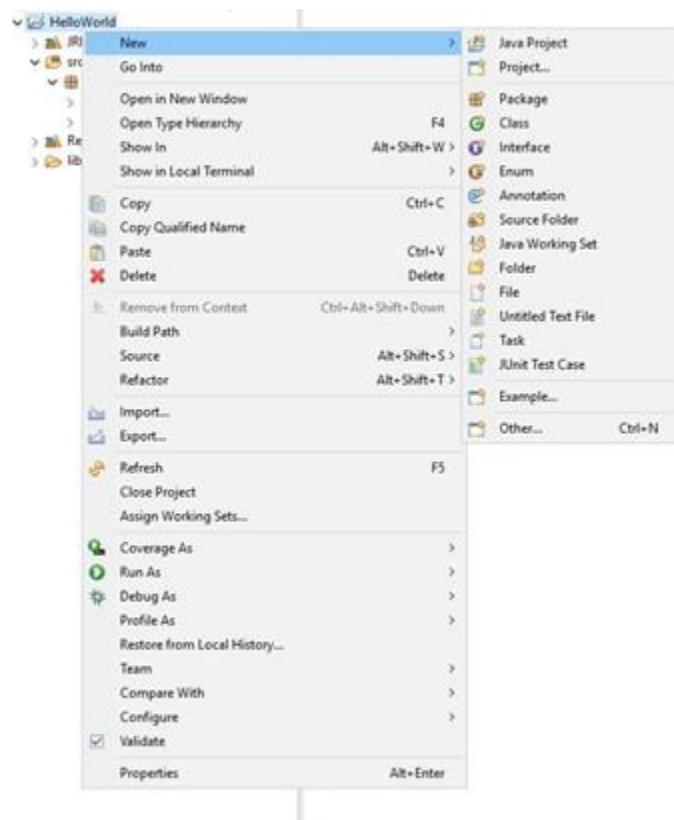


Ejemplo de un paquete llamado com.app.f1 que engloba a dos clases (Bicicleta y Vehiculo)

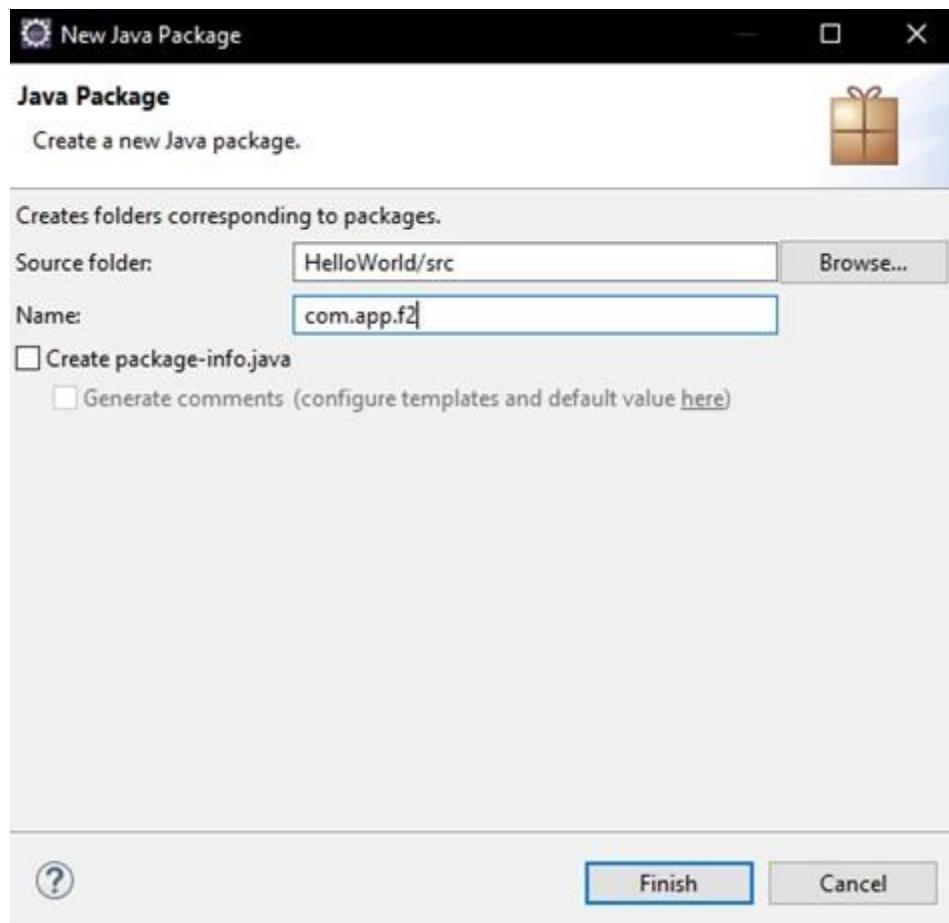
Los archivos del paquete se encuentran contenidos dentro del directorio src, donde la jerarquía del paquete corresponde con la estructura de subdirectorios



- Puede crearse un paquete para un proyecto en Eclipse pulsando el botón derecho del ratón sobre el nombre del Proyecto y, a continuación, accediendo a New -> Package.
- Eclipse creará automáticamente toda la estructura de directorios asociada al paquete.



- Posteriormente se especifica el nombre del paquete, utilizando la notación punto.



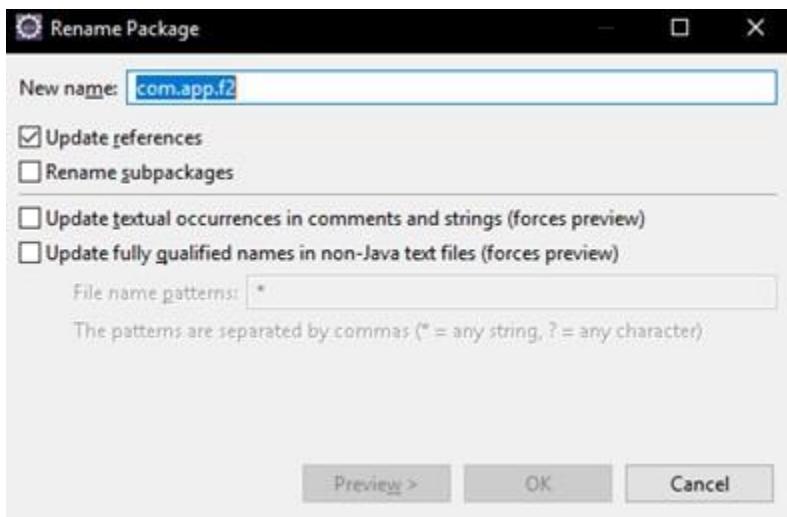
- El nuevo paquete se encuentra vacío tras su creación.



- En el nuevo paquete pueden crearse nuevas clases o arrastrar otras ya existentes desde otros paquetes.



- El nombre del paquete puede ser modificado en cualquier momento si es preciso.



- Las clases que pertenecen a un paquete declaran en su primera línea el nombre del paquete al que pertenecen mediante package.

```
package com.app.f2;  
  
public class Bicicleta {  
  
}
```

- Para que una clase pueda utilizar a otra clase que se encuentra en otro paquete es necesario importarla mediante import.

```
import com.app.f1.Vehiculo;
```

```

package com.app.f2;

import com.app.f1.Vehiculo;

public class Bicicleta implements Vehiculo {

    @Override
    public void parar() {
        // TODO Auto-generated method stub
    }

    @Override
    public void acelerar() {
        // TODO Auto-generated method stub
    }

    @Override
    public void desacelerar() {
        // TODO Auto-generated method stub
    }

    @Override
    public void girar_izquierda() {
        // TODO Auto-generated method stub
    }

    @Override
    public void girar_derecha() {
        // TODO Auto-generated method stub
    }
}

```



Fichero ArchivoBicicleta.java  
ubicado dentro del paquete  
com.app.f2

- El JDK de Java incorpora gran cantidad de clases agrupadas en paquetes y que pueden ser importadas para su uso.

```

import java.math.BigDecimal;

public class Main {

    public static void main(String[] args) {

        BigDecimal numero_grande = new BigDecimal(1000000000);
    }
}

```

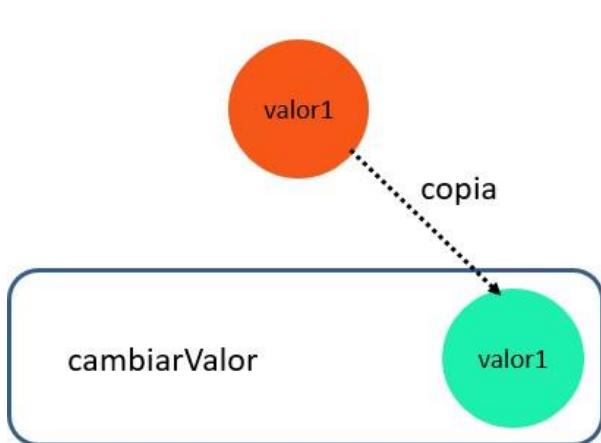
- El uso de paquetes en Java proporciona las siguientes ventajas:
  - Agrupamiento de clases con características comunes.
  - Reutilización de código al promover principios de programación orientada a objetos como la encapsulación y la modularidad.
  - Mayor seguridad al existir niveles de acceso.
  - Evita la colisión de clases que tengan el mismo nombre.

- Mantenibilidad de código: si un paquete se enfoca en la agrupación de clases con características comunes, el cambio en la funcionalidad se limita a las clases contenidas en dicho paquete.

Ejercicio: Crea un nuevo paquete llamado com.clase.info y agrega todos los archivos con las clases e interfaces creadas hasta el momento: PersonaInterfaz.java, Persona.java, Estudiante.java y Profesor.java

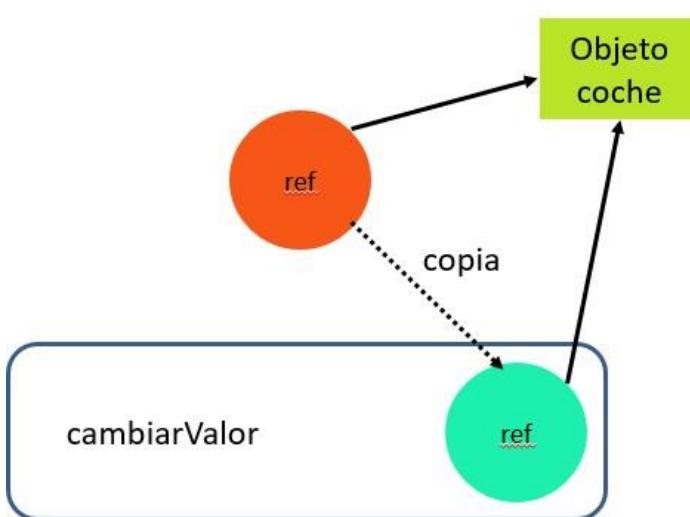
## Referencias a objetos

- La forma de almacenamiento de variables primitivas difiere del almacenamiento de objetos.
- Las variables primitivas se almacenan en un espacio de la memoria del ordenador. Cuando una variable primitiva se pasa por argumentos a un método, se hace una copia de su valor.



```
public class Main {
    public static void main(String[] args) {
        int valor1 = 5;
        Main.cambiarValor(valor1);
        // 5
        System.out.println(valor1);
    }
    public static void cambiarValor(int valor1) {
        valor1 = 6;
        // 6
        System.out.println(valor1);
    }
}
```

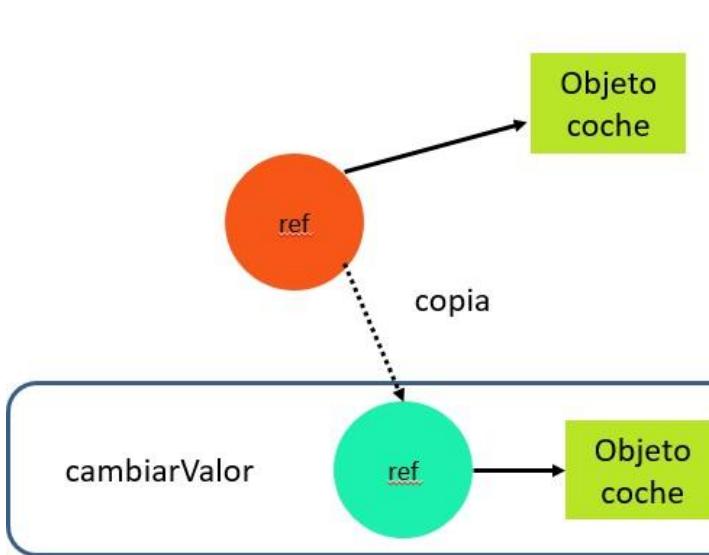
- Las objetos se almacenan en un espacio de la memoria como una dirección de memoria que apunta a los datos del objeto.
- Cuando un objeto se pasa por argumentos a un método, se hace una copia de su referencia (dirección de memoria).



```
public static void main(String[] args) {
    Coche coche = new Coche();
    // cambia el valor de los asientos a 2
    coche.setAsientos(2);
    // 2
    System.out.println(coche.getAsientos());
    cambiarValor(coche);
    // 4
    System.out.println(coche.getAsientos());
}

public static void cambiarValor(Coche coche) {
    // cambia el valor de los asientos a 2
    coche.setAsientos(4);
}
```

- La referencia dentro del método puede reasignarse a un objeto nuevo utilizando new.
- El nuevo objeto creado en el método desaparece al finalizar el método, por lo que el objeto creado se pierde.



```
public static void main(String[] args) {
    Coche coche = new Coche();
    // cambia el valor de los asientos a 2
    coche.setAsientos(2);
    // 2
    System.out.println(coche.getAsientos());
    cambiarValor(coche);
    // 2 (ahora el valor no cambia)
    System.out.println(coche.getAsientos());
}

public static void cambiarValor(Coche coche) {
    coche = new Coche();
    // cambia el valor de los asientos a 4
    coche.setAsientos(4);
}
```

## Convenciones en los nombres

- A continuación se comentan algunas convenciones de nombres para el lenguaje de programación Java.
- Siempre que sea posible es conveniente seguir estas recomendaciones para un buen mantenimiento y legibilidad del código.
- [El documento oficial de convenciones y buenas prácticas puede consultarse en la página oficial de Oracle.](#)

- El nombre completo de un paquete en Java debe ser único, escribirse en letras en minúsculas y debería comenzar con uno de los nombres de dominio de nivel superior (com, edu, gov, mil, net, org).



- Los componentes posteriores del nombre del paquete varían de acuerdo con las convenciones internas de la organización

## Clases e interfaces

- Los nombres de las clases deberían ser sustantivos, con la primera letra de cada palabra en mayúscula.
- El nombre de las interfaces también deben ser sustantivos, con la primera letra de cada palabra en mayúscula.
- Los nombres de los métodos deberían ser verbos, en minúsculas y con la primera letra de cada palabra interna (a partir de la segunda) en mayúscula.

```
public class Bicicleta {
    protected void parar() {
    }
}
```

```
public interface Bicicleta {
    public abstract void parar();
}
```

## Constantes

- Las constantes (declaradas con el modificador final) en Java deberían escribirse con todas las letras en mayúsculas y con las palabras separadas por guiones bajos.

```
public class Bicicleta {
    public static final float CONSTANTE_UNIVERSAL = 9.8f;
    public static final float PI = 3.4f;
}
```

Tu carrera digital ~

# Módulo 4

# Bases de datos (SQL)

Introducción las bases de datos,  
MariaDB y SQL



# Introducción las bases de datos, MariaDB y SQL

- ◆ [Introducción](#)
- ◆ [Tipos de bases de datos](#)
- ◆ [Bases de datos relacionales y conceptos básicos](#)
- ◆ [Instalación de XAMPP](#)
- ◆ [phpMyAdmin](#)
  - [Creación de usuarios y bases de datos](#)
  - [Creación de tablas](#)
  - [Manejo de datos](#)
- ◆ [El lenguaje de consultas SQL](#)
- ◆ [Crear datos](#)
- ◆ [Leer datos](#)
- ◆ [Actualizar datos](#)
- ◆ [Eliminar datos](#)

## Introducción

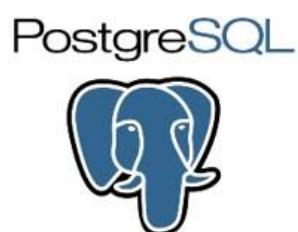
- ◆ La forma clásica de almacenamiento de datos de forma persistente en un computador es mediante el sistema de archivos del sistema operativo, siguiendo una organización jerárquica en directorios.
- ◆ Sin embargo, el almacenamiento de datos en ficheros posee las siguientes desventajas:
  - Redundancia: existen datos que pueden repetirse en diferentes archivos, lo que provoca duplicidad y un uso ineficiente de los recursos.
  - Inconsistencia: debido al problema de la redundancia, si en un archivo se hacen cambios, los otros archivos duplicados no son modificados de forma automática.
  - Acceso: el acceso a archivos ubicados en diferentes ubicaciones, así como la obtención, consulta y modificación de los datos es difícil y poco práctica.
  - Formato de los archivos: los archivos almacenan los datos en formatos de muy diversa índole, no existiendo una forma única de procesar la información.
  - Conurrencia: la modificación al mismo tiempo de un archivo por parte de múltiples usuarios puede producir datos inconsistentes.
  - Seguridad: la granularidad de la seguridad en el acceso es a nivel de fichero, por lo que no existe posibilidad de denegar el acceso a determinadas partes de un fichero para un usuario en particular.



- Esta problemática anterior motivó la aparición de los sistemas de bases de datos.

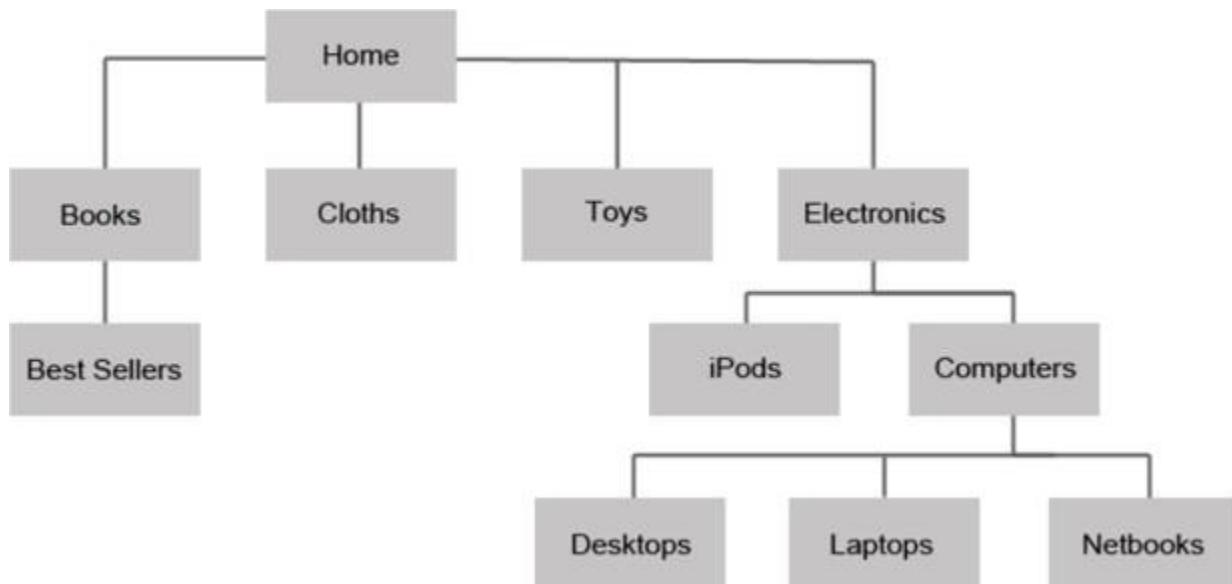


- Una base de datos es una colección organizada de datos estructurados que se almacenan en un sistema informático.
- Normalmente, una base de datos está controlada por un Sistema de gestión de bases de datos (SGBD).
- Un SGBD es un software que permite el almacenamiento, modificación y extracción de la información en una base de datos.
- Los datos y el SGBD constituyen el sistema de base de datos o simplemente base de datos.

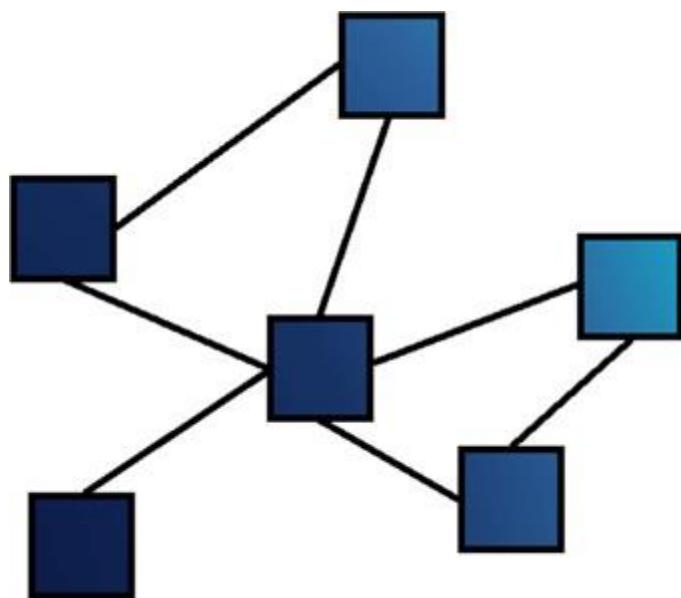


## Tipos de bases de datos

- Bases de datos con estructura jerárquica: las relaciones entre los datos forman una estructura en árbol. Actualmente las bases de datos jerárquicas más utilizadas son IMS de IBM y el Registro de Windows de Microsoft (regedit).



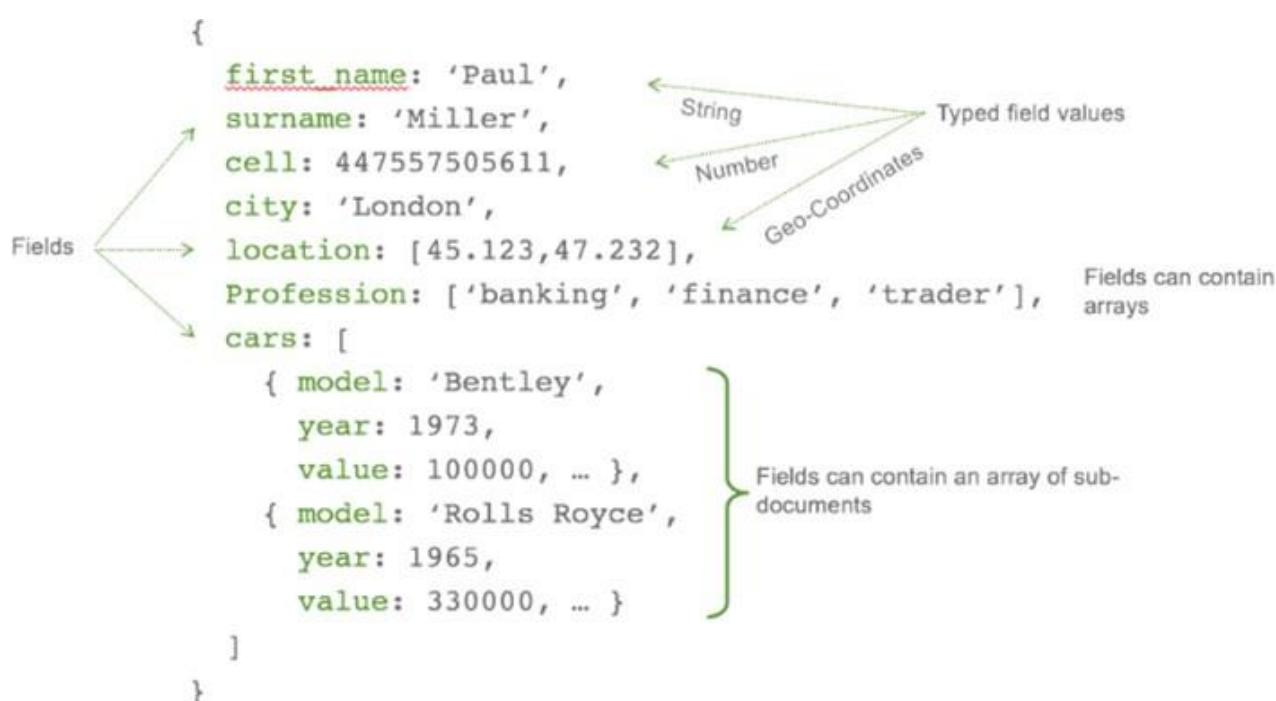
- Bases de datos con estructura en red: contiene relaciones más complejas que las jerárquicas, admitiendo que un dato puede estar relacionado con otro o con varios al mismo tiempo sin necesidad de jerarquía.



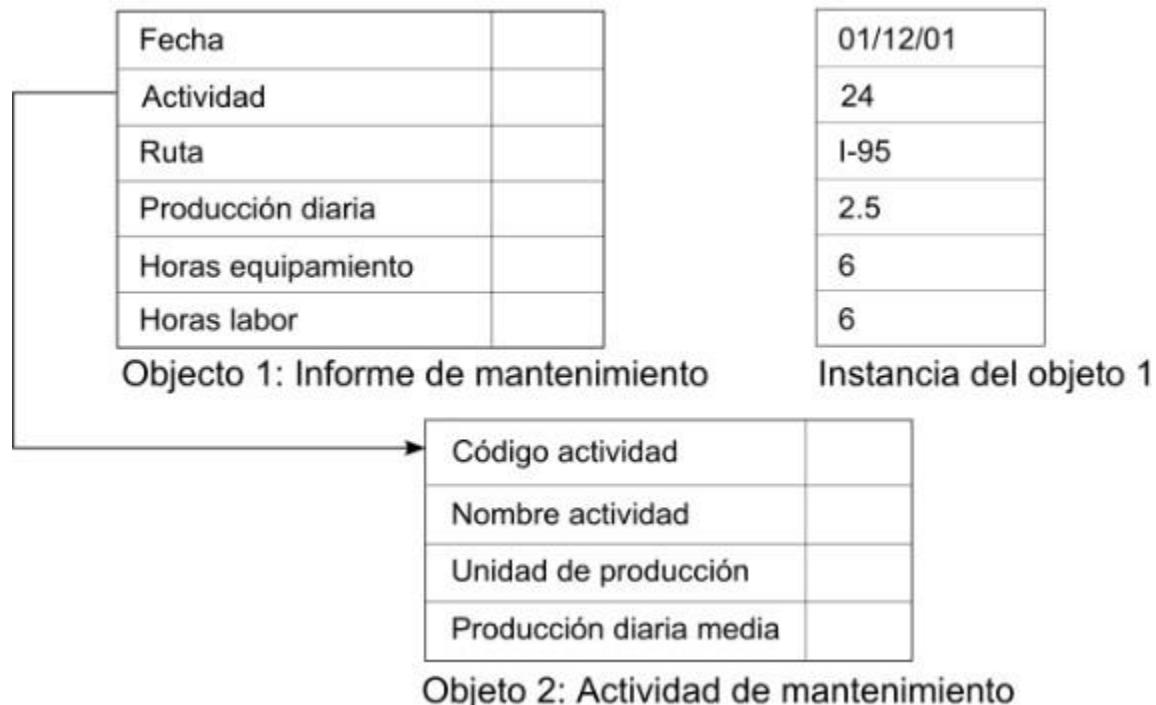
- Bases de datos clave/valor: presentan una estructura de datos que asocia claves con valores. La operación principal que soporta de manera más eficiente es la búsqueda por clave.

Key	Value
12345	4567.3456787
12346	{ addr1 : "The Grange", addr2: "Dublin" }
12347	"top secret password"
12358	"Shopping basket value : 24560"
12787	12345

- Bases de datos documentales: almacenan los datos de forma estructurada en documentos, de acuerdo a un determinado formato (por ejemplo, JSON).

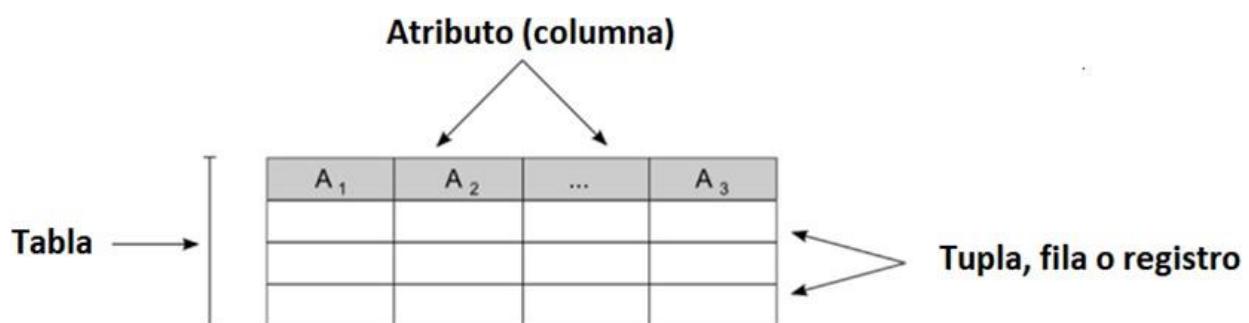


- Bases de datos con estructura orientada a objetos: los datos son almacenados de acuerdo a una estructura orientada a objetos, siguiendo el paradigma de los lenguajes orientados a objetos.



## Bases de datos relacionales y conceptos básicos

- Las bases de datos relacionales son implementaciones del modelo relacional.
- El modelo relacional está basado en dos conceptos:
  - Tablas
  - Relaciones
- Una tabla es una colección de elementos organizados en filas y columnas.



<b>id</b>	<b>usuario</b>	<b>password</b>	<b>email</b>
1	pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7	pepito@correo.com
2	juanito	44e235daceb8ee492f6bb5249ffb0aec993a330d8c1fd9cf30447deae1ecda3	juanito@correo.com
3	marcos	2a5d486545bbd36a0c1a9786a68a1d578370c2e eb339699f853b031322d0eb5c	marcos@correo.com
4	roberto	daf5f918ab83da2bcb7bbe7de3b18d70f77e4041a20626f40144c2d6adf62875	roberto@correo.com

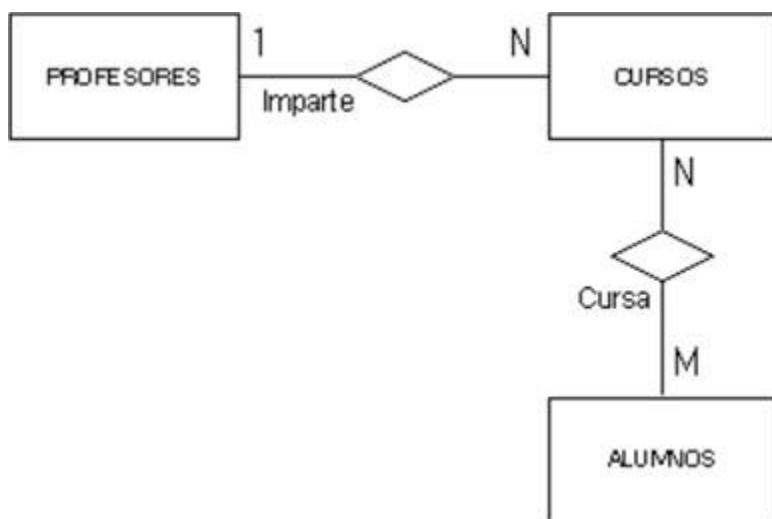
- Cada registro en la tabla es llamado tupla o fila.
- Los atributos o columnas son los campos que constituyen cada tupla.
- Cada columna está compuesto obligatoriamente por un nombre y un tipo de dato.
- Adicionalmente al nombre y al tipo de dato, cada columna de una tabla puede ser también clave primaria.
- Una columna que es clave primaria (primary key) identifica de forma única a cada fila de una tabla.
- No puede existir dos filas en una tabla que poseen el mismo valor para la columna que es clave primaria.



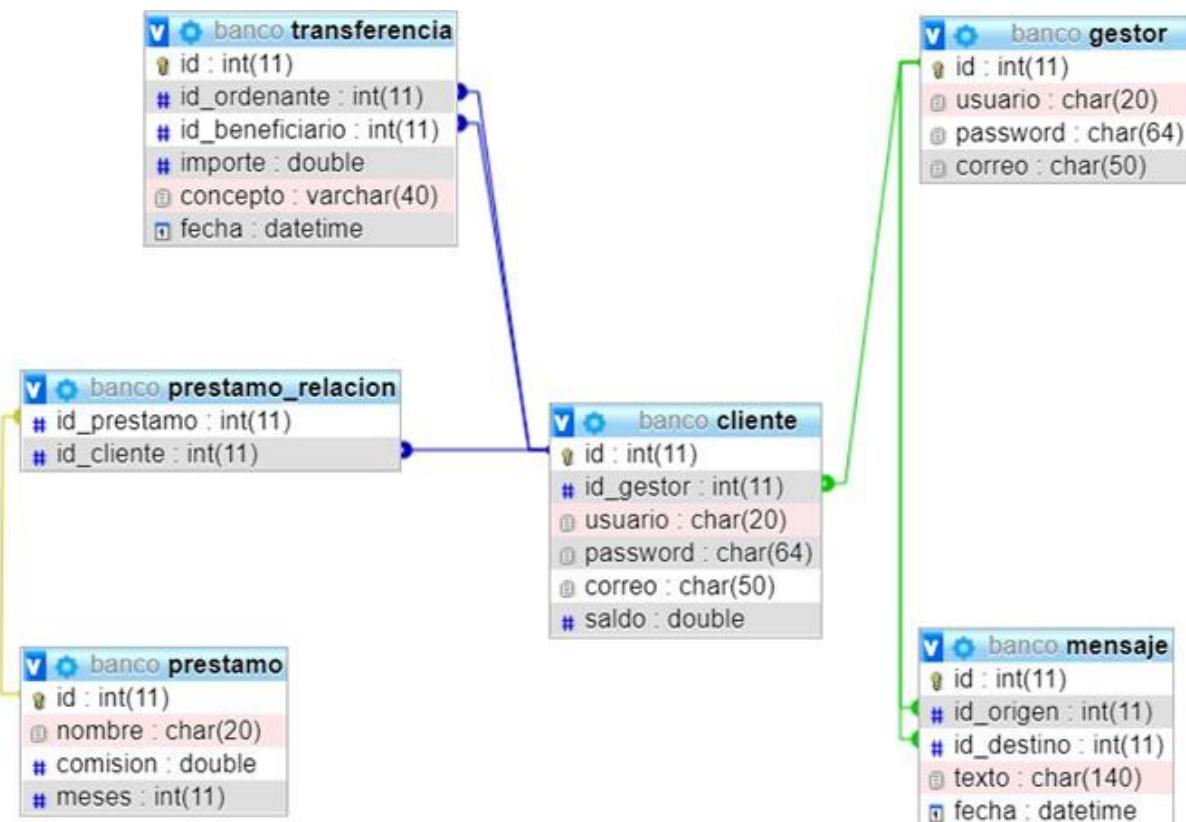
- A veces es útil definir también una columna que es clave primaria también como autoincremental.
- Una columna autoincremental aumentará su valor cada vez que se inserte una nueva tupla.

<b>id</b>	<b>usuario</b>	<b>password</b>	<b>email</b>
1	pepito	627a6ab55608711e421a1132be531a7a7fec1df7 b253dda73ec482824269e5d7	pepito@correo.com
2	juanito	44e235daceb8ee492f6bb5249ffb0aecc993a330 d8c1fd9cf30447deae1ecda3	juanito@correo.com
3	marcos	2a5d486545bbd36a0c1a9786a68a1d578370c2e eb339699f853b031322d0eb5c	marcos@correo.com
4	roberto	daf5f918ab83da2bcb7bbe7de3b18d70f77e4041 a20626f40144c2d6adf62875	roberto@correo.com
5	...	...	...
6	...	...	...
7	...	...	...

- Las relaciones entre tablas pueden ser de varios tipos:
  - Uno a Uno (1:1): un registro de una tabla A se relaciona con un registro de una tabla B. Ejemplo: usuario – DNI.
  - Uno a Varios (1:N): un registro de una tabla A se relaciona con varios registros de una tabla B. Ejemplo: profesor – alumno.
  - Varios a Varios (N:M): varios registros de una tabla A se relaciona con varios registros de una tabla B. Ejemplo: actor – película.



- Un conjunto de tablas constituyen una base de datos.



## Instalación de XAMPP

- XAMPP son las siglas de:
  - X: cualquier sistema operativo.
  - A: Apache
  - M: MariaDB/MySQL
  - P: PHP
  - P: Perl



Apache Web Server



- XAMPP instala también el software phpMyAdmin, una herramienta escrita en PHP para manejar MariaDB/MySQL mediante la Web.



- XAMP puedes descargarse desde su página oficial.

- También existen versiones portables que no requieren de instalación.
- Aunque se ofrecen varias versiones para la descarga de XAMPP, siempre es recomendable optar por la última versión.

## XAMPP para Windows 7.2.26, 7.3.13 & 7.4.1

Versión		Suma de comprobación	Tamaño
7.2.26 / PHP 7.2.26	<a href="#">¿Qué está incluido?</a>	<a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Descargar (64 bit)</a> 145 Mb
7.3.13 / PHP 7.3.13	<a href="#">¿Qué está incluido?</a>	<a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Descargar (64 bit)</a> 146 Mb
7.4.1 / PHP 7.4.1	<a href="#">¿Qué está incluido?</a>	<a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Descargar (64 bit)</a> 146 Mb

- Una advertencia puede aparecer tras ejecutar el programa si existe un antivirus instalado. En el FAQ de XAMPP puede encontrarse más información al respecto.

Question X

It seems you have an antivirus running. In some cases, this may slow down or interfere the installation of the software. Please visit the following link to learn more about this.

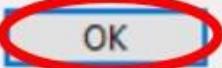
<http://apachefriends.org/en/faq-xampp-windows.html#antivirus>

Continue with installation?

- Algunas funciones de XAMPP pueden estar limitadas si el User Account Control (UAC) de Windows se encuentra activo.

 Warning

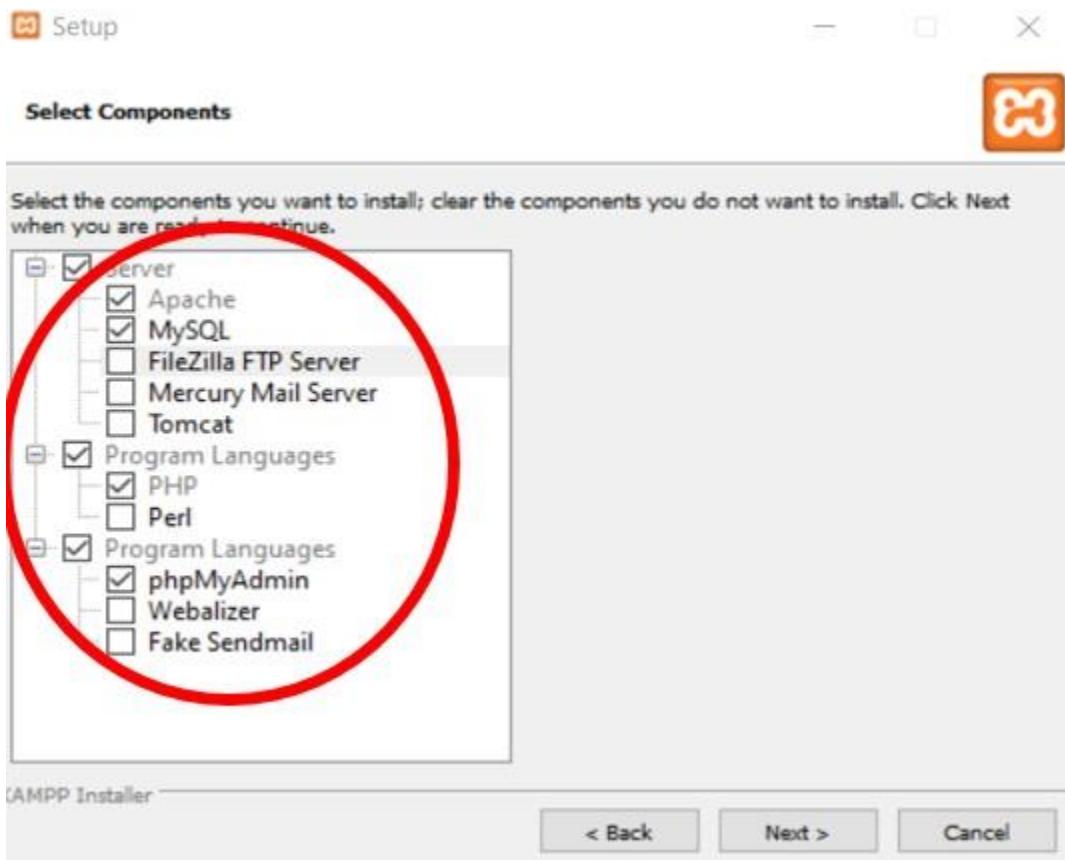
Important! Because an activated User Account Control (UAC) on your system some functions of XAMPP are possibly restricted. With UAC please avoid to install XAMPP to C:\Program Files (missing write permissions). Or deactivate UAC with msconfig after this setup.

 OK

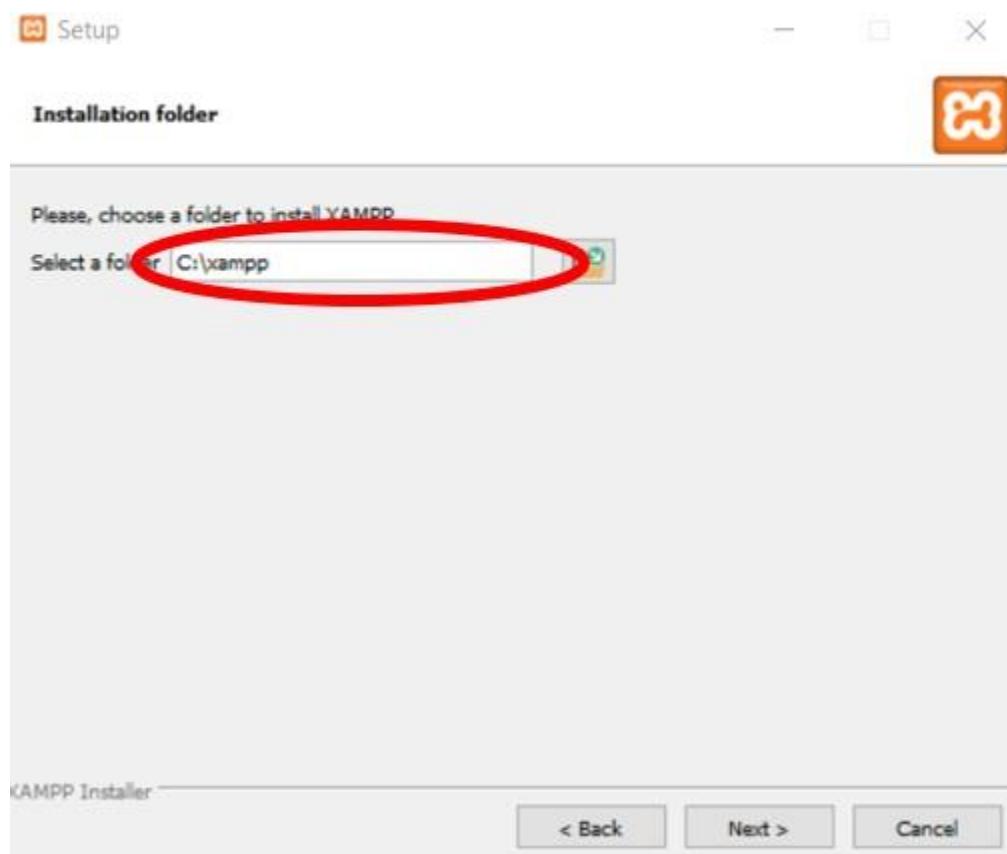
- El asistente de instalación de XAMPP es muy simple.



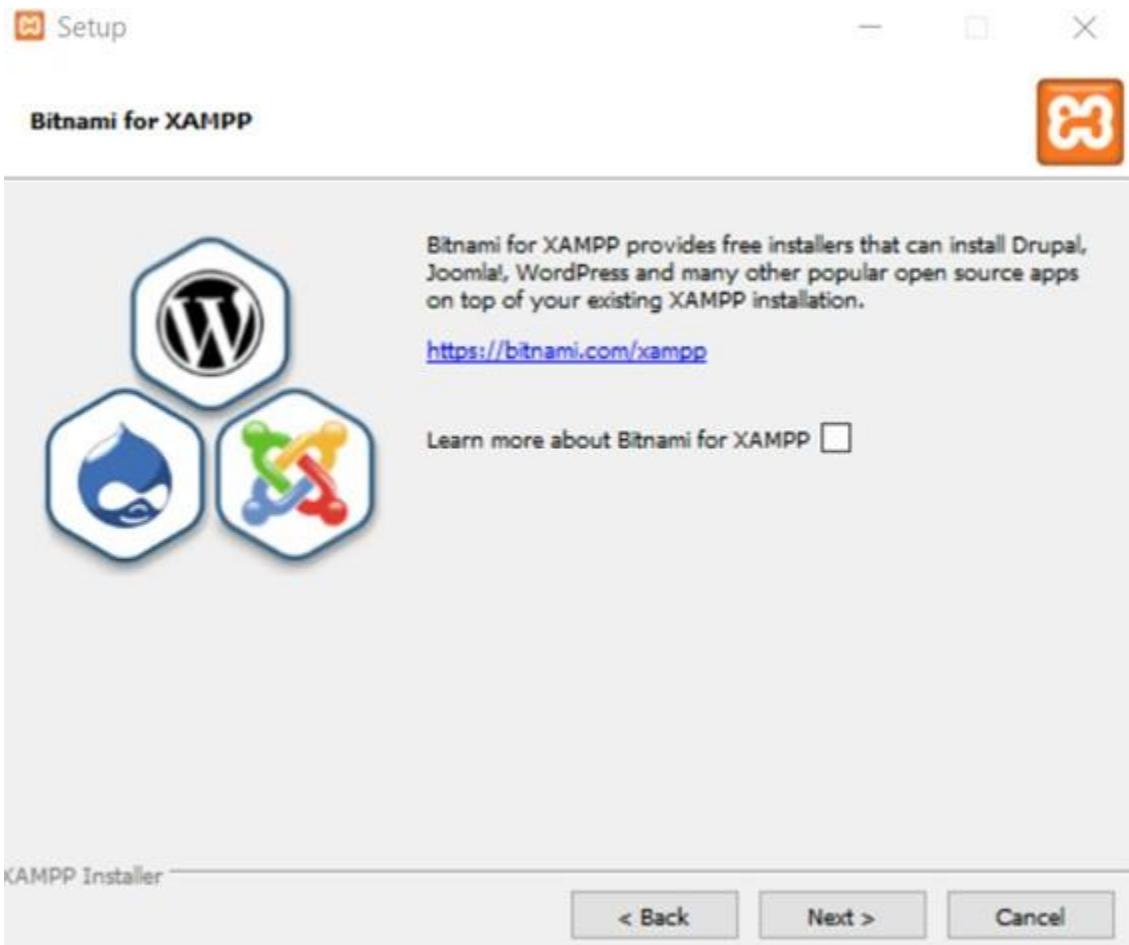
- Los únicos componentes de XAMPP a instalar son: Apache, MySQL, PHP y phpMyAdmin. El resto pueden deseleccionarse.



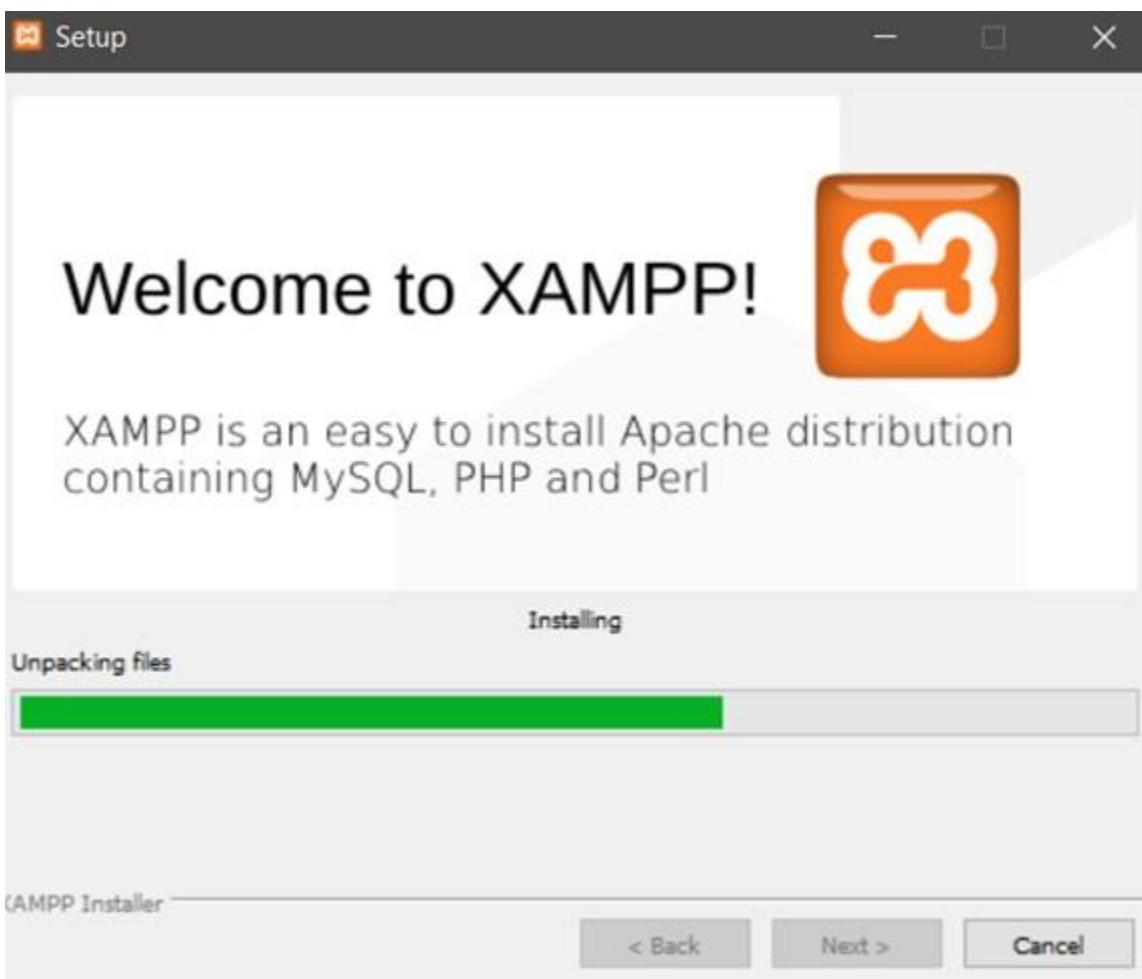
- A continuación aparece la ruta de destino donde se instalará XAMPP.



- En el siguiente paso se ofrecerá la posibilidad de obtener más información sobre la iniciativa Bitnami.



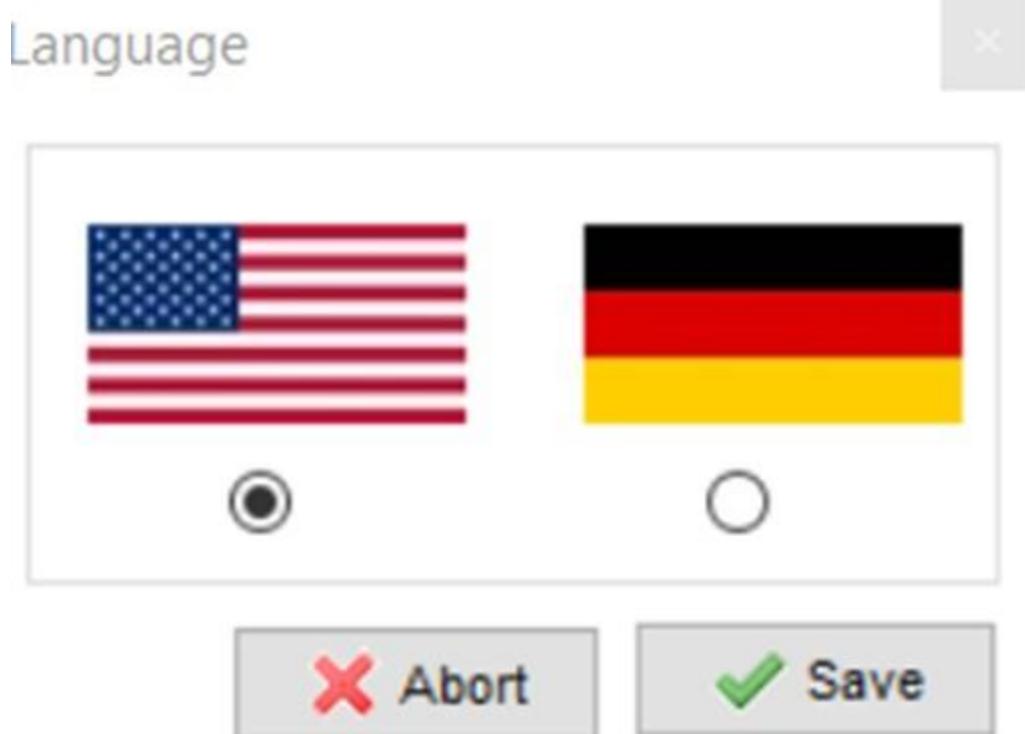
- Seguidamente el proceso de instalación comenzará.



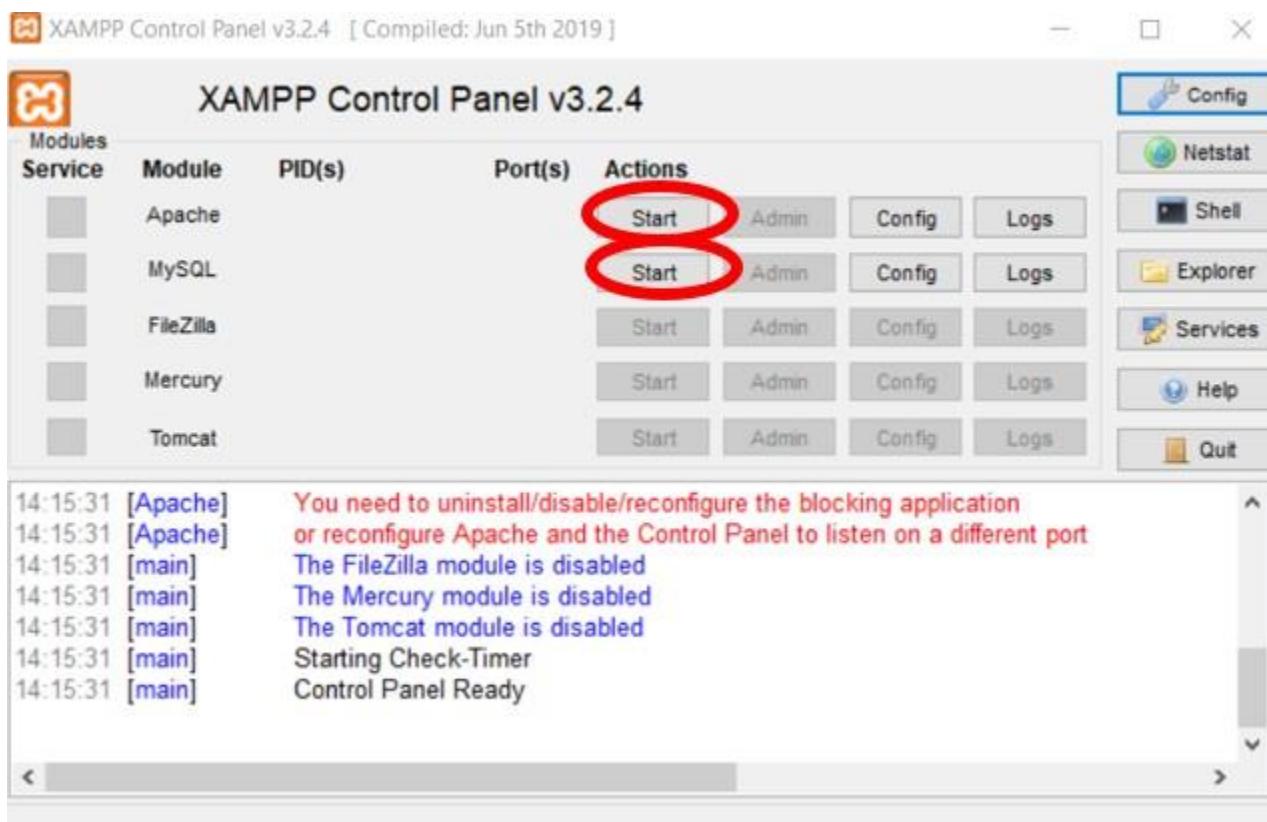
- Finalmente es posible abrir directamente el Panel de Control de XAMPP una vez terminada la instalación.



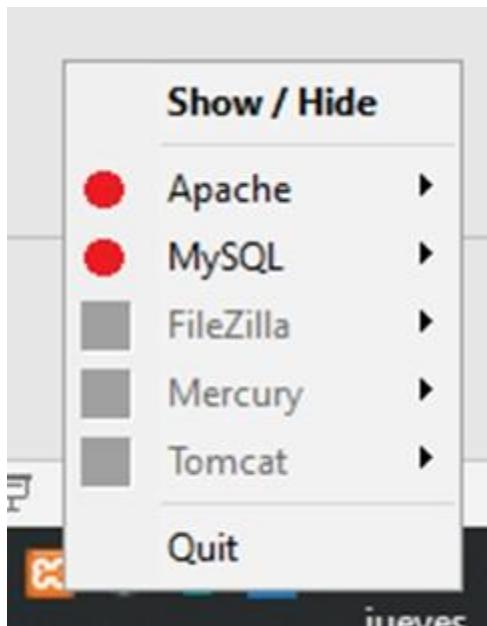
- En la primera ejecución del Panel de Control de XAMPP aparecerá la posibilidad de elegir entre el idioma inglés o el alemán.



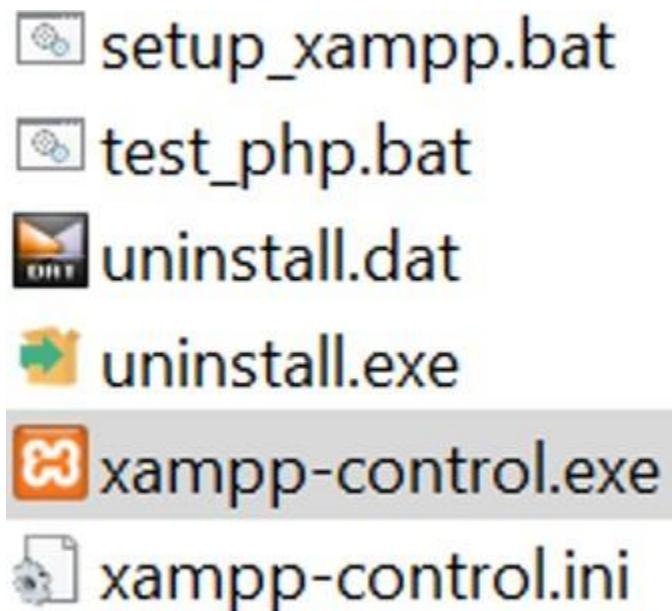
- El Panel de Control de XAMPP permite arrancar o parar servicios, cambiar los archivos de configuración, entre otras tareas.



- El Panel de Control de XAMPP también aparece en la barra de notificaciones de Windows.
- Para cerrarlo hay que pulsar con el botón derecho del ratón sobre el icono de XAMPP y **Quit**.



- El ejecutable (xampp-control.exe) del Panel de Control de XAMPP se encuentra en el directorio C:\xampp (directorio de instalación por defecto).



- En Windows pueden presentarse problemas para abrir puertos por debajo del 1024 (puertos bien conocidos).
- El servidor Apache sirve peticiones HTTP en el puerto 80, por lo que será necesario cambiarlo a uno superior al 1024 desde el archivo httpd.conf

The screenshot shows the XAMPP Control Panel interface. On the left, there's a sidebar with tabs for Config, Icons, and Shell. Under Config, several files are listed: Apache (httpd.conf), Apache (httpd-ssl.conf), Apache (httpd-xampp.conf), PHP (php.ini), phpMyAdmin (config.inc.php), and three <Browse> sections for Apache, PHP, and phpMyAdmin. The Apache (httpd.conf) file is currently selected and highlighted in blue. On the right, the content of this file is displayed in a code editor. A red circle highlights the line "Listen 8081", which is the line being modified.

```

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 8081

```

- También es necesario cambiar el puerto por defecto del servidor Apache para peticiones HTTPS (443) a otro superior a 1024. Para ello es necesario abrir el archivo httpd-ssl.conf

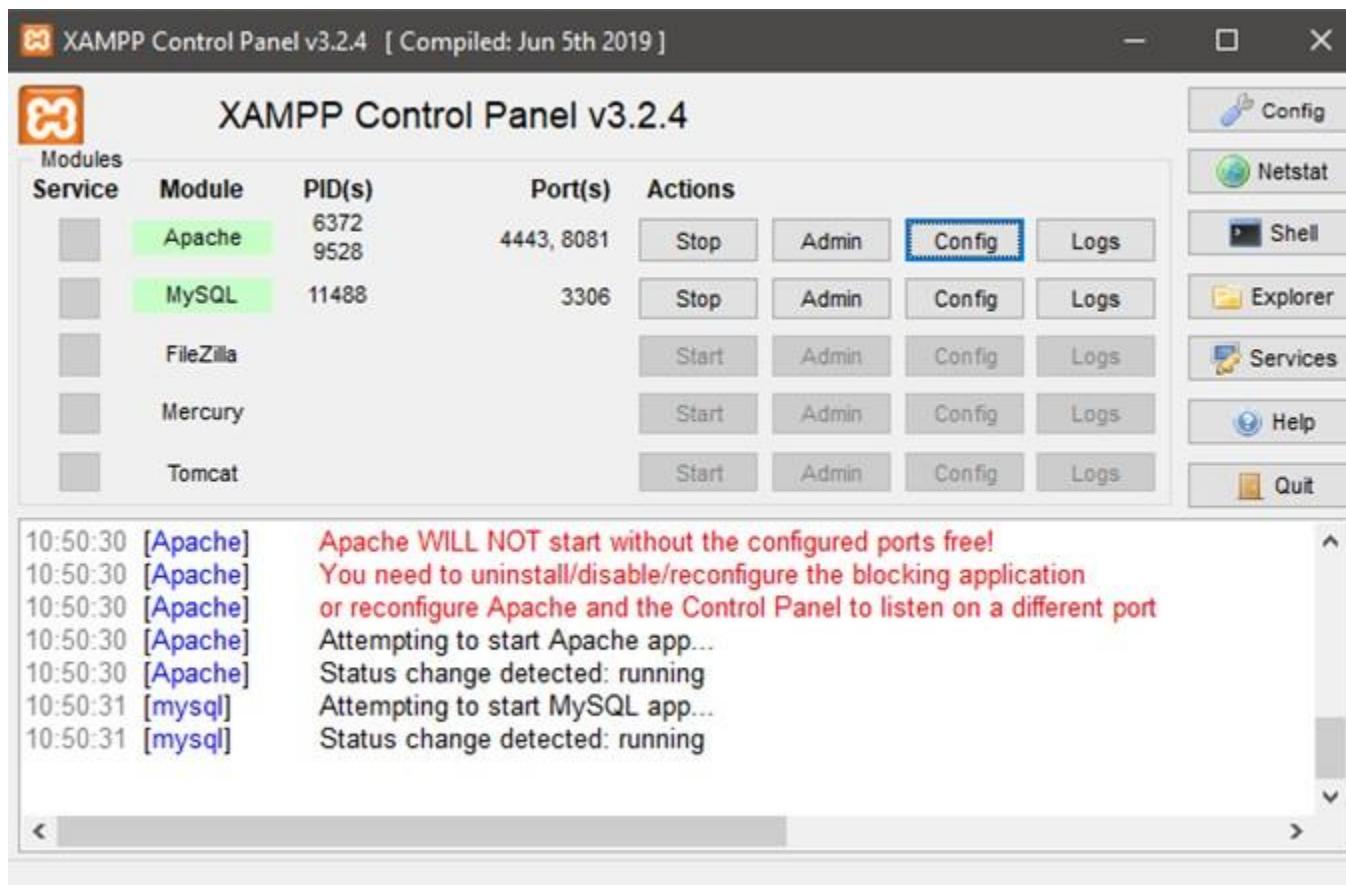
The screenshot shows the XAMPP Control Panel interface. The sidebar shows the Apache (httpd-ssl.conf) file is selected. On the right, the content of this file is displayed in a code editor. A red circle highlights the line "Listen 4443", which is the line being modified.

```

#
# When we also provide SSL we have to listen to the
# standard HTTP port (see above) and to the HTTPS port
#
Listen 4443

```

- Una vez modificados los archivos de configuración pueden iniciarse Apache y MySQL.



- Para verificar que el servidor Apache está funcionando correctamente puede accederse a la dirección <http://127.0.0.1:8081> o <http://localhost:8081>, donde 8081 es el puerto HTTP.

The screenshot shows the XAMPP Apache + MariaDB + PHP + Perl welcome page. It features the XAMPP logo and the text "Welcome to XAMPP for Windows 7.4.1". Below that, it says: "You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications." At the bottom, it says: "XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others. If you want have your XAMPP accessible from the internet, make sure you understand the implications and you checked the FAQs to learn how to protect your site. Alternatively you can use WAMP, MAMP or LAMP which are similar packages which are more suitable for production."

## Welcome to XAMPP for Windows 7.4.1

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others. If you want have your XAMPP accessible from the internet, make sure you understand the implications and you checked the [FAQs](#) to learn how to protect your site. Alternatively you can use WAMP, MAMP or LAMP which are similar packages which are more suitable for production.

Start the XAMPP Control Panel to check the server status.

## phpMyAdmin

- La aplicación phpMyAdmin se encuentra en la dirección <http://localhost:8081/phpmyadmin/>

The screenshot shows the phpMyAdmin configuration page. On the left, there's a sidebar with a tree view of databases (Nueva, information\_schema, mysql, performance\_schema, phpmyadmin, test) and a 'Consola' link. The main area has two tabs: 'Configuraciones generales' and 'Configuraciones de apariencia'. Under 'Configuraciones generales', there's a dropdown for 'Cotejamiento de la conexión al servidor' set to 'utf8mb4\_unicode\_ci'. Under 'Configuraciones de apariencia', there are settings for 'Idioma - Language' (Español - Spanish), 'Tema' (pmahomme), and 'Tamaño de fuente' (82%). A 'Más configuraciones' link is also present. To the right, there are three panels: 'Servidor de base de datos' (Server status: 127.0.0.1 via TCP/IP, MariaDB, SSL off, version 10.4.11-MariaDB), 'Servidor web' (Apache 2.4.41, OpenSSL 1.1.1c, PHP 7.4.1), and 'phpMyAdmin' (About version 4.9.2, documentation, official site, contribute, support, change log, license). A 'Consola' link is at the bottom.

- Al instalar un sistema de bases de datos como MySQL es importante:
  1. Crear un usuario root o administrador que posea permisos exclusivos para la creación, modificación o eliminación de usuarios. Habitualmente la creación del usuario root se realiza automáticamente durante el proceso de instalación de MySQL (aunque con XAMPP no ocurre).
  2. Loguearse con el usuario root.
  3. Crear una base de datos.
  4. Crear un usuario y otorgarle permisos totales y exclusivos sobre la base de datos creada en el paso anterior.
- La creación de usuarios en phpMyAdmin puede realizarse desde la pestaña de "Cuentas de usuarios".

## Creación de usuarios y bases de datos

- A continuación puede crearse un nuevo usuario desde el enlace "Agregar cuenta de usuario".

## Vista global de las cuentas de usuario

Nombre de usuario	Nombre del servidor	Contraseña	Privilegios globales	Grupo de usuario	Conceder	Acción
cualquiera	%	No	USAGE		No	Editar privilegios  Exportar
banco	localhost	Sí	USAGE		No	Editar privilegios  Exportar
pma	localhost	No	USAGE		No	Editar privilegios  Exportar
root	127.0.0.1	No	ALL PRIVILEGES		Si	Editar privilegios  Exportar
root	::1	No	ALL PRIVILEGES		Si	Editar privilegios  Exportar
root	localhost	No	ALL PRIVILEGES		Si	Editar privilegios  Exportar

↑ Seleccionar todo Para los elementos que están marcados: Exportar

[Nuevo](#) [Agregar cuenta de usuario](#)

- Para crear un nuevo usuario se establece: nombre de usuario, nombre de host (el valor debe ser localhost), contraseña (dos veces). Finalmente se crea una base de datos (con el mismo nombre que el usuario) otorgando todos los privilegios al usuario creado.

### Agregar cuenta de usuario

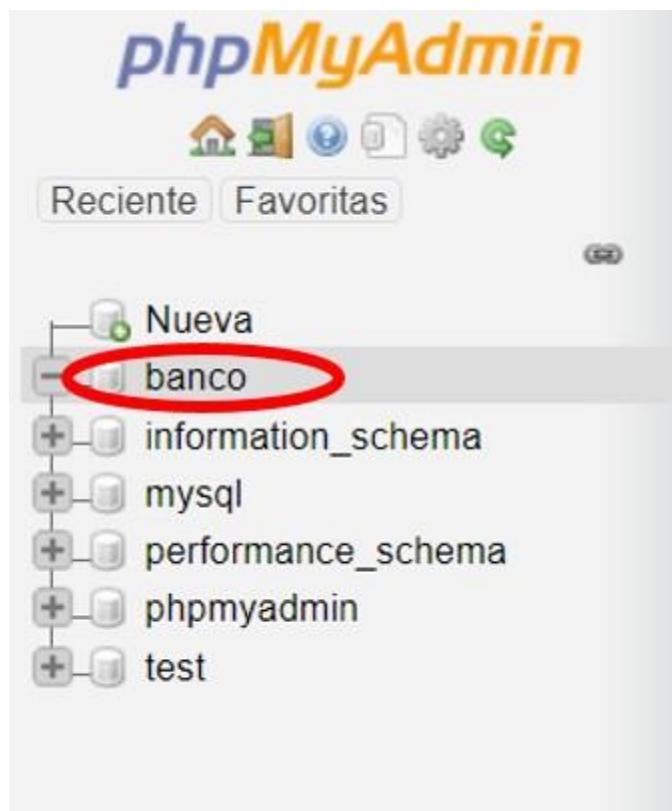
Información de la cuenta

Nombre de usuario:	Use el campo de texto	banco
Nombre de Host:	Use el campo de texto	localhost
Contraseña:	Use el campo de texto	****
Debe volver a escribir:	****	
Complemento de autenticación:	Autenticación de MySQL nativo	
Generar contraseña:	<input type="button" value="Generar"/>	

Base de datos para la cuenta de usuario

Crear base de datos con el mismo nombre y otorgar todos los privilegios.  
Otorgar todos los privilegios al nombre que contiene comodín (username\_%).

- La base de datos nueva aparecerá a la izquierda de la ventana, junto a otras bases de datos creadas por defecto en la instalación de MySQL y phpMyAdmin.



- Al acceder a la nueva base de datos creada (desde el menú anterior ubicado a la izquierda), se acceden a las distintas opciones disponibles.
- Inicialmente la base de datos creada se encuentra vacía.



## Creación de tablas

- El siguiente paso es la creación de las tablas.
- Una tabla está constituida por un conjunto de columnas.
- Cada columna está definida por:
  - Un nombre obligatorio.
  - Un tipo de dato obligatorio y, en el caso del tipo CHAR (cadena de texto con longitud fija) o VARCHAR (cadena de texto con longitud variable), también una longitud.
  - Otras características: clave primaria (primary key), clave foránea (foreign key) para establecer relaciones entre tablas, o autoincremental (A\_I), entre otras.
- A continuación se detallan cada uno de los pasos para crear las siguientes dos tablas:

**Tabla cliente**

Columna	Tipo	Otro
id	INT	PK y AI
id_gestor	INT	FK
usuario	CHAR(20)	
password	CHAR(64)	
correo	CHAR(50)	
saldo	DOUBLE	

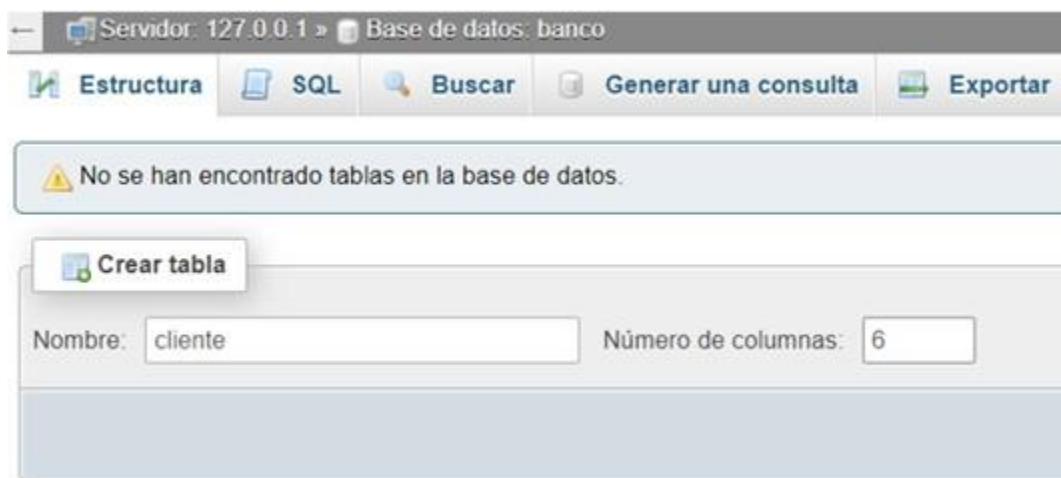


**PK → clave primaria**  
**FK → clave foránea**  
**AI → autoincremental**

**Tabla gestor**

Columna	Tipo	Otro
id	INT	PK y AI
usuario	CHAR(20)	
password	CHAR(64)	
correo	CHAR(50)	

- Al comenzar a crear una nueva tabla es necesario establecer su nombre y el número de columnas que tendrá la tabla.
- En este caso, la tabla es cliente y el número de columnas es de 6.
- Tras la creación de la tabla pueden crearse nuevas columnas o modificar/eliminar las ya existentes.



- A continuación se definen los nombres de cada una de las columnas de la tabla usuario, así como sus tipos y longitudes en el caso de columnas de tipo CHAR.

Nombre	Tipo	Longitud/Valores
id	INT	
id_gestor	INT	
usuario	CHAR	20
password	CHAR	64
correo	CHAR	50
saldo	DOUBLE	

- La columna id debe declararse como clave primaria marcando el valor PRIMARY en el apartado de "Índice" de la columna.

- La columna id también se convierte en autoincremental marcando la casilla A\_I.

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento	Atributos	Nulo Índice	A_I Comentarios
id	INT		Ninguno			<input type="checkbox"/> PRIMARY	<input checked="" type="checkbox"/> PRIMARY

Agregar índice

Nombre del índice: PRIMARY  
Opción de índice: PRIMARY  
Columna: id [int]

Continuar Cancelar

- Finalmente, el botón "Guardar" crea la tabla usuario.
- Un resumen de las columnas de la tabla recién creada aparece desde la pestaña "Estructura".

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)		No	Ninguna			AUTO_INCREMENT	<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más
2	id_gestor	int(11)		No	Ninguna			<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más	
3	usuario	char(20)	utf8mb4_general_ci	No	Ninguna			<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más	
4	password	char(64)	utf8mb4_general_ci	No	Ninguna			<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más	
5	correo	int(50)		No	Ninguna			<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más	
6	saldo	double		No	Ninguna			<input type="button"/> Cambiar <input type="button"/> Eliminar ▾ Más	

- En la tabla anterior puede declararse las columnas usuario y correo como índices de tipo único (UNIQUE).
- Si se declaran varias columnas como de clave única, entonces la clave única es de campo múltiple (por eso es mejor declarar usuario y correo como de tipo UNIQUE).
- Pueden consultarse los datos de la tabla desde la pestaña "Examinar".
- Inicialmente la tabla usuario no presenta ningún registro.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with buttons for 'Examinar' (highlighted with a red circle), Estructura, SQL, Buscar, Insertar, Exportar, Importar, Privilegios, Operaciones, Seguimiento, and Disparadores. Below the toolbar, a message box says: "MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0009 segundos.)". The SQL query shown is: "SELECT \* FROM `cliente`". At the bottom, there's a section titled "Operaciones sobre los resultados de la consulta" with a "Crear vista" button.

- ♦ El procedimiento para crear la tabla gestores es similar.

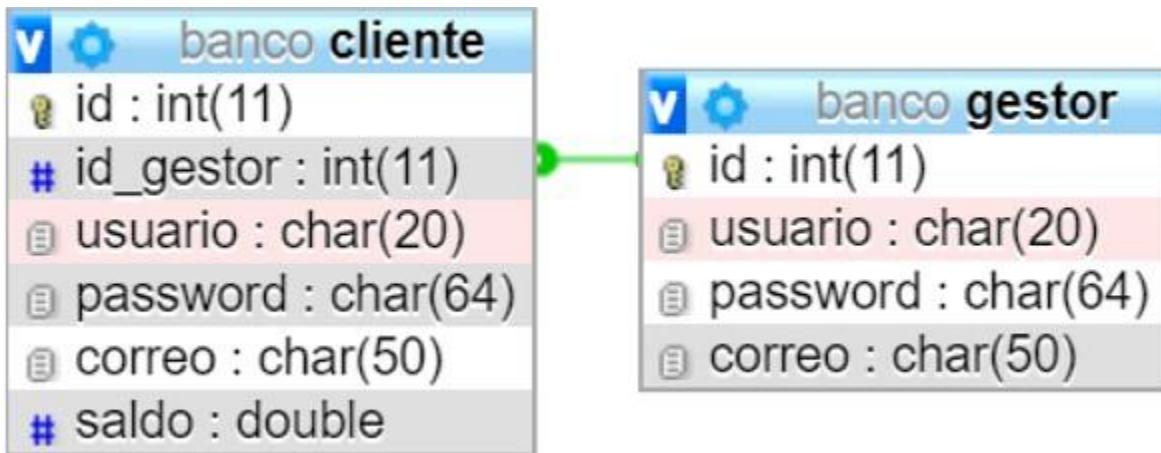
The screenshot shows the MySQL Workbench interface. At the top, there's a "Crear tabla" (Create Table) dialog with "Nombre: gestor" and "Número de columnas: 4". Below it, the table structure is defined:

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento	Atributos	Nulo	índice	A.J.
id	INT		Ninguno			PRIMARY		PRIMARY
usuario	CHAR	20	Ninguno			---		
password	CHAR	64	Ninguno			---		
correo	CHAR	50	Ninguno			---		

At the bottom, the table structure is listed in a table:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	<b>id</b>	int(11)		No	Ninguna			AUTO_INCREMENT	Cambiar  Eliminar  Más
2	<b>usuario</b>	char(20)	utf8mb4_general_ci	No	Ninguna				Cambiar  Eliminar  Más
3	<b>password</b>	char(64)	utf8mb4_general_ci	No	Ninguna				Cambiar  Eliminar  Más
4	<b>correo</b>	char(50)	utf8mb4_general_ci	No	Ninguna				Cambiar  Eliminar  Más

- ♦ Por último, faltan definir las relaciones entre las tablas.
- ♦ La columna id\_gestor debe declararse como clave foránea para que apunte a la columna id de la tabla gestor.



- Para establecer una clave foránea para una columna de la tabla usuario se accede a la pestaña “Estructura” y se pulsa sobre el botón de “Vista de relaciones”.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar  Eliminar  Más
2	id_gestor	int(11)			No	Ninguna		Cambiar  Eliminar  Más	
3	usuario	char(20)	utf8mb4_general_ci		No	Ninguna		Cambiar  Eliminar  Más	
4	password	char(64)	utf8mb4_general_ci		No	Ninguna		Cambiar  Eliminar  Más	
5	correo	int(50)			No	Ninguna		Cambiar  Eliminar  Más	
6	saldo	double			No	Ninguna		Cambiar  Eliminar  Más	

Seleccionar todo    Para los elementos que están marcados:  Examinar Cambiar Eliminar Primaria Único Índice Texto completo  
 Agregar a columnas centrales  Eliminar de las columnas centrales

- Para establecer la clave foránea hay que definir:
  - La columna de la tabla actual que será clave foránea.
  - La columna de la otra tabla a la que apuntará (base de datos, tabla y nombre de la columna).

Acciones	Propiedades de la restricción	Columna	Restricción de clave foránea (INNODB)	Base de datos	Tabla	Columna
Nombre de la restricción	ON DELETE RESTRICT	ON UPDATE RESTRICT	id_gestor	banco	gestor	id
+ Añadir restricción			+ Anadir columna			
+ Relaciones internas						

- Pulsando el botón “Guardar” se creará la clave foránea.
- La tabla usuario presenta finalmente la siguiente estructura de columnas.

Estructura de tabla Vista de relaciones

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	<b>id</b>	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar  Eliminar  Más
2	<b>id_gestor</b>	int(11)			No	Ninguna			Cambiar  Eliminar  Más
3	<b>usuario</b>	char(20)	utf8mb4_general_ci		No	Ninguna			Cambiar  Eliminar  Más
4	<b>password</b>	char(64)	utf8mb4_general_ci		No	Ninguna			Cambiar  Eliminar  Más
5	<b>correo</b>	int(50)			No	Ninguna			Cambiar  Eliminar  Más
6	<b>saldo</b>	double			No	Ninguna			Cambiar  Eliminar  Más

↑  Seleccionar todo Para los elementos que están marcados: Examinar Cambiar Eliminar Primaria Único Índice Texto  
 Agregar a columnas centrales Eliminar de las columnas centrales

---

Imprimir Planteamiento de la estructura de tabla Hacer seguimiento a la tabla Mover columnas Normalizar

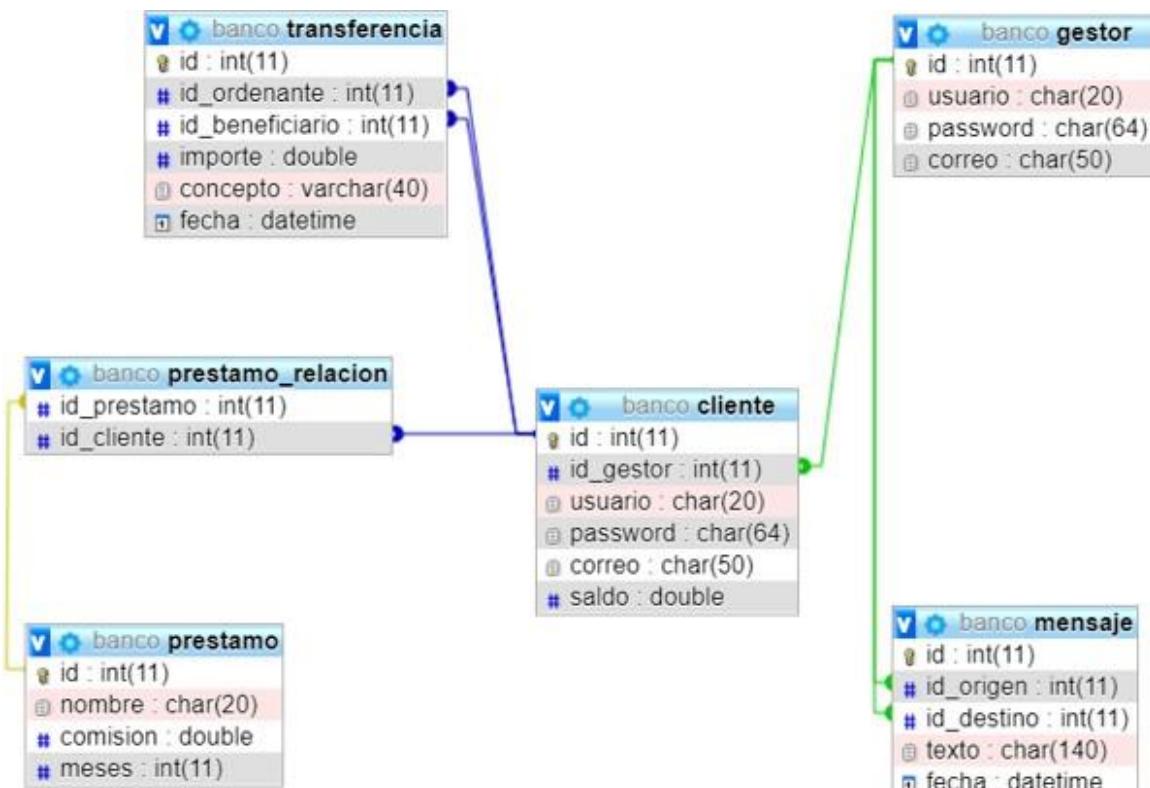
Agregar 1 columna(s) después de saldo Continuar

Índices

Acción	Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
Editar  Eliminar	PRIMARY	BTREE	Sí	No	id	0	A	No	
Editar  Eliminar	id_gestor	BTREE	No	No	id_gestor	0	A	No	

Crear un índice en 1 columna(s) Continuar

Ejercicio: crear el resto de tablas de la base de datos.



## Manejo de datos

- El manejo de datos en bases de datos se realiza mediante operaciones CRUD:
  - C: crear (create)
  - R: leer (read)

- U: actualizar (update)
- D: eliminar (delete)



- La creación de datos en una tabla se realiza en phpMyAdmin desde la pestaña "Insertar".

← Servidor: 127.0.0.1 » Base de datos: banco » Tabla: cliente

Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0005 segundos.)

```
SELECT * FROM `cliente`
```

Operaciones sobre los resultados de la consulta

Crear vista

- La inserción de los datos se realiza mediante campos de formulario para cada una de las columnas de la tabla.

Columna	Tipo	Función	Nulo	Valor
id	int(11)		1	
id_gestor	int(11)		1	<input type="button" value="Mostrar los valores foráneos"/>
usuario	char(20)		pepito	
password	char(64)		627a6ab55608711e421a1132be531a7a7fec1df7b25 3dda73ec482824269e5d7	<input type="button" value=""/>
correo	char(50)		pepito@correo.com	
saldo	double		500	

- La inserción de datos en la tabla cliente no es posible hasta que exista al menos un gestor (debido a la referencia entre la columna id\_gestor de la tabla cliente y la columna id de la tabla gestor).



- Por tanto, es necesario que exista al menos un registro en la tabla gestor para poder introducir datos en la tabla cliente.

Columna	Tipo	Función	Nulo	Valor
id	int(11)			
usuario	char(20)			gestor1
password	char(64)			d94d9ca1462655945362f71cdd0bbb31e4033e50 462bf50a35f745b7b43eea84
correo	char(50)			gestor1@correo.com

[Continuar](#)

+ Opciones		← →	▼	id	usuario	password	correo
<input type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>	1	gestor1	d94d9ca1462655945362f71cdd0bbb31e4033e50462bf50a35...	gestor1@correo.com
<input checked="" type="checkbox"/>	<a href="#">Editar</a>	<a href="#">Copiar</a>	<a href="#">Borrar</a>				
<input type="checkbox"/>	<a href="#">Exportar</a>						

- Tras la inserción de al menos un gestor ya es posible crear registros en la tabla del cliente.

Columna	Tipo	Funcióñ	Nulo	Valor
id	int(11)			
id_gestor	int(11)			gestor1 - 1
usuario	char(20)			pepito
password	char(64)			627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7
correo	char(50)			pepito@correo.com
saldo	double			2000

**Continuar**

 Ignorar

Columna	Tipo	Funcióñ	Nulo	Valor
id	int(11)			
id_gestor	int(11)			gestor1 - 1
usuario	char(20)			juanito
password	char(64)			44e235daceb8ee492f6bb5249ffb0aecc993a330d8c1fd9cf3... 1fd9cf30447deae1ecd3
correo	char(50)			juanito@correo.com
saldo	double			1000

**Continuar**

- Para leer los registros de una tabla es suficiente con acceder a la pestaña "Examinar".

**Examinar** Estructura SQL Buscar Insertar Exportar Importar Privilegios Operaciones Seguimiento

Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0006 segundos.)

```
SELECT * FROM `cliente`
```

Mostrar todo Número de filas: 25 Filtrar filas: Buscar en esta tabla Ordenar según la clave: Ninguna

+ Opciones

	id	id_gestor	usuario	password	correo	saldo
<input type="checkbox"/>	1	1	pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7	pepito@correo.com	2000
<input type="checkbox"/>	2	1	juanito	44e235daceb8ee492f6bb5249ffb0aecc993a330d8c1fd9cf3... 1fd9cf30447deae1ecd3	juanito@correo.com	1000

↑  Seleccionar todo Para los elementos que están marcados:

- La actualización de datos puede realizarse de dos formas:
  - Haciendo doble click sobre el valor de la columna a modificar.
  - Pulsando el enlace "Editar" asociado al registro.

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

+ Opciones

		id	id_gestor	usuario	password	correo	saldo
<input type="checkbox"/>	<input type="checkbox"/> Editar	<input type="checkbox"/> Copiar	<input type="checkbox"/> Borrar	1	1 pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7	pepito@correo.com 2000
<input type="checkbox"/>	<input type="checkbox"/> Editar	<input type="checkbox"/> Copiar	<input type="checkbox"/> Borrar	2	1 juanito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7	juanito@correo.com 1000

← Seleccionar todo Para los elementos que están ma

Mostrar todo | Número de filas: 25 | Filtrar fila

Operaciones sobre los resultados de la consulta

Imprimir  Copiar al portapapeles  Exportar  Mostrar grá

Guardar esta consulta en favoritos

Presione la tecla de escape para cancelar la edición.

Etiqueta:  Permitir que todo usuario pueda acceder a este favorito

Examinar | Estructura | SQL | Buscar | Insertar | Exportar | Importar | Privilegios | Operaciones | Seguimiento

✓ Mostrando filas 0 - 1 (total de 2, La consulta tardó 0.0006 segundos.)

```
SELECT * FROM `cliente`
```

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

+ Opciones

		id	id_gestor	usuario	password	correo	saldo
<input type="checkbox"/>	<input type="checkbox"/> Editar	<input type="checkbox"/> Copiar	<input type="checkbox"/> Borrar	1	1 pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7	pepito@correo.com 2000
<input checked="" type="checkbox"/>	<input type="checkbox"/> Editar	<input type="checkbox"/> Copiar	<input type="checkbox"/> Borrar	2	1 juanito	44e235daceb8ee492f6bb5249ff0aecc993a330d8c1fd9cf3	juanito@correo.com 1000

← Seleccionar todo Para los elementos que están marcados:  Editar  Copiar  Borrar  Exportar

Columna Tipo Función Nulo Valor

id	int(11)		1
id_gestor	int(11)		1 - gestor1
usuario	char(20)		pepito
password	char(64)		627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7
correo	char(50)		pepito@correo.com
saldo	double		2000

Continuar

- La eliminación de datos se realiza mediante el enlace de "Borrar" asociado al registro.

Examinar | Estructura | SQL | Buscar | Insertar | Exportar | Importar | Privilegios | Operaciones | Seguimiento

Mostrando filas 0 - 1 (total de 2, La consulta tardó 0,0006 segundos.)

```
SELECT * FROM `cliente`
```

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

+ Opciones

	id	id_gestor	usuario	password	correo	saldo
<input type="checkbox"/>	1	1	pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73e...	pepito@correo.com	2000
<input checked="" type="checkbox"/>	1	1	juanito	44e235daceb8ee492f6bb5249ffb0aecc993a330d8c1fd9cf3...	juanito@correo.com	1000

← Seleccionar todo Para los elementos que están marcados:

+ Opciones

	id	id_gestor	usuario	password	correo	saldo
<input type="checkbox"/>	1	1	pepito	627a6ab55608711e421a1132be531a7a7fec1df7b253dda73e...	pepito@correo.com	2000

← Seleccionar todo Para los elementos que están marcados:

- Desde la pestaña “Operaciones” de una tabla se puede renombrarla, copiarla, vaciarla (TRUNCATE) o borrarla (DROP), entre otras funciones.

Servidor: 127.0.0.1 » Base de datos: banco » Tabla: gestor

Examinar | Estructura | SQL | Buscar | Insertar | Exportar | Importar | Privilegios | **Operaciones**

Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0005 segundos.)

```
SELECT * FROM `gestor`
```

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla

+ Opciones

	id	usuario	password	correo
<input type="checkbox"/>	1	gestor1	d94d9ca1462655945362f71cdd0bbb31e4033e50462bf50a35...	gestor1@correo.com

← Seleccionar todo Para los elementos que están marcados:

- Desde la pestaña “Operaciones” de una base de datos se puede renombrarla, eliminarla o copiarla, entre otras funciones.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with buttons for Estructura, SQL, Buscar, Generar una consulta, Exportar, Importar, Operaciones (which is circled in red), Privilegios, Rutinas, and Eventos. Below the toolbar is a search bar labeled 'Que contengan la palabra:' and a 'Filtros' button. The main area displays a table of database structures:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
cliente	Examinar Estructura Buscar Insertar Vaciar Eliminar	32.0	InnoDB	utf8mb4_general_ci	32.0 KB	-
gestor	Examinar Estructura Buscar Insertar Vaciar Eliminar	16.0	InnoDB	utf8mb4_general_ci	16.0 KB	-
mensaje	Examinar Estructura Buscar Insertar Vaciar Eliminar	48.0	InnoDB	utf8mb4_general_ci	48.0 KB	-
prestamo	Examinar Estructura Buscar Insertar Vaciar Eliminar	16.0	InnoDB	utf8mb4_general_ci	16.0 KB	-
prestamo_relacion	Examinar Estructura Buscar Insertar Vaciar Eliminar	48.0	InnoDB	utf8mb4_general_ci	48.0 KB	-
transferencia	Examinar Estructura Buscar Insertar Vaciar Eliminar	48.0	InnoDB	utf8mb4_general_ci	48.0 KB	-
<b>6 tablas</b>	<b>Número de filas</b>				<b>268.0 KB</b>	<b>0 B</b>

At the bottom left are buttons for 'Seleccionar todo' and 'Para los elementos que están marcados: ▾'. On the right side of the table are buttons for 'Filas', 'Tipo', 'Cotejamiento', 'Tamaño', and 'Residuo a depurar'.

- El botón “Exportar” permite descargar en un archivo SQL todos los datos y la estructura de todas las tablas de una base de datos.

The screenshot shows the phpMyAdmin interface. On the left, there's a sidebar with a tree view of databases: Nueva, banco (which is circled in red), information\_schema, mysql, performance\_schema, phpmyadmin, and test. Above the tree view are buttons for Reciente and Favoritas. On the right, there's a main panel for the 'banco' database, showing the same toolbar and search/filter fields as the MySQL Workbench screenshot above. The 'Exportar' button in the toolbar is also circled in red.

- Existe gran cantidad de opciones a elegir en la exportación, pero pueden dejarse las establecidas por defecto y pulsar el botón “Continuar”.

## Exportando tablas de la base de datos "banco"

### Exportar plantillas:

Nueva plantilla:

Nombre de plantilla

**Crear**

Plantillas existentes:

Plantilla:

– Seleccionar plantilla – ▾

**Actualizar**

**Borrar**

### Método de exportación:

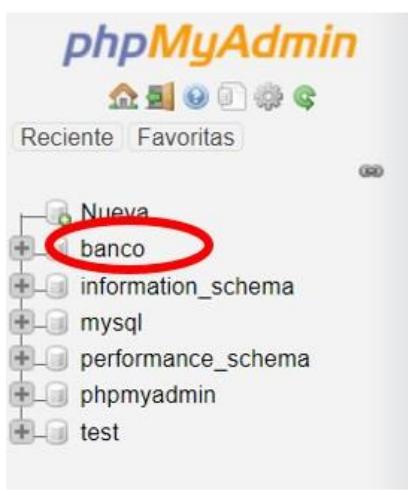
- Rápido - mostrar sólo el mínimo de opciones de configuración
- Personalizado - mostrar todas las opciones de configuración posibles

### Formato:

SQL

**Continuar**

- Para importar es necesario que no existan tablas en la base de datos donde se cargarán los datos y las tablas desde el archivo SQL.



The screenshot shows the phpMyAdmin interface. On the left, there is a tree view of databases: Nueva, banco (which is selected and highlighted with a red circle), information\_schema, mysql, performance\_schema, phpmyadmin, and test. On the right, the main panel shows the following details:

- Servidor: 127.0.0.1 » Base de datos: banco
- Estructura, SQL, Buscar, Generar una consulta, Exportar, Importar (the Importar button is circled in red)
- A message: "No se han encontrado tablas en la base de datos."
- A "Crear tabla" form with fields: Nombre: \_\_\_\_\_, Número de columnas: 4

- En el siguiente paso se selecciona el archivo SQL anteriormente exportado y se pulsa el botón "Continuar".

**Archivo a importar:**

El archivo puede ser comprimido (gzip, bzip2, zip) o descomprimido.  
Un archivo comprimido tiene que tener la extensión [formato].[compresión]. Por ejemplo: .sql.zip

Buscar en su ordenador:  Ningún archivo seleccionado (Máximo: 40MB)

También puede arrastrar un archivo en cualquier página.

Conjunto de caracteres del archivo:

**Importación parcial:**

Permitir la interrupción de una importación en caso que el script detecte que se ha acercado al límite de tiempo PHP.

Omitir esta cantidad de consultas (en SQL) desde la primera:

**Otras opciones:**

Habilite la revisión de las claves foráneas

**Formato:**

**Opciones específicas al formato:**

Modalidad SQL compatible:

No utilizar auto\_INCREMENT con el valor 0

Ejercicio: escribir dos clases en Java que representen a la tabla gestor y a la tabla cliente.

Clase Cliente	
	id
	id_gestor
	usuario
	password
	correo
	saldo

Métodos getters y setters

Clase Gestor	
	id
	usuario
	password
	correo

Métodos getters y setters

## El lenguaje de consultas SQL

- Aunque el manejo de datos mediante phpMyAdmin es muy útil para explorar los datos y las tablas, en la práctica no es muy utilizado.
- En programación es más habitual hacer uso del lenguaje de consultas estructurado SQL para realizar operaciones CRUD.

- SQL es el estándar de facto en la actualidad para la inmensa mayoría de los bases de datos



## Crear datos

- El juego de instrucciones del lenguaje SQL está dividido en tres grandes grupos: sentencias para la creación de objetos, sentencias para el control de seguridad y sentencias para la manipulación de datos (siendo éstas últimas las más importantes).
- En phpMyAdmin pueden ejecutarse instrucciones SQL para una base de datos desde la pestaña "SQL".



- La sentencia INSERT de SQL permite la inserción de uno o más registros en una tabla de una base de datos.
- La sintaxis para insertar un único registro es la siguiente:

```
INSERT INTO tabla (columna1, columna2,...) VALUES (valor1, valor2, ...);
```

- La sintaxis para insertar varios registros es la siguiente.

```
INSERT INTO tabla (columna1, columna2,...) VALUES (valor1, valor2, ...), (valor1, valor2, ...);
```

- ♦ Ejemplo de inserción de un único registro.

```
INSERT INTO gestor (usuario, password, correo) VALUES ('gestor2', 'gestor2',  
'gestor2@correo.com');
```

- ♦ Ejemplo de inserción de varios registros.

```
INSERT INTO gestor (usuario, password, correo) VALUES ('gestor3', 'gestor3',  
'gestor3@correo.com'), ('gestor4', 'gestor4', 'gestor4@correo.com'), ('gestor5', 'gestor5',  
'gestor5@correo.com');
```

## Leer datos

- ♦ La sentencia SELECT de SQL permite recuperar valores almacenados en una base de datos, tanto en una tabla como en varias. Su sintaxis es la siguiente.

```
SELECT columna1, columna2,... FROM tabla WHERE condición/condiciones ORDER BY  
columna1;
```

- ♦ Con SELECT se obtienen todos los registros de la tabla indicada en la cláusula FROM, formados por las columnas indicadas en SELECT, que cumplan la condición o condiciones establecidas en WHERE y ordenadas según el campo o campos definidos en ORDER BY.
- ♦ Ejemplos de SELECT.

```
SELECT id, usuario FROM gestor WHERE id>1 ORDER BY correo;
```

```
SELECT * FROM gestor WHERE id>2 ORDER BY correo;
```

- ♦ Las condiciones de filtro se establecen en la cláusula WHERE mediante una expresión condicional del tipo.

```
columna operador valor
```

- ♦ Donde columna es el nombre del campo de la tabla indicada en el FROM sobre el que se establece la condición, operador es el operador condicional que determina el tipo de

comparación a realizar y valor el dato con el que se comparará el campo.

- Los operadores que pueden utilizarse en WHERE son:

Operador	Significado
<	Menor que
>	Mayor que
<>	Distinto que
<=	Menor o igual que
>=	Mayor o igual que
=	Igual que
LIKE	Como
BETWEEN	Entre
IN	En

- Ejemplo de SELECT con el operador <>

```
SELECT id, usuario FROM gestor WHERE id<>1;
```

- Ejemplo de SELECT con el operador <=

```
SELECT id, usuario FROM gestor WHERE id<=1;
```

- El operador LIKE se utiliza para realizar comparaciones en cadenas de texto.
- Ejemplos de SELECT con los operadores = y LIKE (ambos devuelven los mismos resultados).

```
SELECT id, usuario FROM gestor WHERE usuario LIKE 'gestor1';
```

```
SELECT id, usuario FROM gestor WHERE usuario = 'gestor1';
```

- Con el operador LIKE se pueden utilizar caracteres especiales en la comparación.

Carácter especial	Significado
%	Cero, uno o más caracteres
_	Carácter no nulo
[x-y]	Carácter dentro del rango x-y

Valor de ejemplo	Posibles resultados de la consulta
J%	Jennifer Warnes, Joni Mitchell, Jose Carreras
%Spark	Court and Spark
%Blue%	Famous Blue Raincoat, Blue, Blues on the Bayou
%Cline%Hits	Patsy cline: 12 Greatest Hits
194_	1940, 1941, 1947
19_	1900, 1907, 1938, 1963, 1999
9_3%	9032343, 903, 95312, 99306, 983393300333

- El operador BETWEEN comprueba si un valor está comprendido entre dos proporcionados.
- Ejemplos de SELECT con el operador BETWEEN.

```
SELECT id, usuario FROM gestor WHERE id BETWEEN 1 AND 2;
```

```
SELECT id, usuario FROM gestor WHERE usuario BETWEEN 'gestor' AND 'gestor2';
```

- El operador IN comprueba si un valor está incluido en una lista de valores.
- Ejemplos de SELECT con el operador IN.

```
SELECT id, usuario FROM gestor WHERE id IN (1,2,4);
```

```
SELECT id, usuario FROM gestor WHERE usuario IN ('gestor1','gestor2','gestor10')
```

- Los operadores lógicos OR y AND también pueden utilizarse para varias condiciones en el WHERE.

```
SELECT id, usuario FROM gestor WHERE id > 1 AND id <=3;
```

```
SELECT id, usuario FROM gestor WHERE id = 1 OR id =3;
```

- Por último, para obtener el número de registros de una tabla puede utilizarse la función COUNT.

```
SELECT COUNT(*) FROM gestor;
```

## Actualizar datos

- La sentencia UPDATE de SQL se emplea para modificar los valores de ciertas columnas en aquellos registros que cumplan una determinada condición o condiciones establecidas con WHERE. Su sintaxis es la siguiente:

```
UPDATE tabla SET columna1 = expr1, columna2 = expr2,... WHERE  
condición/condiciones
```

- La cláusula SET establece los valores que se asignarán a cada columna. Estos valores pueden ser cualquier expresión que devuelva como resultado un dato compatible con el tipo soportado por la columna.
- Ejemplos de UPDATE:

```
UPDATE gestor SET correo = 'gestor3a@gmail.com' WHERE id=3;
```

```
UPDATE gestor SET correo = 'gestor3@gmail.com', id=8 WHERE id=3 AND  
usuario='gestor3'
```

## Eliminar datos

- La sentencia DELETE de SQL se utiliza para eliminar un conjunto de registros de una tabla a partir de una determinada condición o condiciones establecidas con WHERE. Su sintaxis es la siguiente:

```
DELETE FROM tabla WHERE condición/condiciones
```

- ◆ Ejemplo de DELETE.

```
DELETE FROM gestor WHERE id=3;
```

Tu carrera digital ~

# Módulo 4

# Bases de datos (SQL)

JDBC

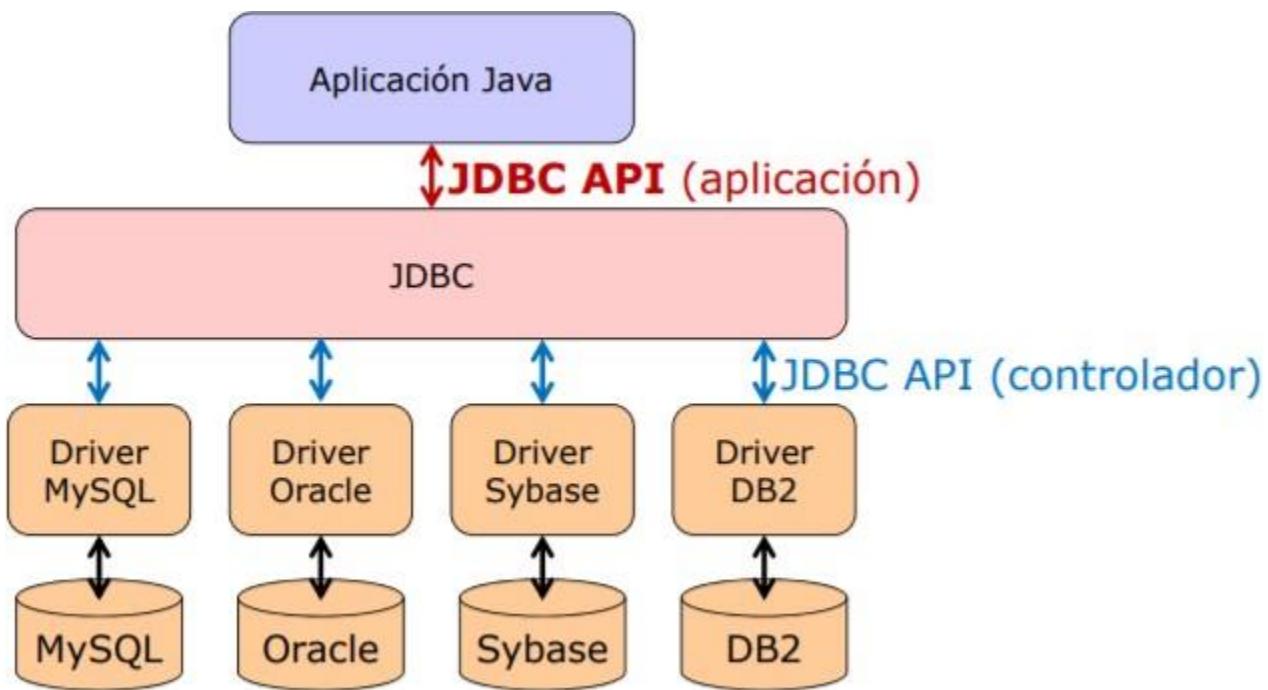


# JDBC

- ◆ **Introducción e instalación**
- ◆ **Establecimiento y liberación de la conexión con la base de datos**
- ◆ **Ejecutar operaciones CRUD**
- ◆ **Inyecciones SQL**
- ◆ **Instrucciones preparadas**
- ◆ **Transacciones**

## Introducción e instalación

- ◆ Utilizar JDBC para conectar con un sistema de base de datos es poco habitual actualmente en Java debido a que existen herramientas ORM (Mapeo Objeto-Relacional) más poderosas como iBatis e Hibernate. Sin embargo, JDBC puede ser útil para:
  - Su uso en pequeñas aplicaciones.
  - Aprender los conceptos básicos de comunicación con un sistema de base de datos, del lenguaje SQL y de cómo se ejecutan operaciones CRUD.
  - Facilitar el aprendizaje del manejo de bases de datos con otros lenguajes de programación puesto que todos disponen de librerías similares a JDBC lanzando instrucciones SQL. iBatis e Hibernate no están disponibles para otros lenguajes de programación.
  - Los ORM se encuentran construidos sobre JDBC y construyen las instrucciones SQL de forma dinámica, por lo que utilizar JDBC directamente es siempre más rápido que hacer uso de un ORM como Hibernate.
- ◆ Asimismo, es importante conocer el lenguaje SQL porque determinadas operaciones con la base de datos (copias de seguridad, restauración de datos, exploración de datos, pequeñas operaciones CRUD) son más simples de realizar con SQL que con herramientas ORM como Hibernate.
- ◆ JDBC o Java Database Connectivity (Conectividad a bases de datos de Java) es un conjunto de clases en Java que permite la ejecución de operaciones sobre sistemas de bases de datos, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede.



- Los drivers JDBC son adaptadores que convierten la petición procedente del programa Java a un protocolo específico de un sistema de base de datos.
- El driver de MySQL para JDBC puede descargarse [desde la página oficial](#).
  - El sistema operativo a seleccionar es "Platform Independent".
  - Puede descargarse el archivo comprimido en formato TAR o ZIP indistintamente.
  - Dentro del comprimido se encuentra un archivo JAR, que debe ser importado en el proyecto como una librería normal de Java.

## ① MySQL Community Downloads

◀ Connector/

**General Availability (GA) Releases**

Select Operating System: Platform Independent ▾

Looking for previous GA versions?

<b>Platform Independent (Architecture Independent), Compressed TAR Archive</b> (mysql-connector-java-8.0.18.tar.gz)	8.0.18	3.7M	<b>Download</b>
<b>Platform Independent (Architecture Independent), ZIP Archive</b> (mysql-connector-java-8.0.18.zip)	8.0.18	4.4M	<b>Download</b>

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

mysql-connector-java-8.0.18.tar.gz

Archivo Órdenes Herramientas Favoritos Opciones Ayuda

Añadir Extraer en Comprobar Ver Eliminar Buscar Asistente Información Buscar virus

mysql-connector-java-8.0.18.tar.gz\mysql-connector-java-8.0.18 - archivo TAR+GZIP, tamaño descomprimido 13.835.613 bytes

Nombre

- ..
- src
- mysql-connector-java-8.0.18.jar
- build.xml
- CHANGES
- INFO\_BIN
- INFO\_SRC
- LICENSE
- README

▼ HelloWorld

- > JRE System Library [JavaSE-11]
- > src
- > Referenced Libraries
- ▼ lib
- mysql-connector-java-8.0.18.jar

Establecimiento y liberación de la conexión con la base de datos

- En versiones antiguas de Java el primer paso era registrar el driver mediante Class.forName, pasando por parámetros el String que identifica a MySQL (com.mysql.jdbc.Driver).

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

- Actualmente ya no es necesario registrar el driver y puede abrirse directamente la conexión con la base de datos.
- Para abrir la conexión con la base de datos es necesario establecer el String JDBC de conexión, que posee los siguientes campos.
  - jdbc: indica que la conexión es de tipo jdbc.
  - subprotocolo: es el tipo de sistema de base de datos utilizada. En este caso su valor es mysql.
  - localizador: indica la dirección IP, puerto y nombre de la base de datos a la que conectar.

```
jdbc:subprotocol://localizadorBD
```

- El String JDBC de conexión es pasado por argumento como primer parámetro al método DriverManager.getConnection.
  - El segundo parámetro pasado es el nombre del usuario.
  - El tercer parámetro pasado es el password.

```
try {  
    /*  
     * subprotocolo: mysql  
     * host: localhost  
     * puerto: 3306  
     * base de datos: banco  
     * usuario: banco  
     * contraseña: banco  
    */  
    Connection conexion =  
    DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");  
  
    // muestra información del tipo de sistema de base de datos (MySQL)  
    System.out.println("Base de datos: " +  
        conexion.getMetaData().getDatabaseProductName());  
  
    // muestra información sobre la versión del sistema de base de datos (5.5.5-10.4.11-  
    MariaDB)  
    System.out.println("Versión: " +  
        conexion.getMetaData().getDatabaseProductVersion());
```

```
// muestra información del driver MySQL (MySQL Connector/J)
System.out.println("Driver: " + conexion.getMetaData().getDriverName());

// muestra información de la versión del driver MySQL (mysql-connector-java-8.0.18)
System.out.println("Versión del driver: " +
conexion.getMetaData().getDriverVersion());

} catch (SQLException e) {
    e.printStackTrace();
}
```

- ◆ La liberación de recursos es importante y debe realizarse dentro de la cláusula finally en el manejo de excepciones.
- ◆ Debe invocarse al método close, y en el siguiente orden, todas las instrucciones (objetos de clase Statement), conjunto de resultados (objetos de clase ResultSet) y conexiones (objetos de clase Connection).

```
Statement instrucion = null;
ResultSet resultados = null;
Connection conexion = null;

try {

    conexion = DriverManager.getConnection("jdbc:mysql://localhost:3306/banco",
"banco", "banco");

/* .. */

} catch (SQLException e) {
    e.printStackTrace();
} finally {
    try {

        // libera los resultados
        if (resultados != null) {
            resultados.close();
        }

        // libera la instrucción
        if (instrucion != null) {
            instrucion.close();
        }

        // libera la conexión
        if (conexion != null) {
            conexion.close();
        }

    }
}
```

```
} catch (Exception e) {  
    e.printStackTrace();  
}  
}
```

## Ejecutar operaciones CRUD

- Para ejecutar operaciones CRUD es necesario obtener un objeto de tipo Statement a partir del método createStatement del objeto de conexión.

```
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");  
Statement instrucion = conexion.createStatement();
```

- Una vez obtenido el objeto de tipo Statement, puede invocarse a alguno de los siguientes métodos para ejecutar instrucciones SQL sobre el sistema de base de datos:
  - executeUpdate(String sql): para ejecución de consultas INSERT, UPDATE, DELETE. Devuelve el número de registros afectados.
  - executeQuery(String sql): para ejecución de consultas SELECT. Devuelve un objeto de tipo ResultSet.
  - execute(String sql): para la ejecución de todo tipo de consultas. Devuelve true si el resultado es de tipo ResultSet o false si es una inserción, actualización o eliminación. Posteriormente se requiere de la invocación de métodos como getResultset o getUpdateCount del objeto de tipo Statement.

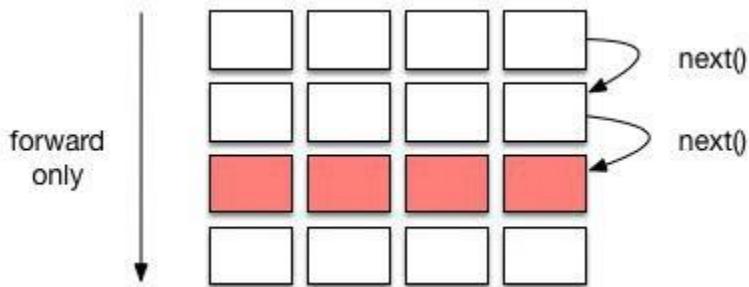
```
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");  
  
// obtención de un objeto de tipo Statement  
Statement instrucion = conexion.createStatement();  
  
// construir sentencia SQL  
String query = "INSERT INTO gestor (usuario, password, correo) VALUES ('gestor1',  
'gestor1', 'gestor1@correo.com')";  
  
// ejecutar instrucción con el método execute  
boolean resultado = instrucion.execute(query);  
  
// si es false, entonces la instrucción no devuelve un objeto de tipo ResultSet  
if (!resultado) {  
  
    // mostrar el número de registros insertados  
    System.out.println("Registros insertados: " + instrucion.getUpdateCount());  
}  
  
// ejecutar instrucción con el método executeUpdate
```

```

int registrosInsertados = instrucion.executeUpdate(query);

// mostrar el número de registros insertados
System.out.println("Registros insertados: " + registrosInsertados);
    
```

- ◆ Para las consultas SQL de tipo SELECT es necesario manejar un objeto de tipo ResultSet.
  - La forma de iterar los resultados devueltos es invocando al método next del objeto de tipo ResultSet utilizando un bucle.
  - Los métodos tipo getX permiten obtener cada uno de los valores de las columnas para los registros en cada iteración.



```

Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");

// obtención de un objeto de tipo Statement
Statement instrucion = conexion.createStatement();

// construir sentencia SQL
String query = "SELECT * FROM gestor";
ResultSet resultados1 = instrucion.executeQuery(query);

System.out.println("Listado de gestores (con executeQuery): ");

// ejecutar instrucción con el método executeQuery
while (resultados1.next()) {
    System.out.println("Gestor " + resultados1.getInt("id"));
    System.out.println("Usuario: " + resultados1.getString("usuario"));
    System.out.println("Password: " + resultados1.getString("password"));
    System.out.println("Correo: " + resultados1.getString("correo"));
    System.out.println("...");
}

// ejecutar instrucción con el método execute
boolean resultado = instrucion.execute(query);

// si es true, entonces la instrucción devuelve un objeto de tipo ResultSet
if (resultado) {

    ResultSet resultados2 = instrucion.getResultSet();
    
```

```

System.out.println("Listado de gestores (con execute): ");

while (resultados2.next()) {
    System.out.println("Gestor " + resultados2.getInt("id"));
    System.out.println("Usuario: " + resultados2.getString("usuario"));
    System.out.println("Password: " + resultados2.getString("password"));
    System.out.println("Correo: " + resultados2.getString("correo"));
    System.out.println("...");
}
}

```

- La conversión entre tipos de datos SQL y tipos de datos de Java es expuesta a continuación.

<b>Tipo SQL</b>	<b>Tipo Java</b>
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT	double
REAL	float
DOUBLE	double
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
LONGVARCHAR	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
BINARY	byte []
VARBINARY	byte []

- También se puede obtener información de los metadatos de los resultados devueltos mediante la ejecución del método getMetaData del objeto de tipo ResultSet.

```
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");  
  
Statement instrucion = conexion.createStatement();  
String query = "SELECT * FROM gestor";  
ResultSet resultados = instrucion.executeQuery(query);  
  
// devuelve el número de columnas  
System.out.println(resultados.getMetaData().getColumnCount());  
  
// devuelve el nombre de la primera columna  
System.out.println(resultados.getMetaData().getColumnName(1));  
  
// devuelve el tipo de dato de la primera columna (un número entero)  
// https://docs.oracle.com/javase/8/docs/api/constant-values.html#java.sql  
System.out.println(resultados.getMetaData().getColumnType(1));
```

## Inyecciones SQL

- Una incorrecta implementación en el acceso a la base de datos puede exponer al sistema a una vulnerabilidad explotada mediante un ataque de inyección SQL.
- Las inyecciones SQL son una de las técnicas de hacking más importantes en aplicaciones Web.
- La vulnerabilidad se produce cuando un programa construye una sentencia SQL en tiempo de ejecución de forma poco segura.
- El pirata informático aprovecha la vulnerabilidad introduciendo fragmentos de código SQL de las distintas maneras que la aplicación web ofrece para que el usuario envíe información al servidor. Por ejemplo, en los campos de formulario.
- El siguiente código presenta una aplicación web simple en Java que es vulnerable a un ataque de inyección SQL. Para explotar la vulnerabilidad puede utilizarse la siguiente instrucción en el campo del usuario del formulario (después de los caracteres -- hay un espacio en blanco):

```
" OR 1=1 --
```

```
// RootHandler.java  
public class RootHandler implements HttpHandler {  
  
    public void handle(HttpExchange he) throws IOException {  
  
        try {
```

```
// respuesta al cliente para peticiones GET (descarga de la página)
if (he.getRequestMethod().equalsIgnoreCase("GET")) {

    String response = "<html><head></head><body><form action=\"\""
method="post\">\r\n"
        + " Usuario: <input type=\"text\" name=\"usuario\"><br>\r\n"
        + " Password: <input type=\"text\" name=\"password\"><br>\r\n"
        + " <input type=\"submit\" value=\"Submit\">\r\n" + "</form></body>
</html>";
    he.sendResponseHeaders(200, response.length());
    OutputStream os = he.getResponseBody();
    os.write(response.getBytes());
    os.close();
}

// respuesta al cliente para peticiones GET (envío de los datos del formulario)
else if (he.getRequestMethod().equalsIgnoreCase("POST")) {

    // obtención de los datos del cuerpo
    BufferedReader in = new BufferedReader(new
InputStreamReader(he.getRequestBody()));
    StringBuilder content = new StringBuilder();
    String line = in.readLine();
    while (line != null) {
        content.append(line);
        line = in.readLine();
    }

    in.close();

    // obtención del usuario y password hasheado
    Map<String, String> parameters = this.queryToMap(content.toString());
    String usuario = parameters.get("usuario");
    String password = parameters.get("password");
    String passwordSHA3 = SHA3(password);

    // conexión a la base de datos
    Connection conexion =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");

    // creación de la instrucción SQL
    Statement instrucion = conexion.createStatement();
    String querySQL = "SELECT * FROM cliente WHERE usuario =\"" + usuario + "\""
AND password =\"" + passwordSHA3 + "\"";
    System.out.println(querySQL);

    // ejecución de la instrucción SQL
    ResultSet resultados = instrucion.executeQuery(querySQL);

    // si la autenticación es correcta, se devuelve ok. En caso contrario, se devuelve
error
```

```
String response = "error";
if (resultados.next()) {
    response = "ok";
}

// envía respuesta al cliente y cierra la conexión
he.sendResponseHeaders(200, response.length());
OutputStream os = he.getResponseBody();
os.write(response.getBytes());
os.close();
}
} catch (Exception e) {
    e.printStackTrace();
}

// extrae los valores del formulario enviados en el cuerpo de la petición POST
public Map<String, String> queryToMap(String query) throws
UnsupportedEncodingException {
    Map<String, String> result = new HashMap<>();
    for (String param : query.split("&")) {
        String[] entry = param.split("=");
        if (entry.length > 1) {
            entry[1] = entry[1].replace("+", " ");
            entry[1] = entry[1].replace("%3D", "=");
            entry[1] = entry[1].replace("%22", "\"");
            result.put(entry[0], entry[1]);
        } else {
            result.put(entry[0], "");
        }
    }
    return result;
}

// calcular el hash SHA3 a partir de un String
public String SHA3(String str) {
    try {
        final MessageDigest digest = MessageDigest.getInstance("SHA3-256");
        final byte[] hashbytes = digest.digest(str.getBytes(StandardCharsets.UTF_8));
        return bytesToHex(hashbytes);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// convierte un array de bytes en hexadecimal a String
private static String bytesToHex(byte[] hash) {
    StringBuffer hexString = new StringBuffer();
    for (int i = 0; i < hash.length; i++) {
        String hex = Integer.toHexString(0xff & hash[i]);
        if (hex.length() == 1)
            hexString.append('0');
        hexString.append(hex);
    }
    return hexString.toString();
}
```

```
    hexString.append('0');
    hexString.append(hex);
}
return hexString.toString();
}
}
```

```
// Main.java
public class Main {

    public static void main(String[] args) {

        try {
            HttpServer server = HttpServer.create(new InetSocketAddress(9000), 0);
            System.out.println("Servidor ejecutándose en el puerto 9000");
            server.createContext("/", new RootHandler());
            server.start();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- La explotación de ataques de inyección SQL sobre campos de formulario puede automatizarse utilizando herramientas como [sqlmap](#).
- Debido a esta problemática, para construir sentencias SQL es mejor utilizar las llamadas instrucciones preparadas (prepared statements), que utilizar una simple concatenación de cadenas de caracteres.

## Instrucciones preparadas

- -Una instrucción preparada o parametrizada permite construir sentencias SQL de forma segura ante ataques de inyección SQL y además es más eficiente en la ejecución que las sentencias SQL construidas a partir de la concatenación.
- Las instrucciones preparadas toman una cadena de caracteres en la cual ciertos valores (representados por el carácter ?) son sustituidos durante la ejecución del programa.

```
Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");

// se inicializa la instrucción preparada, marcando los valores a sustituir con el carácter ?
PreparedStatement ps = conexion.prepareStatement("INSERT INTO gestor(usuario,
```

```

password, correo) VALUES (?, ?, ?)");
// se sustituye la primera aparición del carácter ? con el valor gestor10
ps.setString(1, "gestor10");

// se sustituye la segunda aparición del carácter ? con el valor gestor10
ps.setString(2, "gestor10");

// se sustituye la tercera aparición del carácter ? con el valor gestor10@correo.com
ps.setString(3, gestor10@correo.com);

// se ejecuta la instrucción SQL siguiente:
// INSERT INTO gestor(usuario, password, correo) VALUES ('gestor10', 'gestor10',
'gestor10@correo.com')
if (ps.executeUpdate() != 1) {
    throw new SQLException("Error en la Inserción");
}

System.out.println("Programa finalizado");

```

- Las instrucciones preparadas pueden utilizarse para cualquier sentencia SQL (INSERT, UPDATE, SELECT o DELETE).

```

Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost:3306/banco", "banco", "banco");

// se inicializa la instrucción preparada, marcando los valores a sustituir con el carácter ?
PreparedStatement ps = conexion.prepareStatement("SELECT * from gestor WHERE id
IN (?,?)");

// se sustituye la primera aparición del carácter ? con el valor 3
ps.setInt(1, 3);

// se sustituye la segunda aparición del carácter ? con el valor 4
ps.setInt(2, 4);

// se ejecuta la instrucción SQL siguiente:
// SELECT * from gestor WHERE id IN (3,4)
ResultSet resultados = ps.executeQuery();

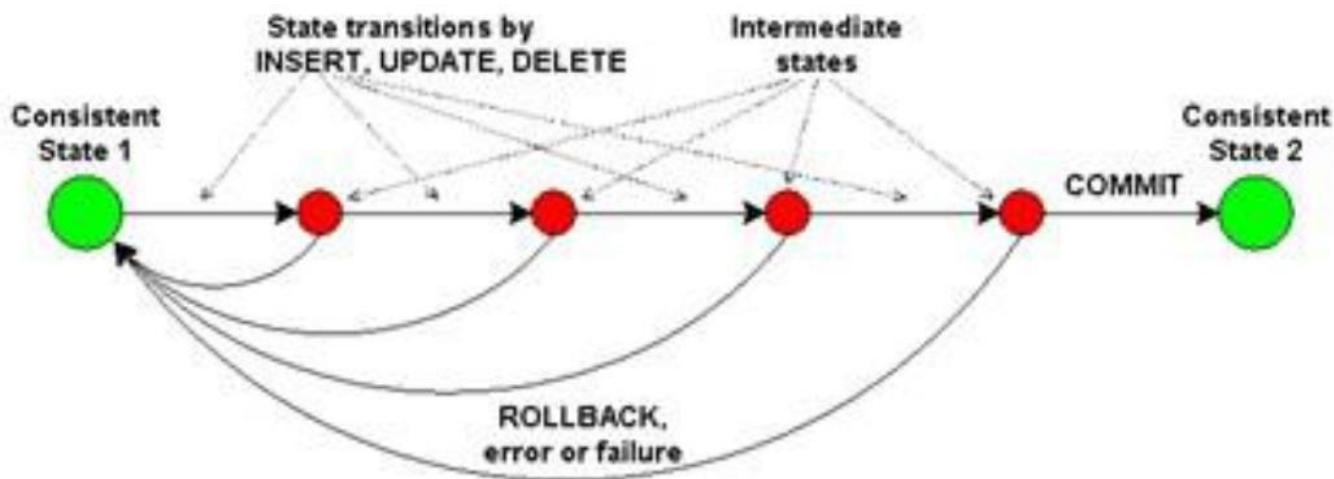
System.out.println("Listado de gestores: ");

while (resultados.next()) {
    System.out.println("Gestor " + resultados.getInt("id"));
    System.out.println("Usuario: " + resultados.getString("usuario"));
    System.out.println("Password: " + resultados.getString("password"));
    System.out.println("Correo: " + resultados.getString("correo"));
    System.out.println("...");
}

```

## Transacciones

- ◆ En entornos multiusuario (por ejemplo, en aplicaciones web) es necesario controlar el acceso concurrente al sistema de base de datos para evitar inconsistencias en los datos.
- ◆ Una transacción es un conjunto de instrucciones SQL que se ejecutan sobre el sistema de base de datos, de tal forma que:
  - Si alguna instrucción SQL falla, entonces ninguna operación del conjunto de instrucciones de la transacción se efectúa.
  - Si todas las instrucciones SQL se ejecutan correctamente, entonces se hacen efectivos los cambios.
- ◆ La clase Connection ofrece dos métodos fundamentales para preparar y ejecutar transacciones:
  - El método commit: ejecuta todas las instrucciones SQL desde el último commit/rollback. Los registros afectados por las instrucciones SQL permanecen bloqueados mientras se ejecuta la transacción (otras operaciones pendientes tienen que esperar).
  - El método rollback: deshace todos los cambios realizados y libera el bloqueo de los registros afectados por la transacción.



- ◆ Es necesario establecer la propiedad autoCommit a false (por defecto está establecido a true) mediante el método setAutoCommit de la clase Connection antes de preparar las instrucciones que contendrá la transacción.

```
Connection conexion = null;
```

```
try {
    conexion = DriverManager.getConnection("jdbc:mysql://localhost:3306/banco",
                                         "banco", "banco");
```

```
// antes de comenzar a utilizar transacciones es fundamental cambiar el valor del
autoCommit a false
conexion.setAutoCommit(false);

// primera sentencia SQL que formará parte de la transacción
PreparedStatement ps1 = conexion.prepareStatement("INSERT INTO gestor(usuario,
password, correo) VALUES (?,?,?) ");
ps1.setString(1, "gestor10");
ps1.setString(2, "gestor10");
ps1.setString(3, gestor10@correo.com);

// segunda sentencia SQL que formará parte de la transacción
PreparedStatement ps2 = conexion.prepareStatement("INSERT INTO gestor(usuario,
password, correo) VALUES (?,?,?) ");
ps2.setString(1, "gestor11");
ps2.setString(2, "gestor11");
ps2.setString(3, gestor11@correo.com);

// tercera sentencia SQL que formará parte de la transacción
PreparedStatement ps3 = conexion.prepareStatement("INSERT INTO gestor(usuario,
password, correo) VALUES (?,?,?) ");
ps3.setString(1, "gestor12");
ps3.setString(2, "gestor12");
ps3.setString(3, gestor12@correo.com);

// cuarta sentencia SQL que formará parte de la transacción y fallará si existe un
registro con id = 4
PreparedStatement ps4 = conexion.prepareStatement("INSERT INTO gestor(id,
usuario, password, correo) VALUES (?,?,?,?) ");
ps4.setInt(1, 4);
ps4.setString(2, "gestor12");
ps4.setString(3, "gestor12");
ps4.setString(4, gestor12@correo.com);

// se añaden todas las sentencias SQL a la transacción
ps1.execute();
ps2.execute();
ps3.execute();
ps4.execute();

// se ejecutan todas las sentencias SQL de la transacción
conexion.commit();

System.out.println("Programa finalizado");

// captura de la excepción SQLException (en el caso en que se produzca)
} catch (SQLException e) {

if (conexion != null) {

try {
```

```
// se realiza un rollback de la transacción, liberando el bloqueo de la base de datos  
conexion.rollback();  
  
System.out.println("Rollback realizado");  
  
// el rollback puede arrojar también una excepción de tipo SQLException  
} catch (SQLException ex) {  
    ex.printStackTrace();  
}  
  
// muestra información si la transacción falló  
e.printStackTrace();  
}  
}
```

Ejercicio: adaptar el anterior ejercicio de la API de Star Wars (el enunciado se encuentra en el tema de Red) para guardar los datos en formato JSON a través de la URL <https://swapi.dev/api/people/1> en una base de datos, en lugar de utilizar ficheros.

Ejercicio proyecto (Main21): modifica los métodos insertarGestor y verGestores para integrar la comunicación con el servidor. Puede utilizarse como ayuda los archivos siguientes: Database.java, Criptografia.java y el archivo de prueba Main21Test.java. Almacena la contraseña en formato SHA3 utilizando la librería [Bouncy Castle](#).

```
// Criptografia.java  
  
package com.banco.utils;  
  
import org.bouncycastle.jcajce.provider.digest.SHA3;  
import org.bouncycastle.util.encoders.Hex;  
  
public class Criptografia {  
  
    public static String SHA3(String str) {  
        SHA3.DigestSHA3 digestSHA3 = new SHA3.Digest512();  
        byte[] digest = digestSHA3.digest(str.getBytes());  
        return Hex.toHexString(digest);  
    }  
}
```

```
// Database.java  
  
package com.banco.utils;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;

import com.banco.entidades.Gestor;

public class Database {

    private Connection conexion;

    public Database() {

        try {

            // conecta con la base de datos
            conexion = DriverManager.getConnection("jdbc:mysql://localhost:3306/banco",
"banco", "banco");

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public boolean insertarGestor(Gestor gestor) {

        Statement instrucion = null;

        try {

            // obtiene un objeto de tipo Statement
            instrucion = conexion.createStatement();

            PreparedStatement ps = conexion.prepareStatement("INSERT INTO gestor(usuario,
password, correo) VALUES (?,?,?)");

            ps.setString(1, gestor.getUsuario());

            // se sustituye la segunda aparición del carácter ? con el valor gestor10
            ps.setString(2, Criptografia.SHA3(gestor.getPassword()));

            // se sustituye la tercera aparición del carácter ? con el valor gestor10@correo.com
            ps.setString(3, gestor.getCorreo());

            // ejecuta sentencia SQL
            ps.executeUpdate();

            // cierra la sentencia
            instrucion.close();

        } catch (SQLException e) {
            e.printStackTrace();
        }

        return true;
    }
}
```

```
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
    if (instruccion != null) {
        try {
            instruccion.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

return false;
}

public ArrayList<Gestor> getGestores() {

Statement instruccion = null;
ArrayList<Gestor> gestores = new ArrayList<Gestor>();

try {

// obtiene un objeto de tipo Statement
instruccion = conexion.createStatement();

// ejecuta sentencia SQL
ResultSet resultados = instruccion.executeQuery("SELECT * from gestor");

while (resultados.next()) {

int id = resultados.getInt("id");
String usuario = resultados.getString("usuario");
String password = resultados.getString("password");
String correo = resultados.getString("correo");

Gestor gestor = new Gestor(id, usuario, password, correo);
gestores.add(gestor);
}

// cierra la sentencia
instruccion.close();

return gestores;
} catch (SQLException e) {
e.printStackTrace();
} finally {
if (instruccion != null) {
try {
instruccion.close();
} catch (SQLException e) {
```

```
        e.printStackTrace();
    }
}

return null;
}
}
```

```
// Main21Test.java

package com.banco.main;

import java.util.ArrayList;

import com.banco.entidades.Gestor;
import com.banco.utils.Database;

public class Main21Test {

    public static void main(String[] args) {

        // inicializa la base de datos
        Database database = new Database();

        // obtiene los gestores
        ArrayList<Gestor> gestores = database.getGestores();
        System.out.println(gestores);

        // inserta un gestor
        Gestor gestor = new Gestor(1, "gestor2", "gestor2", "gestor1@mail.com");
        database.insertarGestor(gestor);

        // obtiene los gestores de nuevo
        gestores = database.getGestores();
        System.out.println(gestores);
    }
}
```

Tu carrera digital ~

# Módulo 5

# Spring e Hibernate

Introducción a Spring



# Introducción a Spring

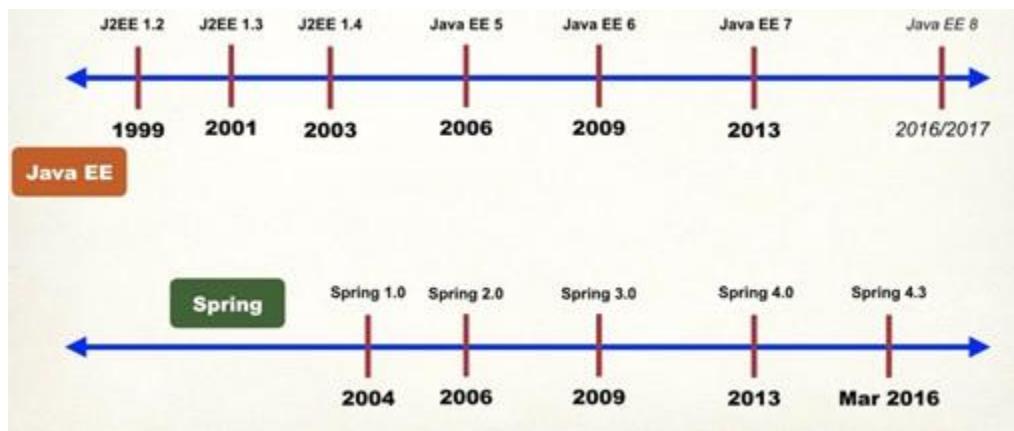
- ◆ [Introducción](#)
- ◆ [Preparativos de instalación](#)
- ◆ [Instalación de Tomcat](#)
- ◆ [Conectar Tomcat con Eclipse](#)
- ◆ [Instalar Spring sin Maven](#)
- ◆ [Extensión de Spring para Eclipse](#)

## Introducción

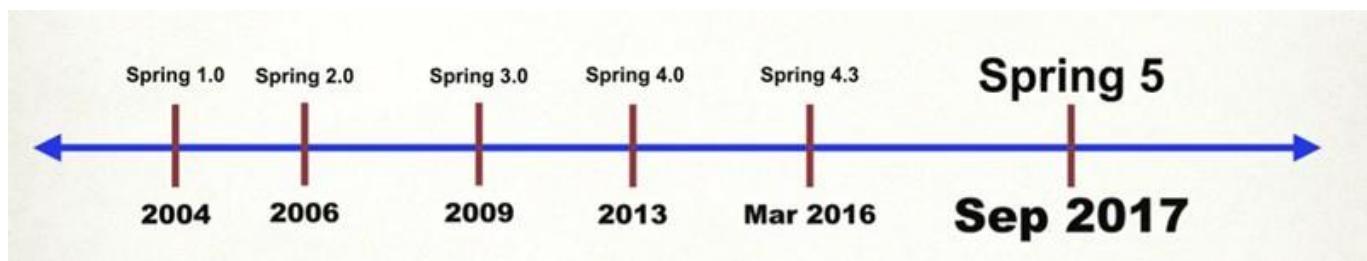
- ◆ Spring es un framework muy popular para el desarrollo de aplicaciones web empresariales basadas en el lenguaje Java.
- ◆ Inicialmente fue una alternativa simple y ligera a J2EE (antiguo Java EE/Jakarta EE).
- ◆ Con el tiempo ha ido adquiriendo una mayor relevancia en la comunidad y actualmente ha desplazado a Java EE como opción preferida para el desarrollo de aplicaciones web en Java.



- ◆ Spring surgió para solucionar los problemas de los EJB (Enterprise JavaBeans) en las primeras versiones de J2EE:
  - Difíciles de entender e integrar en un proyecto.
  - Rendimiento muy pobre en producción.
- ◆ Debido a estas dificultades, Rod Johnson (fundador de Spring) escribió en 2004 el libro J2EE Development without EJB y un año después Java Development with the Spring Framework.

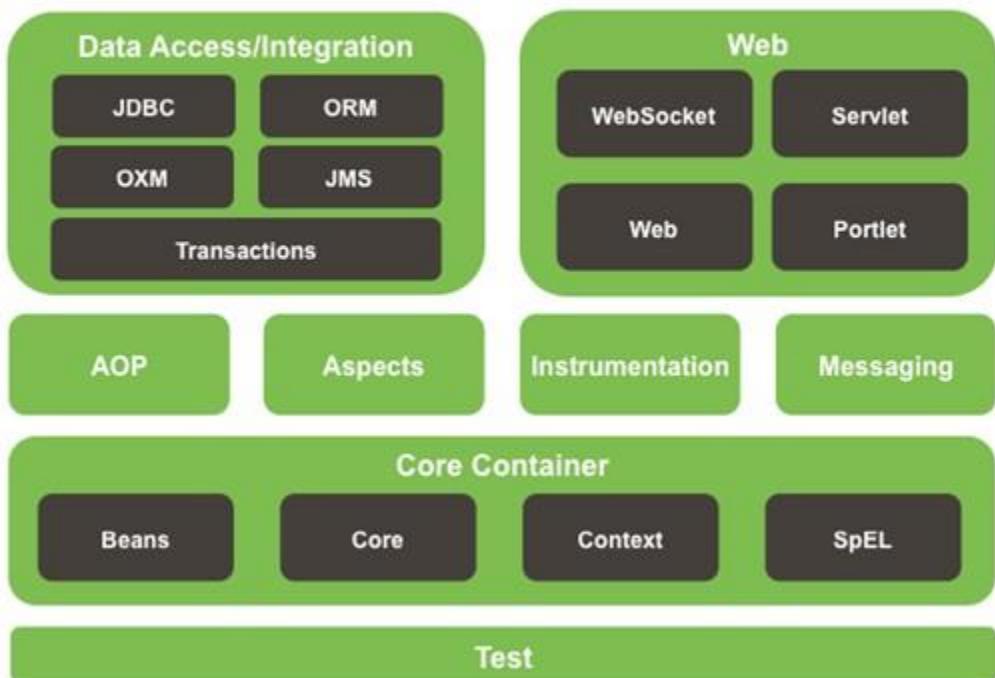


- Los EJB mejoraron mucho en su versión 3.1 (2013) a partir de Java EE 7, pero para entonces Spring ya había cobrado demasiado importancia en la industria.
- Actualmente no hay ninguna característica o funcionalidad importante de Spring que no ofrezca Java EE, pero los desarrolladores prefieren utilizar Spring.
- La mayor diferencia práctica entre estas dos tecnologías es que Spring puede funcionar con un servidor web convencional al estilo Tomcat, mientras que Java EE requiere de un servidor de aplicaciones más sofisticado.
- Actualmente la última versión de Spring es la 5.x, que requiere al menos Java 8 o superior.



- La principal nueva característica de Spring 5 es WebFlux, que permite incorporar a un proyecto Spring las funcionalidades ofrecidas por la programación reactiva.
- Spring tiene como principales características:
  - Una arquitectura orientada al desarrollo con Java POJOs (Plain Old Java Object), es decir, clases simples de Java que no requieren de un framework en particular para ser manejadas.
  - Inyección de dependencias para promover un máximo desacoplamiento entre objetos.
  - Programación declarativa con mediante programación orientada a aspectos (AOP).
  - Simplificación y minimización del código que debe escribir un desarrollador para que la aplicación sea fácil de entender y mantener.
- La arquitectura de Spring está organizada en unos 20 módulos agrupados en los componentes *Core Container*, *Data Access/Integration*, *Web*, *AOP*, *Instrumentation*, *Messaging* y *Test*.

## Spring Framework Runtime



- El *Core Container* está compuesto por:
  - *Core* y *beans*: permite crear los beans (*Bean factory*) y gestionar las dependencias de éstos.
  - *Context*: ofrece una forma de acceder a los beans de una manera similar al registro JNDI (Java Naming and Directory Interface) de Java EE.
  - *SpEL*: es un lenguaje para la consulta y manipulación de beans en tiempo real.



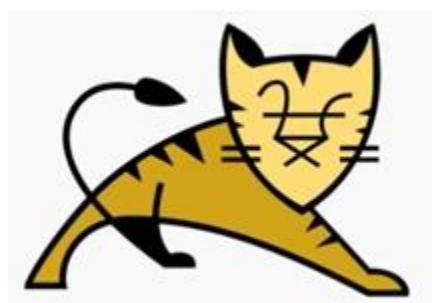
- El *Data Access/Integration* está compuesto por:
  - JDBC: permite trabajar con bases de datos de forma más simple que con las clases clásicas de JDBC.
  - ORM: facilita el mapeo de objetos mediante JPA (Java Persistence API) o Hibernate.
  - JMS: es un sistema de mensajería asíncrono.
  - Transactions: facilita el tratamiento de transacciones de métodos o llamadas a la base de datos.
  - OXM: facilita el tratamiento de archivos XML.



- ♦ Respecto al resto de componentes:
  - *AOP y Aspects*: añaden funcionalidades adicionales (logging, seguridad, transacciones, etc.) a los objetos Java de forma declarativa mediante archivos de configuración XML o anotaciones.
  - *Instrumentation*: permite monitorizar el nivel de rendimiento de una aplicación o diagnosticar errores.
  - *Messaging*: facilita la adición de otros sistemas de mensajería distintos a JMS.
- ♦ Adicionalmente hay otros servicios de Spring ([Projects](#)) proporcionados por la comunidad. Los más importantes son:
  - Spring Boot
  - Spring Cloud
  - Spring Integration
  - Spring Security
  - Spring for Android
  - Spring Web Services
  - Spring LDAP

## Preparativos de instalación

- ♦ Para trabajar con Spring se requiere:
  - JDK 8 o superior.
  - Un servidor que permita ejecutar aplicaciones Java empresariales (por ejemplo, Tomcat).
  - Un IDE (Eclipse es una buena opción).
  - Complementos o plugins que faciliten la integración de Spring con el IDE.
  - Las librerías de Spring, ya sea descargadas mediante Maven/Gradle o directamente desde su página oficial.



## Instalación de Tomcat

- Tomcat puede descargarse desde su página oficial.

### 9.0.38

Please see the [README](#) file for packaging information. It explains what every distribution contains.

#### Binary Distributions

- Core:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
  - [32-bit Windows zip \(pgp, sha512\)](#)
  - [64-bit Windows zip \(pgp, sha512\)](#)
  - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
  - [tar.gz \(pgp, sha512\)](#)
- Deployer:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
- Embedded:
  - [tar.gz \(pgp, sha512\)](#)
  - [zip \(pgp, sha512\)](#)

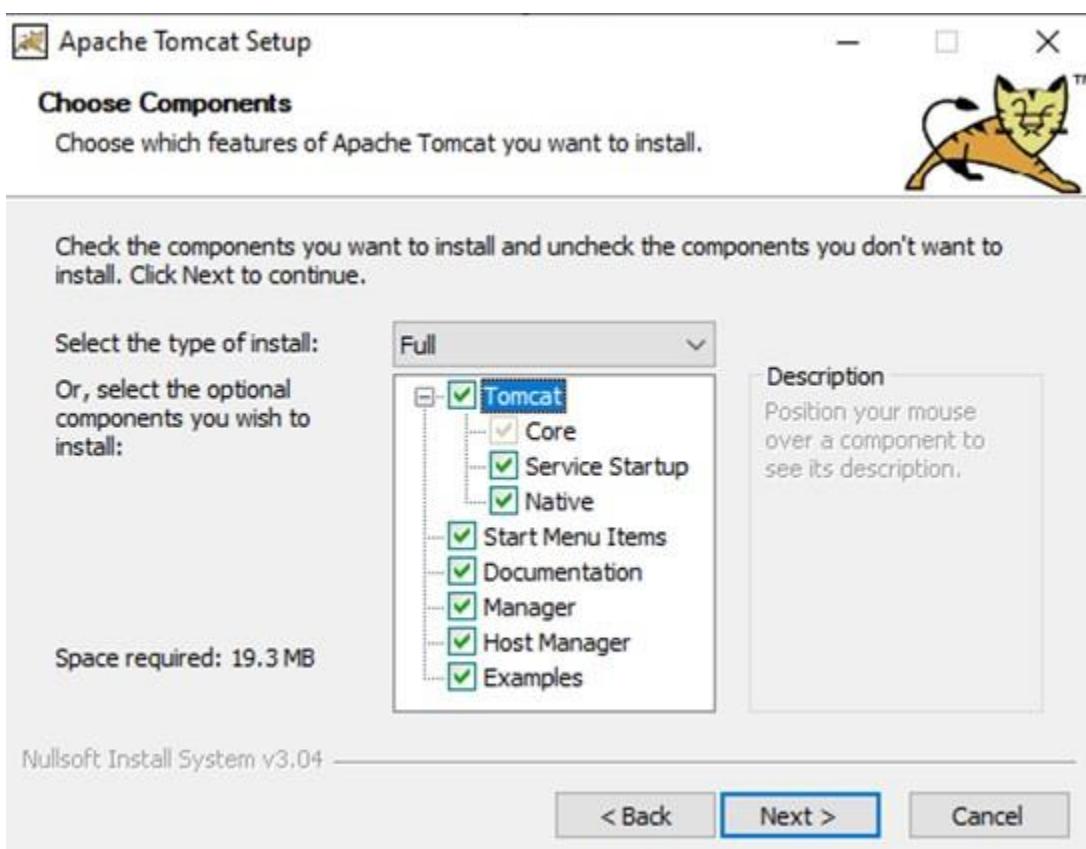
#### Source Code Distributions

- [tar.gz \(pgp, sha512\)](#)
- [zip \(pgp, sha512\)](#)

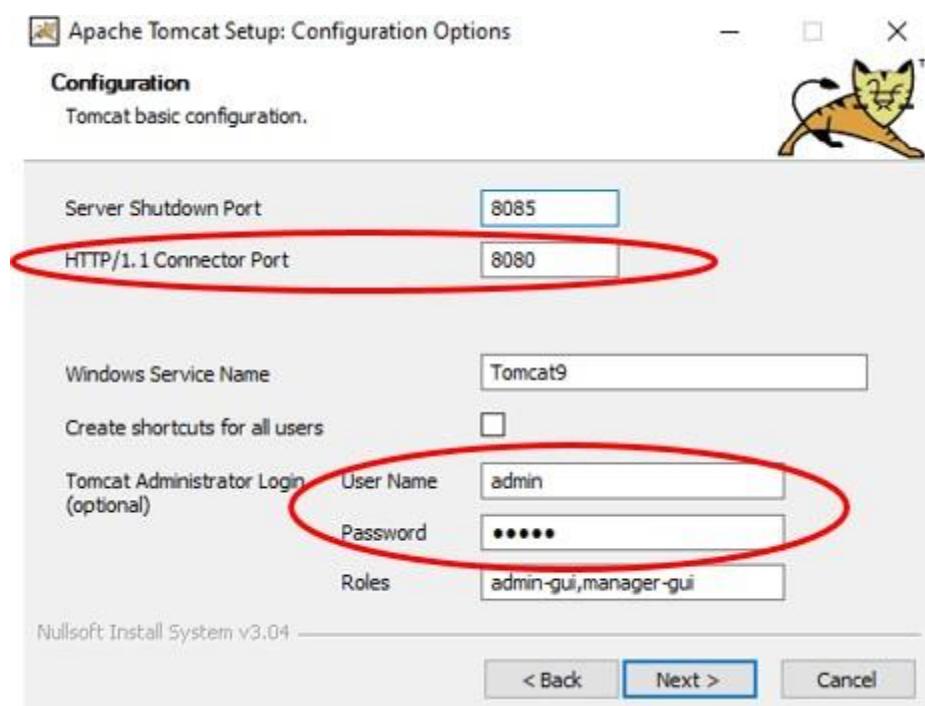
- El procedimiento de instalación es muy simple mediante un asistente.



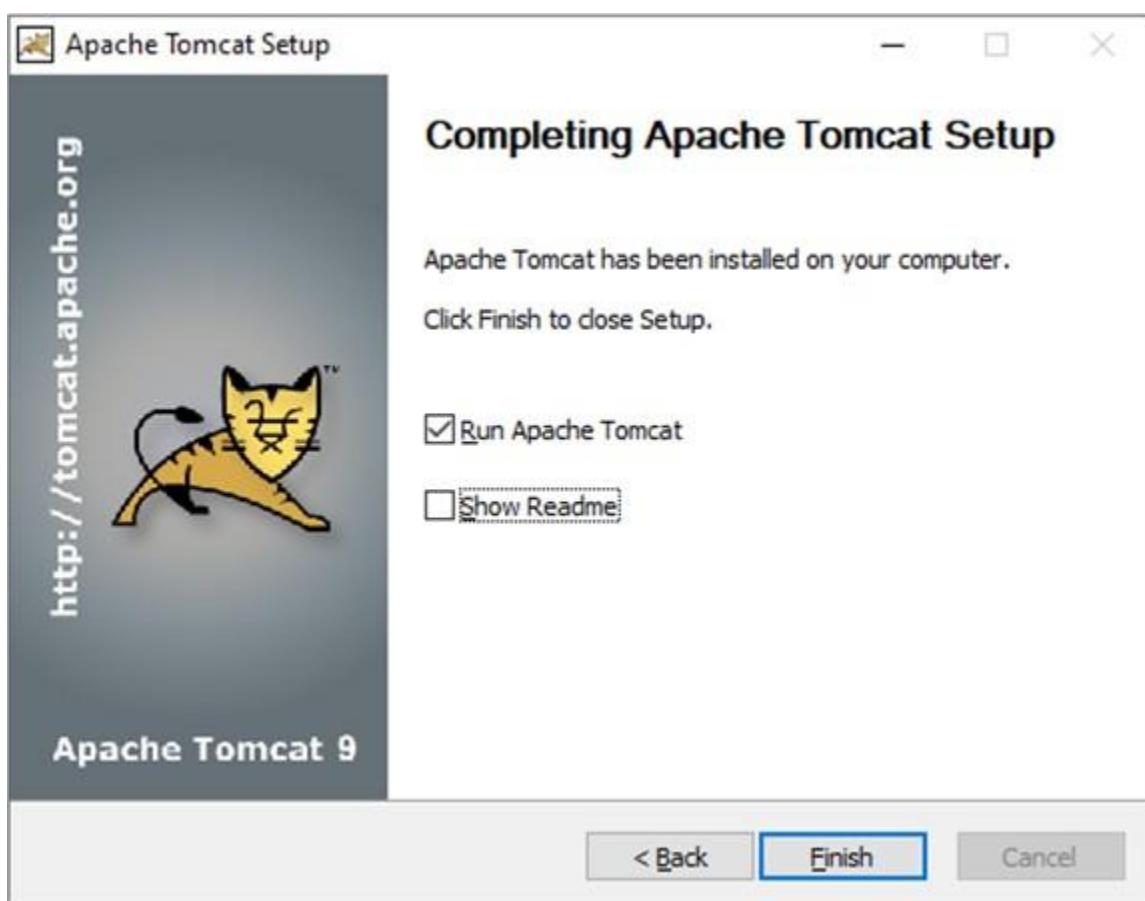
- Es recomendable realizar la instalación completa (Full) y asegurarse que Tomcat se instala como servicio de Windows.



- Tomcat se ejecuta por defecto en el puerto 8080 (puede mantenerse ese puerto o modificarse).
- Será necesario establecer unas credenciales para el acceso del administrador de Tomcat.



- En los pasos sucesivos no es necesario modificar nada.
- Tras la instalación, el asistente finaliza.



- Para verificar que la instalación se ha efectuado correctamente puede accederse desde un navegador a la dirección de [localhost](http://localhost)

## Apache Tomcat/9.0.38



If you're seeing this, you've successfully installed Tomcat. Congratulations!



### Recommended Reading:

- [Security Considerations How-To](#)
- [Manager Application How-To](#)
- [Clustering/Session Replication How-To](#)

[Server Status](#)
[Manager App](#)
[Host Manager](#)

### Developer Quick Start

[Tomcat Setup](#)
[First Web Application](#)
[Realms & AAA](#)
[JDBC DataSources](#)
[Examples](#)
[Servlet Specifications](#)
[Tomcat Versions](#)

### Managing Tomcat

For security, access to the [manager webapp](#) is restricted. Users are defined in:

`$CATALINA_HOME/conf/tomcat-users.xml`

In Tomcat 9.0 access to the manager application is split between different users.  
[Read more...](#)

[Release Notes](#)
[Changelog](#)
[Migration Guide](#)
[Security Notices](#)

### Documentation

[Tomcat 9.0 Documentation](#)
[Tomcat 9.0 Configuration](#)
[Tomcat Wiki](#)

Find additional important configuration information in:

`$CATALINA_HOME RUNNING.txt`

Developers may be interested in:

[Tomcat 9.0 Bug Database](#)

[Tomcat 9.0 JavaDocs](#)

[Tomcat 9.0 Git Repository at GitHub](#)

### Getting Help

[FAQ and Mailing Lists](#)

The following mailing lists are available:

[tomcat-announce](#)

Important announcements, releases, security vulnerability notifications. (Low volume).

[tomcat-users](#)

User support and discussion

[tomcat-user](#)

User support and discussion for [Apache Tomcat](#).

[tomcat-dev](#)

Development mailing list, including commit messages

- Un nuevo ícono de Tomcat aparecerá en el área de notificaciones de la barra de tareas.
- Desde este ícono es posible acceder a las opciones de configuración de Tomcat.



**Configure...**


---

[Start service](#)
[Stop service](#)
[Thread Dump](#)
[Exit](#)


---

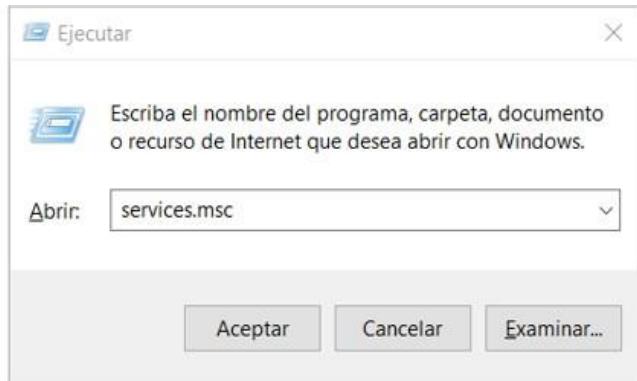
[About](#)

**Apache Tomcat 9.0 Tomcat9 Properties**

General		Log On	Logging	Java	Startup	Shutdown
Service Name:	Tomcat9					
Display name:	<b>Apache Tomcat 9.0 Tomcat9</b>					
Description:	Apache Tomcat 9.0.38 Server - <a href="https://tomcat.apache.org">https://tomcat.apache.org</a>					
Path to executable:	<code>"C:\Program Files\Apache Software Foundation\Tomcat 9.0\bin\Tomcat9"</code>					
Startup type:	<b>Automatic</b>					
Service Status:	Started					
		<b>Start</b>	<b>Stop</b>	<b>Pause</b>	<b>Restart</b>	

**Aceptar**   **Cancelar**   **Aplicar**

- Tomcat se encontrará en ejecución como servicio de Windows.

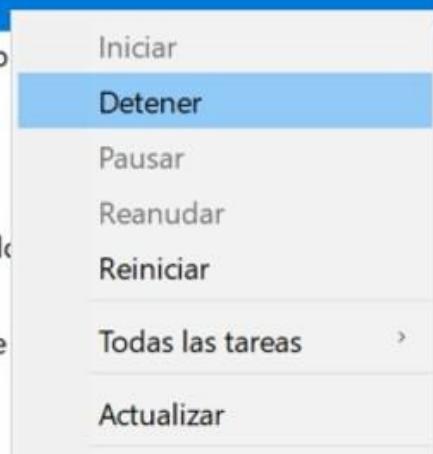


		Controla el a...	Manual	Sistema local
Almacenamiento de datos de usuarios_525fe190		Apache Tom...	En ejecu...	Automático Servicio local
Apache Tomcat 9.0 Tomcat9		Proporciona ...	En ejecu...	Manual (desen... Servicio local
Aplicación auxiliar de NetBIOS sobre TCP/IP		Proporciona ...	En ejecu...	Automático Sistema local
Aplicación auxiliar IP		Administra l...	Manual	Sistema local
Aplicación del sistema COM+		El servicio de...	Manual (desen... Sistema local	
Archivos sin conexión				

- Para una correcta integración con Tomcat es preferible parar el servicio (se arrancará de nuevo posteriormente en Eclipse).

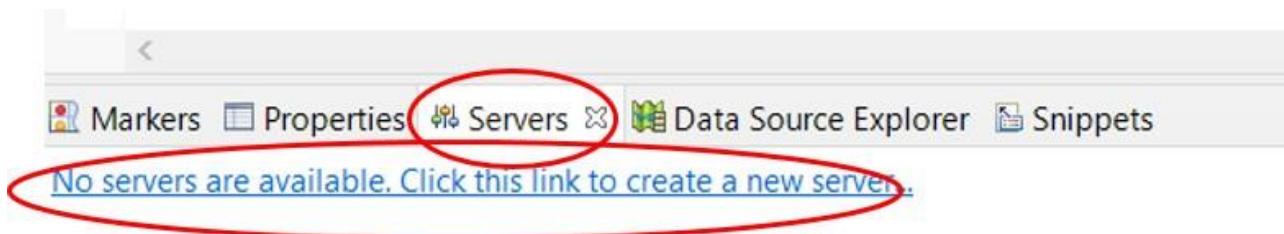
#### Almacenamiento de datos de usuarios\_7a043

- Apache Tomcat 9.0 Tomcat9
- Aplicación auxiliar de NetBIOS so...
- Aplicación auxiliar IP
- Aplicación del sistema COM+
- Archivos sin conexión
- Asignador de detección de topolo...
- Asignador de extremos de RPC
- Asistente para la conectividad de...
- ASP.NET State Service
- Audio de Windows



## Conectar Tomcat con Eclipse

- Las principales ventajas de conectar Tomcat con Eclipse son:
  - Arrancar o parar Tomcat desde Eclipse
  - Desplegar fácilmente aplicaciones en Tomcat.
- Para integrar Tomcat con Eclipse, en primer lugar se accede a la pestaña Servers, en la parte inferior de la ventana de Eclipse.



- Se elige a continuación el servidor Tomcat v9.0 Server de Apache.



- En la siguiente ventana se establece el directorio donde se ha instalado Tomcat 9. Por defecto la ruta es: *C:\Program Files\Apache Software Foundation\Tomcat 9.0*

## Tomcat Server

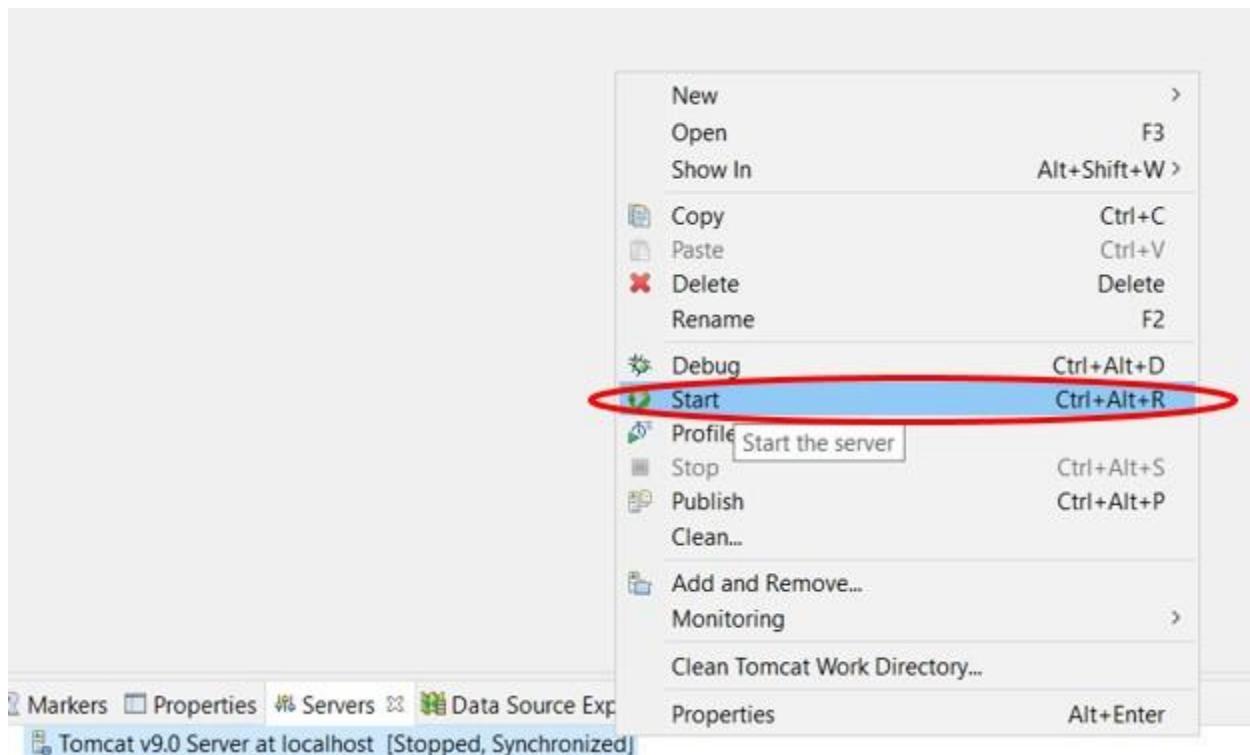
Specify the installation directory



Name:

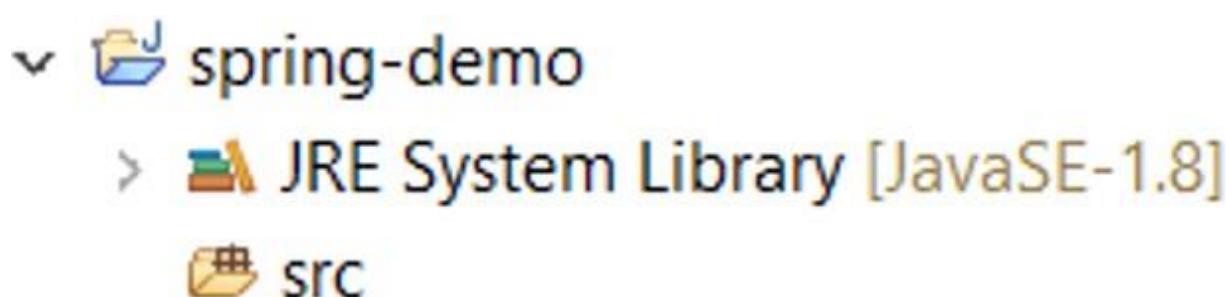
Tomcat installation directory:    Browse... Download and Install...

- En la siguiente ventana se pulsa sobre Finish. El nuevo servidor aparecerá en la pestaña Servers y puede iniciarse pulsando en Start desde el menú contextual.



## Instalar Spring sin Maven

- Los archivos de Spring deben descargarse desde la página oficial de Spring.
- Estos archivos se añadirán al proyecto donde quiere utilizarse Spring.
- Por tanto, en primer lugar es necesario crear un proyecto de Java normal.

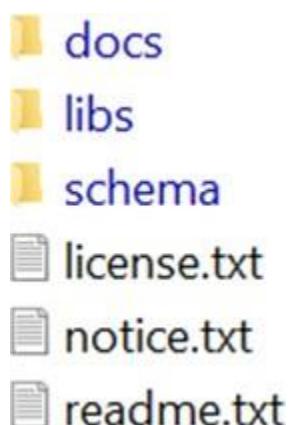


- Spring puede descargarse desde su página oficial.

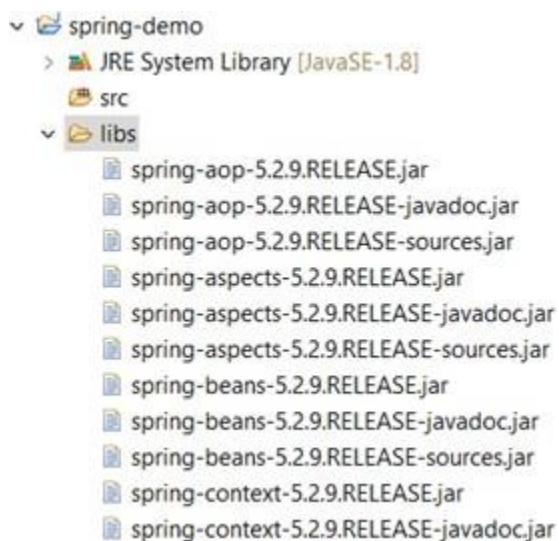
## Index of release/org/springframework/spring

Name	Last modified	Size
<a href="#">..</a>		
<a href="#">1.0/</a>	27-Apr-2017 02:22	-
<a href="#">1.0-m4/</a>	27-Apr-2017 12:56	-
<a href="#">1.0-rc1/</a>	27-Apr-2017 13:31	-
<a href="#">1.0.1/</a>	27-Apr-2017 18:06	-
<a href="#">1.1/</a>	30-Apr-2017 07:54	-
<a href="#">1.1-rc1/</a>	26-Jan-2017 01:14	-
<a href="#">1.1-rc2/</a>	18-Jan-2017 23:39	-
<a href="#">1.1.1/</a>	27-Apr-2017 08:49	-
<a href="#">1.1.2/</a>	30-Apr-2017 07:56	-
<a href="#">1.1.3/</a>	28-Apr-2017 07:11	-
<a href="#">1.1.4/</a>	28-Apr-2017 21:26	-
<a href="#">1.1.5/</a>	28-Apr-2017 10:31	-
<a href="#">1.2/</a>	27-Apr-2017 13:43	-
<a href="#">1.2-rc1/</a>	27-Apr-2017 13:32	-

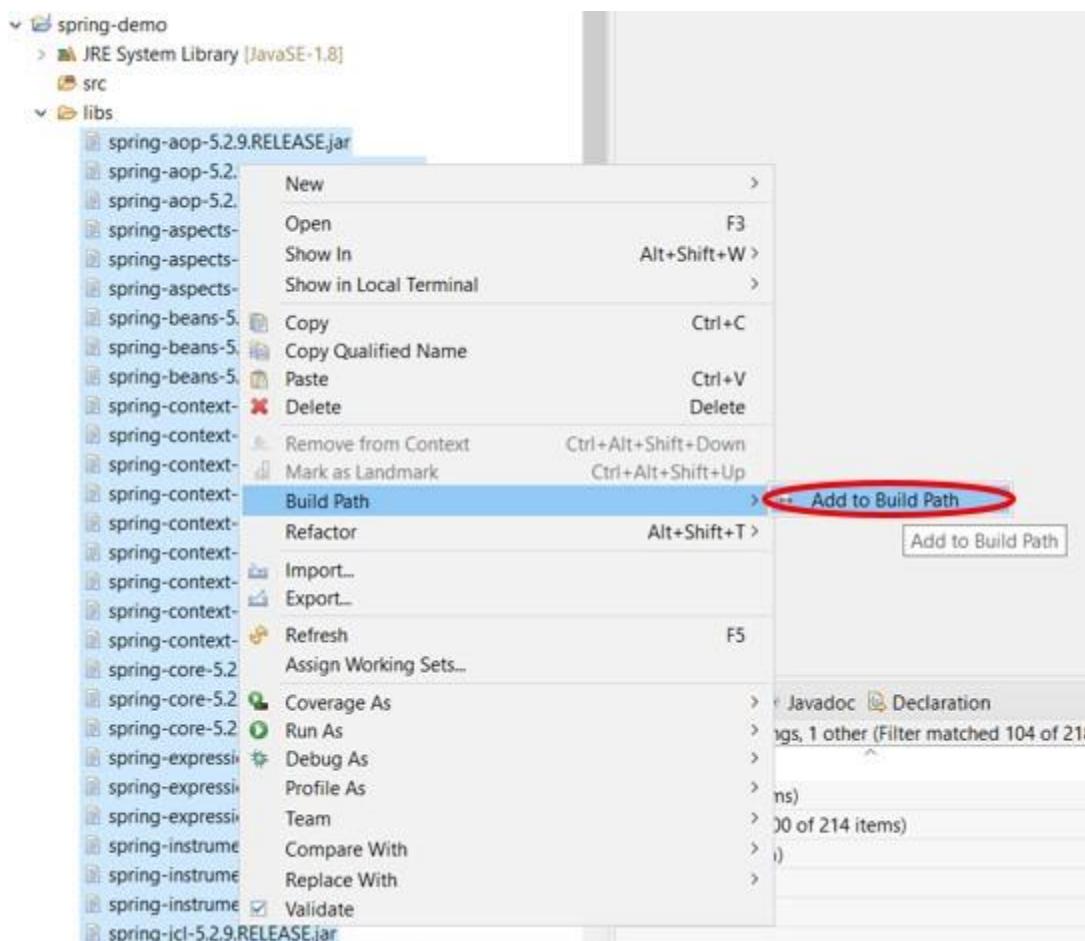
- La versión recomendada a instalar es la última disponible (actualmente 5.x.x RELEASE), seleccionando para descargar el archivo spring-5.x.x-RELEASE-dist.zip
- El directorio más importante del comprimido descargado es *libs*



- Este directorio debe copiarse al proyecto de Eclipse



- Por último, se seleccionan todos los archivos dentro de libs y se añade al *Build Path*.



## Extensión de Spring para Eclipse

- La mejor extensión de Spring para Eclipse se llama Spring Tool Suite (STS) y puede descargarse desde el Marketplace (*Help -> Eclipse Marketplace...*).

## Eclipse Marketplace

### Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.  
Press the "more info" link to learn more about a solution.



Search	Recent	Popular	Favorites	Installed	Giving IoT An Edge
Find: <input type="text" value="Spring Tool Suite"/>			<input type="button" value="X"/> All Markets		All Categories <input type="button" value="Go"/>

### Spring Tools 4 (aka Spring Tool Suite 4) 4.8.1.RELEASE

Spring Tools 4 is the next generation of Spring Boot tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support... [more info](#)

by [VMware](#), EPL  
[spring](#) [Spring IDE](#) [Cloud Spring Tool Suite STS](#)

2844
 Installs: 1,71M (24.978 last month)

- Será necesario seleccionar todos los componentes para instalar.

## Confirm Selected Features

Press Confirm to continue with the installation. Or go back to choose more solutions to install.

Spring Tools 4 (aka Spring Tool Suite 4) 4.8.1.RELEASE <https://download.springsource.com/release/TOOLS/sts4/update/>

- Spring Boot Language Server Feature (required)
- Spring Tool Suite 4 Main Feature (required)
- BOSH Language Server Feature
- Cloud Foundry Manifest Language Server Feature
- Concourse Pipeline Language Server Feature

- Y también aceptar las condiciones de uso.

 Eclipse Marketplace

## Review Licenses

Licenses must be reviewed and accepted before the software can be installed.



## Licenses:

- > Eclipse Public License - v 1.0
- > SPRING IDE PROJECT SOFTWARE USER AGREEMENT
- > The content of this package is provided under the EPL v1.0 license (s)

## License text:

## Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

## 1. DEFINITIONS

- I accept the terms of the license agreements  
 I do not accept the terms of the license agreements

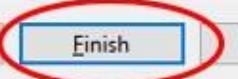


&lt; Back

Next &gt;

Finish

Cancel



Tu carrera digital ~

# Módulo 5

## Spring e Hibernate

Inversión de control



# Inversión de control

- Introducción
- Inversión de control en Spring
- Inversión de control en Spring con archivo XML
- Inversión de control en Spring con anotaciones

## Introducción

- La inversión de control es un principio de diseño de software cuya idea fundamental es la externalización de la instanciación y gestión de objetos.
- La mejor forma de entender este concepto de programación es mediante un ejemplo.



```
// EntrenadorFutbol.java

public class EntrenadorFutbol {

    public String getEntrenamiento() {
        return "Correr durante 30 minutos";
    }
}
```

```
// MainSpring.java

public class Main {

    public static void main(String[] args) {
        EntrenadorFutbol entrenador = new EntrenadorFutbol();
        System.out.println(entrenador.getEntrenamiento());
    }
}
```

- El ejemplo anterior funcionaría correctamente, pero una buena práctica sería generalizar el Entrenador para que sea sencillo adaptarlo a otros deportes.

- La generalización del Entrenador se construye mediante una interfaz.

```
// Entrenador.java

public interface Entrenador {
    public abstract String getEntrenamiento();
}
```

- Entrenadores de distintos deportes pueden implementar ahora la nueva interfaz.

```
// EntrenadorFutbol.java

public class EntrenadorFutbol implements Entrenador {

    @Override
    public String getEntrenamiento() {
        return "Correr durante 30 minutos";
    }
}
```

```
// EntrenadorBaloncesto.java

public class EntrenadorBaloncesto implements Entrenador {

    @Override
    public String getEntrenamiento() {
        return "Lanzar 30 tiros a canasta";
    }
}
```

- En Java no se puede crear una instancia a partir de una interfaz con *new* (a la derecha del signo igual), pero puede crearse una referencia a partir una interfaz (a la izquierda del signo igual). En estos casos la clase que se instance con *new* debe implementar obligatoriamente la interfaz.

```
// Main.java

public class Main {

    public static void main(String[] args) {
        Entrenador entrenadorFutbol = new EntrenadorFutbol();
        Entrenador entrenadorBaloncesto = new EntrenadorBaloncesto();
    }
}
```

```

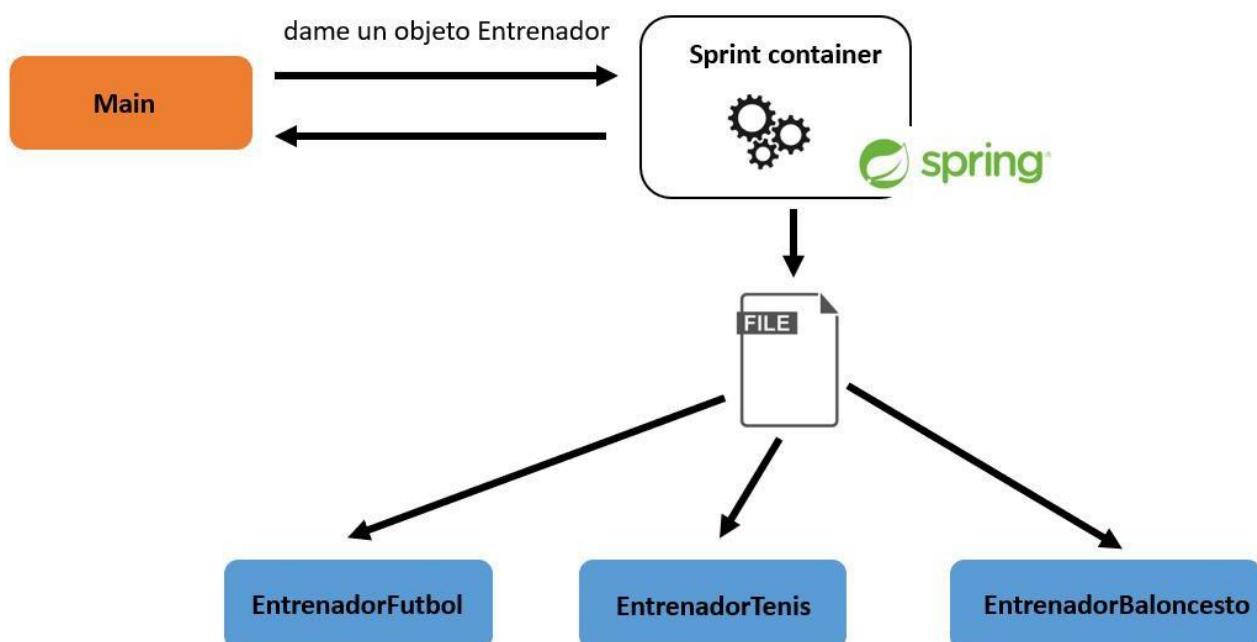
        System.out.println(entrenadorFutbol.getEntrenamiento());
        System.out.println(entrenadorBaloncesto.getEntrenamiento());
    }
}

```

- ◆ Sin embargo, cuando se manejan muchos entrenadores de distintos deportes es fácil cometer errores, además de resultar repetitivo tener que instanciar continuamente el objeto indicado y complicado de mantener si se añaden nuevos entrenadores de otros deportes o se elimina alguno ya existente.
- ◆ Debido a este problema surge la inversión de control y, muy relacionado también, la inyección de dependencias.
  - La inversión de control se encarga de crear y gestionar objetos.
  - La inyección de dependencias facilita el acceso al objeto donde se requiera y maneja todas sus dependencias.
- ◆ El núcleo de Spring (Core) se basa en los conceptos de inversión de control y en la inyección de dependencias. Spring permite inyectar los objetos donde se desee añadiéndole funcionalidades adicionales (seguridad, transaccionalidad, etc.) con muy poca intervención por parte del desarrollador.

## Inversión de control en Spring

- ◆ El *container* de Spring se encarga de instanciar el objeto (bean) correspondiente a partir de una configuración previamente establecida.



- ◆ Un bean es simplemente un objeto de Java que no es obtenido a través de una instancia normal, sino mediante a través de una factoría de objetos o contenedor

(como sucede en Spring).

- ◆ La configuración de cómo se instancia y manejan los objetos se establece a través de una de las siguientes opciones:
  - Un archivo XML (forma clásica).
  - Anotaciones (forma moderna).
  - ◊ Directamente en el código fuente (forma moderna).
- ◆ El proceso de gestión de objetos en Spring está constituido por las siguientes fases:
  - Configuración de los beans.
  - ◊ Creación del *Spring container*.
  - Obtención de los beans a partir del *Spring Container*.

## Inversión de control en Spring con archivo XML

- ◆ En primer lugar es necesario crear un archivo XML llamado applicationContext.xml en el directorio src del proyecto (este archivo también puede encontrarse en el directorio utilidades).
- ◆ El archivo XML applicationContext.xml define todos los beans de la aplicación a través de la etiqueta *bean* y dos atributos:
  - ◊ *id*: un identificador arbitrario establecido por el desarrollador.
  - *class*: la clase asociada al bean y el paquete donde se encuentra.

```
<!-- applicationContext.xml -->

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- a partir de aquí se definen los beans -->

    <!-- bean mientrenador asociado a la clase EntrenadorBaloncesto -->
    <bean id="mientrenador" class="com.ejemplo.EntrenadorBaloncesto">

        </bean>

    </beans>
```

- A continuación, desde el código en Java puede abrirse el contexto a partir del archivo de configuración, obtener el bean y, finalmente, cerrar el contexto.

```
// MainSpring.java

public class MainSpring {
    public static void main(String[] args) {
        // abre el contexto a partir del archivo de configuración
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");

        /*
         obtiene el bean, donde:
         - el primer parámetro es el identificador del bean
         - el segundo parámetro es la interfaz que implementa el bean a recibir, realizando
         una especie casting
        */
        Entrenador entrenador = context.getBean("mientrenador", Entrenador.class);
        System.out.println(entrenador.getEntrenamiento());

        // cierra el contexto
        context.close();
    }
}
```

- Es posible cambiar la clase asociada al identificador del bean desde el archivo XML sin necesidad de modificar el código Java y, por tanto, volverlo a compilar.
  - Tras este cambio, el *container* devolverá ahora un objeto de la nueva clase establecida.

```
<!-- ahora la clase es EntrenadorFutbol en lugar de EntrenadorBaloncesto -->
<bean id="mientrenador" class="com.ejemplo.EntrenadorFutbol">
</bean>
```

- A partir de la versión 5.1 de Spring se eliminaron los mensajes de log. Para habilitarlos de nuevo puede seguirse el manual "*Add Logging Messages in Spring 5.1*" en la carpeta "tutoriales".

## Inversión de control en Spring con anotaciones

- Una alternativa a los archivos de configuración en Spring es el uso de anotaciones.

- Las anotaciones son etiquetas o marcas que son colocadas a las clases de Java y proveen información adicional o metadatos sobre una clase.
- Las anotaciones son procesadas en tiempo de compilación o en tiempo de ejecución.
- ◆ En Spring pueden utilizarse las anotaciones para poder minimizar la configuración del archivo applicationContext.xml. Este archivo puede volverse extenso y difícil de interpretar en proyectos complejos.
- ◆ Spring escanerá todas las clases Java para buscar anotaciones especiales y automáticamente registrará los beans en el *container*. Este escaneo debe establecerse en el archivo de configuración applicationContext.xml

```
<!-- ... -->

<!-- Spring escaneará recursivamente todas las clases del paquete y subpaquetes
com.ejemplo -->
<context:component-scan base-package="com.ejemplo"/>

<!-- ... -->
```

- ◆ La anotación para registrar un bean es @Component y debe ubicarse antes de la definición de la clase.
  - El @Component recibe como parámetro el id del bean.

```
// EntrenadorFutbol.java

// mientrenador es el id del bean
@Component("mientrenador")
public class EntrenadorFutbol implements Entrenador {
    public String getEntrenamiento() {
        return "Correr durante 30 minutos";
    }
}
```

- ◆ Finalmente puede obtenerse el bean a partir del contexto.

```
// MainSpring.java

public class MainSpring {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");
```

```
Entrenador entrenador = context.getBean("mientrenador", Entrenador.class);
System.out.println(entrenador.getEntrenamiento());

context.close();
}
```

- Si no se establece el id en la anotación, entonces el id toma por defecto el nombre de la clase con la primera letra en minúscula, excepto si las dos primeras letras son mayúsculas (en ese caso se mantendría el id igual al nombre de la clase del bean. Por ejemplo, el bean RESTServicio mantendría RESTServicio como id).
  - Spring utiliza el método *decapitalize* para obtener el id de un bean a partir del nombre de la clase.

```
// EntrenadorFutbol.java

// entrenadorFutbol es el id del bean
@Component
public class EntrenadorFutbol implements Entrenador {
    public String getEntrenamiento() {
        return "Correr durante 30 minutos";
    }
}
```

- Y será necesario también modificar la solicitud del bean.

```
Entrenador entrenador = context.getBean("entrenadorFutbol", Entrenador.class);
```

Tu carrera digital ~

# Módulo 5

## Spring e Hibernate

Inyección de dependencias



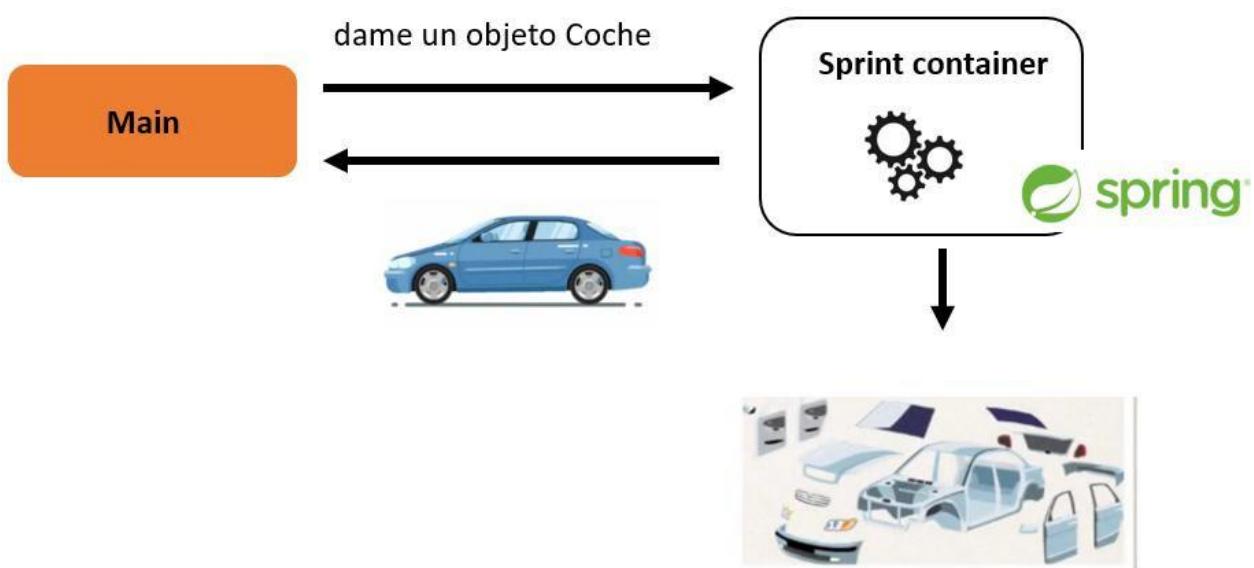
# Inyección de dependencias

---

- ◆ [Introducción](#)
- ◆ [Inyección de dependencias en Spring con archivo XML](#)
  - [Inyección en el constructor](#)
  - [Inyección en un método setter](#)
  - [Injectar valores literales](#)
  - [Injectar valores desde un archivo properties](#)
- ◆ [Inyección de dependencias en Spring con anotaciones y autowiring](#)
  - [Inyección en el constructor](#)
  - [Inyección en un método setter u otro método](#)
  - [Inyección en un atributo](#)
  - [Elección de la dependencia](#)
  - [Injectar valores desde un archivo properties](#)

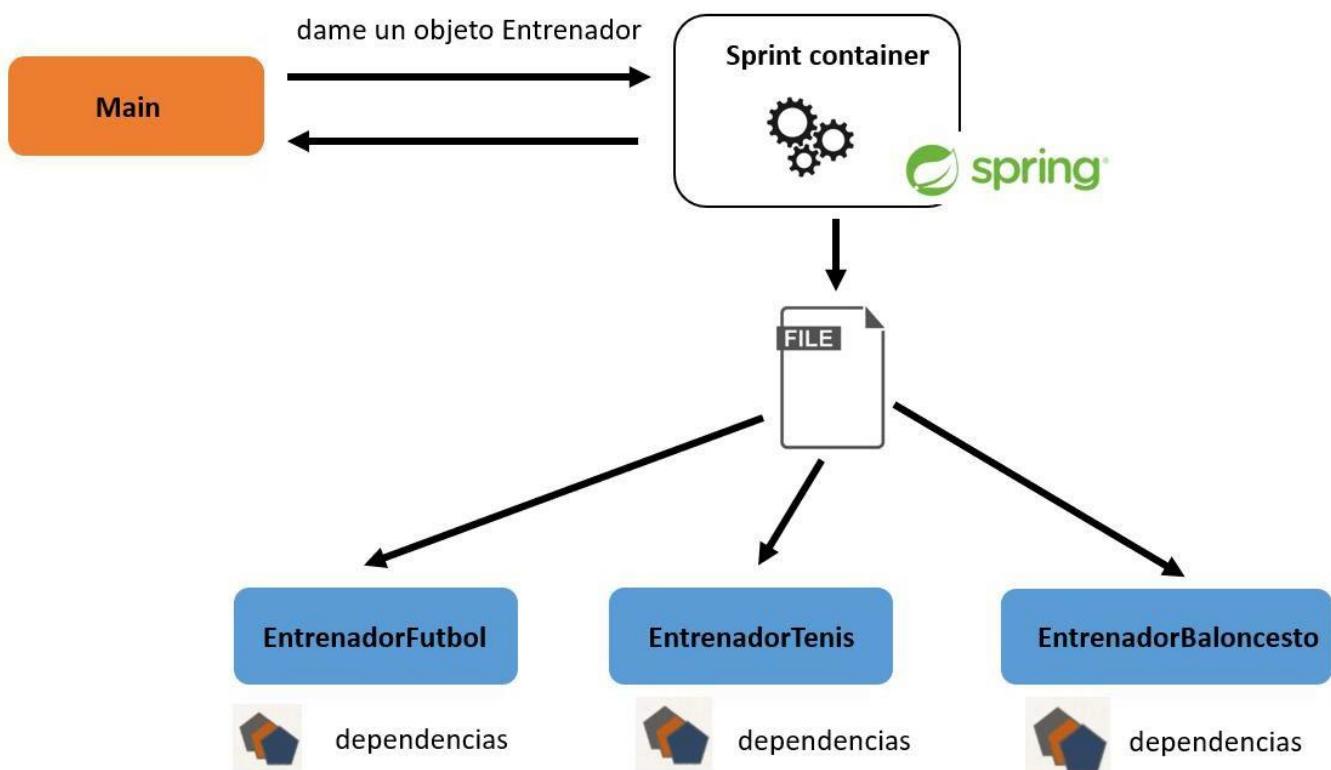
## Introducción

- ◆ La inyección de dependencias es un patrón de diseño en programación similar a la inversión de control, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos.
  - Los objetos cumplen determinados requisitos para poder ser injectados (de ahí el concepto de dependencia).
  - El *container* será el encargado de injectar el objeto deseado en el lugar correcto y con los requisitos necesarios.
- ◆ La inyección de dependencias es una forma de inversión de control, donde los objetos se pasan o inyectan generalmente a una clase a través de su constructor o algún método *setter*, en lugar de solicitarlos explícitamente al *container* como se ha visto anteriormente en el ejemplo de inversión de control.
- ◆ Un proyecto grande puede contener objetos que dependan de cientos de dependencias. Sin la inversión de control y la inyección de dependencias sería necesario inicializar esas cientos de dependencias de forma manual antes de obtener el objeto deseado.
- ◆ El *container* se encargará de construir y de resolver todas las dependencias (también llamadas *helpers*) del objeto a devolver.



## Inyección de dependencias en Spring con archivo XML

- En el ejemplo del entrenador visto anteriormente, cada tipo de entrenador puede estar constituido por múltiples dependencias.



- Existen varios tipos de inyección en Spring. Las dos más comunes son:
  - Inyección en el constructor
  - Inyección en un método *setter*

## Inyección en el constructor

- ◆ Los pasos a seguir son los siguientes:
  - Creación de la dependencia (interfaz y clase). Generalmente las dependencias proporcionan un servicio a un objeto, por lo que su nombre suele determinar en Servicio o Service
  - Creación de un constructor en la clase donde se espera recibir la inyección.
  - Configuración de la inyección de dependencia en el archivo de configuración de Spring.
- ◆ En primer lugar se crea la dependencia del objeto que se pretende inyectar (con una interfaz y una clase).

```
// ExperienciaServicio.java
public interface ExperienciaServicio {
    public int getExperiencia();
}
```

```
// PocaExperienciaServicio.java
public class PocaExperienciaServicio implements ExperienciaServicio {
    public int getExperiencia() {
        return 1;
}
```

- ◆ Seguidamente se añade en el constructor del entrenador el objeto de la clase que se quiere inyectar y se añade un nuevo método en la interfaz para manejar la respuesta del servicio.

```
// Entrenador.java
public interface Entrenador {
    public abstract String getEntrenamiento();

    // nuevo método para obtener la experiencia a partir del servicio que se inyectará
    public abstract int getExperiencia();
}
```

```
// EntrenadorFutbol.java
public class EntrenadorFutbol implements Entrenador {

    private ExperienciaServicio experienciaServicio;
```

```
// constructor con el servicio inyectado
public EntrenadorFutbol(ExperienciaServicio experienciaServicio) {
    System.out.println("Inyección en el constructor");
    this.experienciaServicio = experienciaServicio;
}

@Override
public String getEntrenamiento() {
    return "Correr durante 30 minutos";
}

// implementación del método del servicio inyectado
@Override
public int getExperiencia() {
    return experienciaServicio.getExperiencia();
}
}
```

- Por último, se añade el bean del servicio al archivo applicationContext.xml de Spring y se hace referencia al id de este bean en el que se pretende inyectar a través de la etiqueta etiqueta `constructor-arg`

```
<!-- -->

<!-- servicio añadido -->
<bean id="miExperienciaServicio" class="com.ejemplo.PocaExperienciaServicio">
</bean>

<bean id="mientrenador" class="com.ejemplo.EntrenadorFutbol">

    <!-- se establece qué bean se inyecta (miExperienciaServicio) y dónde (en el
        constructor) -->
    <constructor-arg ref="miExperienciaServicio"></constructor-arg>

</bean>

<!-- -->
```

- Ahora puede invocarse al servicio desde la clase MainSpring.

```
// MainSpring.java
public class MainSpring {
    public static void main(String[] args) {

        // la inyección se produce en el momento de abrir el contexto
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");
```

```

// obtener el bean
Entrenador entrenador = context.getBean("mientrenador", Entrenador.class);
System.out.println(entrenador.getExperiencia());

// cerrar el contexto
context.close();
}
}

```

## Inyección en un método *setter*

- Los pasos a seguir son los siguientes:
  - Creación de la dependencia (interfaz y clase).
  - Creación de un método *setter* donde se espera recibir la inyección.
  - Configuración de la inyección de dependencia en el archivo de configuración de Spring.
- La creación de la dependencia se realizó en el paso anterior. Ahora la inyección se realizará en un método *setter*

```

// EntrenadorBaloncesto.java
public class EntrenadorBaloncesto implements Entrenador {

    private ExperienciaServicio experienciaServicio;

    @Override
    public String obtenerEntrenamiento() {
        return "Lanzar 30 tiros a canasta";
    }

    @Override
    public int getExperiencia() {
        return experienciaServicio.getExperiencia();
    }

    // inyección del servicio en un método setter
    public void setExperienciaServicio(ExperienciaServicio experienciaServicio) {
        System.out.println("Inyección en un método setter");
        this.experienciaServicio = experienciaServicio;
    }
}

```

- Finalmente, se realiza las modificaciones necesarias en el archivo de configuración de Spring. En este caso se utiliza la etiqueta *property* dentro del bean donde se va a inyectar el servicio. Posee dos atributos

- *name*: hace referencia al método *setter* donde se inyectará el servicio. Spring le añade el prefijo *set* y convierte la primera letra en mayúscula. Por ejemplo, el nombre *experienciaServicio* corresponderá con el método *setExperienciaServicio*
- *ref*: hace referencia el id de bean que se va a inyectar.

```
<!-- -->

<!-- servicio añadido -->
<bean id="miExperienciaServicio" class="com.ejemplo.PocaExperienciaServicio">
</bean>

<bean id="mientrenador" class="com.ejemplo.EntrenadorBaloncesto">

    <!-- se establece qué bean se inyecta (miExperienciaServicio) y sobre qué método
    setter (setExperienciaServicio) -->
    <property name="experienciaServicio" ref="miExperienciaServicio" />

</bean>

<!-- -->
```

- ◆ Por último para testear el buen funcionamiento de la inyección puede obtenerse el bean desde la clase MainSpring.

```
// MainSpring.java
public class MainSpring {

    public static void main(String[] args) {

        // la inyección se produce en el momento de abrir el contexto
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");

        // obtener el bean
        Entrenador entrenador = context.getBean("mientrenador", Entrenador.class);
        System.out.println(entrenador.getExperiencia());

        // cerrar el contexto
        context.close();
    }
}
```

## Inyectar valores literales

- ◆ Para inyectar valores literales en atributos de un objeto es necesario en primer lugar crear los atributos en la clase que se pretende inyectar y sus respectivos métodos *setter*

```
// Entrenador.java
public interface Entrenador {

    public String getEmail();

    public void setEmail(String email);

    public String getEquipo();

    public void setEquipo(String equipo);
}
```

```
// EntrenadorFutbol.java
public class EntrenadorFutbol implements Entrenador {

    protected String email;

    protected String equipo;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        System.out.println("Inyección de un literal en un método setter: " + email);
        this.email = email;
    }

    public String getEquipo() {
        return equipo;
    }

    public void setEquipo(String equipo) {
        System.out.println("Inyección de un literal en un método setter: " + equipo);
        this.equipo = equipo;
    }
}
```

- En el archivo applicationContext.xml se establece mediante una etiqueta *property*, que posee dos atributos principales:
  - *name*: el nombre del atributo del objeto que se inyectará a través de su método *setter*
  - *value*: el valor que tendrá el atributo inyectado.

```
<bean id="mientrenador" class="com.ejemplo.EntrenadorFutbol">
```

```
<property name="email" value="correo de prueba@mail.com" />
<property name="equipo" value="Mean Machine" />

</bean>
```

```
// MainSpring.java
public class MainSpring {

    public static void main(String[] args) {

        // la inyección se produce en el momento de abrir el contexto
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");

        // obtener el bean
        Entrenador entrenador = context.getBean("mientrenador", Entrenador.class);
        System.out.println(entrenador.getEquipo());
        System.out.println(entrenador.getEmail());

        // cerrar el contexto
        context.close();
    }
}
```

## Inyectar valores desde un archivo properties

- La clase estándar de Java Properties (java.util.Properties) permite manipular archivos que almacenen de manera permanente en forma de pares clave/valor, con una estructura muy simple.

```
# datos.properties
correo de prueba@mail.com
equipo=Mean Machine
```

- En el archivo applicationContext.xml se carga el properties mediante la etiqueta *context:property-placeholder*, indicándole la ubicación con el atributo \*location
- Posteriormente puede accederse a los valores del archivo properties mediante la sintaxis \${clave}

```
<!-- carga del archivo datos.properties -->
<context:property-placeholder location="classpath:datos.properties"/>

<bean id="miExperienciaServicio" class="com.ejemplo.PocaExperienciaServicio">
```

```
</bean>

<bean id="mientrenador" class="com.ejemplo.EntrenadorFutbol">

    <!-- asigna el valor de la clave email del archivo properties -->
    <property name="email" value="${email}" />

    <!-- asigna el valor de la clave equipo del archivo properties -->
    <property name="equipo" value="${equipo}" />

</bean>
```

- La clase MainSpring.java no tendrá que ser modificada.
- En la práctica, es recomendable que todas las claves de un archivo properties posean un prefijo idéntico.

```
# datos.properties
datos.email=correoedeprueba@mail.com
datos.equipo=Mean Machine
```

## Inyección de dependencias en Spring con anotaciones y autowiring

- Spring posee un mecanismo de inyección automática llamado autowiring, que busca las dependencias requeridas por los beans y las inyecta automáticamente.
- Hay varios de inyección con autowiring:
  - Inyección en el constructor
  - Inyección en un método *setter* o en cualquier otro tipo de método
  - Inyección en un atributo

### Inyección en el constructor

- Los pasos a seguir son los siguientes:
  - Definir la interface y la clase de la dependencia.
  - Crear un constructor en la clase donde se realizará la inyección.
  - Configurar la inyección de depencia con la anotación @Autowired
- En primer lugar se define la interfaz y la clase de la dependencia.
  - En la clase debe colocarse la anotación @Component indicar que se trata de un bean y, de esta forma puede ser injectado en otro bean.

```
// ExperienciaServicio.java
public interface ExperienciaServicio {
    public int getExperiencia();
}
```

```
// PocaExperienciaServicio.java
// la clase se convierte en un bean
@Component
public class PocaExperienciaServicio implements ExperienciaServicio {
    public int getExperiencia() {
        return 1;
    }
}
```

- Ahora la dependencia puede inyectarse en el constructor mediante la anotación @Autowired

```
// Entrenador.java
public interface Entrenador {
    public int getExperienciaEntrenador();
}
```

```
// EntrenadorFutbol.java
// la clase se convierte en un bean
@Component
public class EntrenadorFutbol implements Entrenador {

    private ExperienciaServicio experienciaServicio;

    // constructor con el servicio inyectado
    @Autowired
    public EntrenadorFutbol(ExperienciaServicio experienciaServicio) {
        System.out.println("Inyección en el constructor");
        this.experienciaServicio = experienciaServicio;
    }

    public int getExperiencia() {
        return this.experienciaServicio.getExperiencia();
    }
}
```

- ◆ El ejemplo anterior funcionaría incluso sin utilizar la anotación @Autowired.
  - A partir de la versión 4.3 de Spring no es necesaria la anotación @Autowired si solamente hay un constructor en la clase.
  - Si existen varios constructores, entonces al menos uno debe incluir la anotación @Autowired.
  - El constructor con la anotación @Autowired no puede recibir parámetros que no sean beans.
  - Aunque no sea necesario el uso de @Autowired con un solo constructor, en la práctica es recomendable colocarlo para facilitar la lectura del código y ser más explícito.

## Inyección en un método setter u otro método

- ◆ El procedimiento es similar, pero utilizando la anotación @Autowired en un método setter

```
// EntrenadorFutbol.java

// la clase se convierte en un bean
@Component
public class EntrenadorFutbol implements Entrenador {

    private ExperienciaServicio experienciaServicio;

    // inyección del servicio en un método setter
    @Autowired
    public void setExperienciaServicio(ExperienciaServicio experienciaServicio) {
        System.out.println("Inyección en un método setter");
        this.experienciaServicio = experienciaServicio;
    }

    @Override
    public int getExperienciaEntrenador() {
        return this.experienciaServicio.getExperiencia();
    }
}
```

- ◆ La anotación @Autowired puede colocarse en cualquier método.

```
@Component
public class EntrenadorFutbol implements Entrenador {

    private ExperienciaServicio experienciaServicio;

    // este método se invocará automáticamente al crearse una instancia de
    // EntrenadorFutbol
    @Autowired
    public void ejecutar1(ExperienciaServicio experienciaServicio) {
```

```
System.out.println("ejecutar1");
this.experienciaServicio = experienciaServicio;
}

// este método también se invocará automáticamente al crearse una instancia aunque
no inyecte ningún bean
@Override
public int ejecutar2() {
    System.out.println("ejecutar2");
}
```

## Inyección en un atributo

- También se pueden injectar las dependencias directamente en un atributo, incluso en aquellos que son privados (Spring utiliza la tecnología de Java Reflection).
- En este caso la anotación @Autowired se coloca encima del atributo.

```
@Component
public class EntrenadorFutbol implements Entrenador {

    // la dependencia se inyecta directamente en el atributo
    @Autowired
    private ExperienciaServicio experienciaServicio;

    @Override
    public int getExperienciaEntrenador() {
        return this.experienciaServicio.getExperiencia();
    }
}
```

## Elección de la dependencia

- Puede suceder que existan diferentes implementaciones (todas con la anotación @Component) de una dependencia a inyectar. En ese caso, ¿cuál inyecta Spring?
  - Por ejemplo, en el caso anterior la interfaz Entrenador era implementada por EntrenadorFutbol, EntrenadorBaloncesto, EntrenadorTenis, etc. Si todos son beans y trata de inyectarse Entrenador, ¿qué bean se inyecta?

```
@Component
public class Prueba {

    // ¿Qué entrenador inyectaría? ¿EntrenadorFutbol? ¿EntrenadorBaloncesto?
    @Autowired
    private Entrenador entrenador;
```

```
public int getExperienciaEntrenador() {  
    return entrenador.getExperienciaEntrenador();  
}  
}
```

- Si existe ambigüedad en la inyección, Spring generará una excepción de tipo *NoUniqueBeanDefinitionException*
- Es necesario ser más explícito y utilizar la anotación `@Qualifier` para definir el identificador del bean concreto que debe inyectarse.

```
@Component  
public class Prueba {  
  
    @Autowired  
    // se inyectará el bean EntrenadorFutbol, cuyo id por defecto es entrenadorFutbol  
    @Qualifier("entrenadorFutbol")  
    private Entrenador entrenador;  
  
    public int getExperienciaEntrenador() {  
        return entrenador.getExperienciaEntrenador();  
    }  
}
```

- La anotación `@Qualifier` puede utilizarse en el constructor y cualquier método o atributo del bean.
- En los constructores debe colocarse obligatoriamente dentro de los paréntesis de los parámetros. Para el resto de métodos también es recomendable realizarlo de esta manera.

```
public Prueba(@Qualifier("entrenadorFutbol") Entrenador entrenador) {  
    this.entrenador = entrenador;  
}
```

## Inyectar valores desde un archivo properties

- Puede utilizarse la anotación `@Value` para inyectar valores desde un archivo properties directamente a un atributo de un bean.
- En primer lugar será necesario establecer la ruta donde se encuentra el archivo properties mediante la etiqueta `context:property-placeholder`

```
<context:property-placeholder location="classpath:data.properties"/>
```

- Seguidamente puede injectarse el valor mediante la anotación @Value y la sintaxis \${}

```
@Value("${email}")  
private String email;
```

Tu carrera digital ~

# Módulo 5

# Spring e Hibernate

Introducción a Hibernate



# Introducción a Hibernate

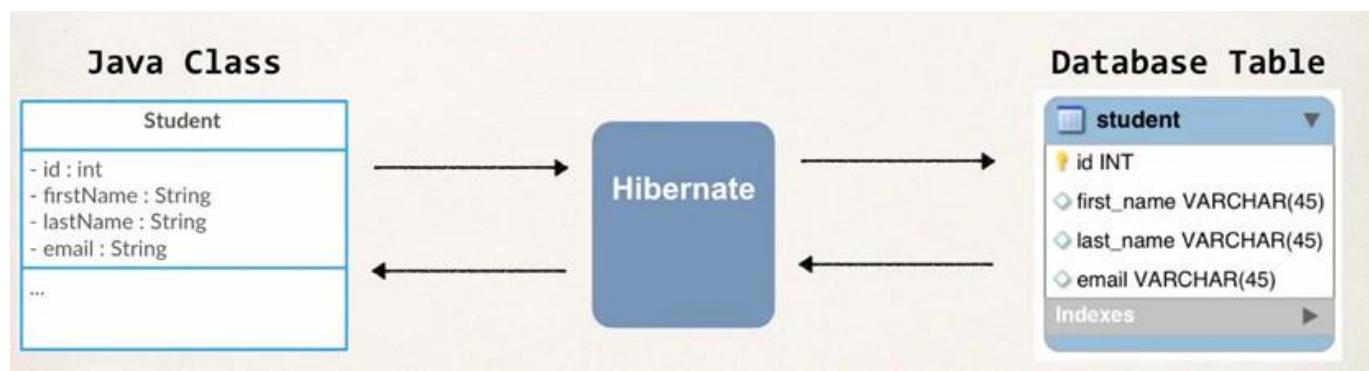
- ◆ [Introducción](#)
- ◆ [Integración de Hibernate en un proyecto](#)

## Introducción

- ◆ Hibernate es un framework que soporta la persistencia de datos mediante el almacenamiento de objetos de Java a una base de datos.
- ◆ Los beneficios principales de Hibernate son:
  - No requiere de la construcción de sentencias SQL para interactuar con la base de datos.
  - Es más fácil de utilizar que el API de JDBC.
  - Provee un mapeo Objeto-Relacional (ORM).
  - Elimina errores en tiempo de ejecución.
  - Es agnóstico de sistema de base de datos.



- ◆ El mapeo se produce entre un objeto de una clase de Java y un registro de una tabla en una base de datos.



- ◆ Los nombres de los atributos en la clase se encuentran en formato camel case, mientras que en la base de datos la equivalencia correspondería con el formato *underscore*.
- ◆ Al igual que con Spring, en Hibernate se puede establecer la configuración a través de archivos o de anotaciones.
- ◆ Hibernate utiliza internamente JDBC para interactuar con la base de datos.

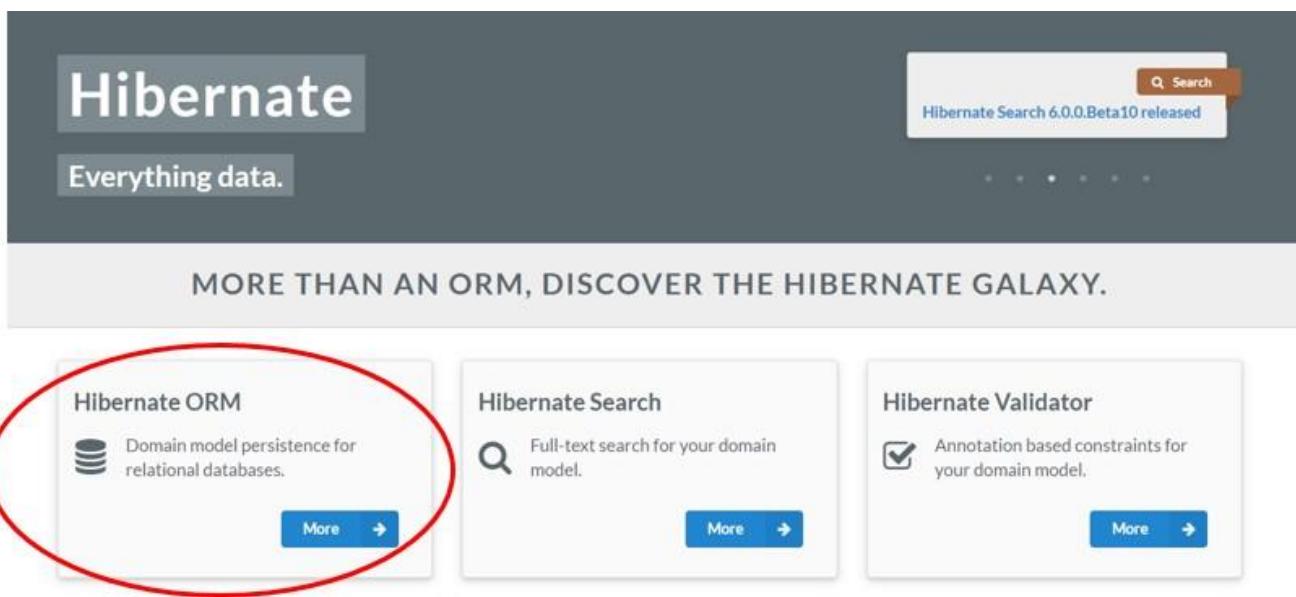


## Integración de Hibernate en un proyecto

- Hibernate puede integrarse en un proyecto de Java simple con Eclipse.
- Antes de crear un proyecto con Java es preferible cambiar la perspectiva de Eclipse a Java.
- Seguidamente se crea un nuevo proyecto Java y se añade la carpeta *lib*, donde se irán copiando las librerías necesarias y añadiendo al *Build Path*.



- En la [página web oficial de Hibernate](#) están disponibles las librerías que se necesitan importar al proyecto.



- Es recomendable instalar la última versión estable.

**Hibernate ORM**

Your relational data. Objectively.

Getting started →

Latest stable (5.4) → **(Red circle)**

Development (6.0) →

- About
- Releases
- Documentation
- Books
- Migration guides
- Roadmap
- Tooling
- Envers
- Contribute
- Paid support
- FAQ

**Compatibility**

Java	8 or 11
JPA	2.2

Not compatible with your requirements? Have a look at the other series.  
See also the [Compatibility policy](#).

**Documentation**

Documentation for this specific series can be accessed through the links below:

[HTML](#) [API \(JavaDoc\)](#)

You can find more documentation for all series on the [documentation page](#).

**How to get it**

Maven, Gradle...

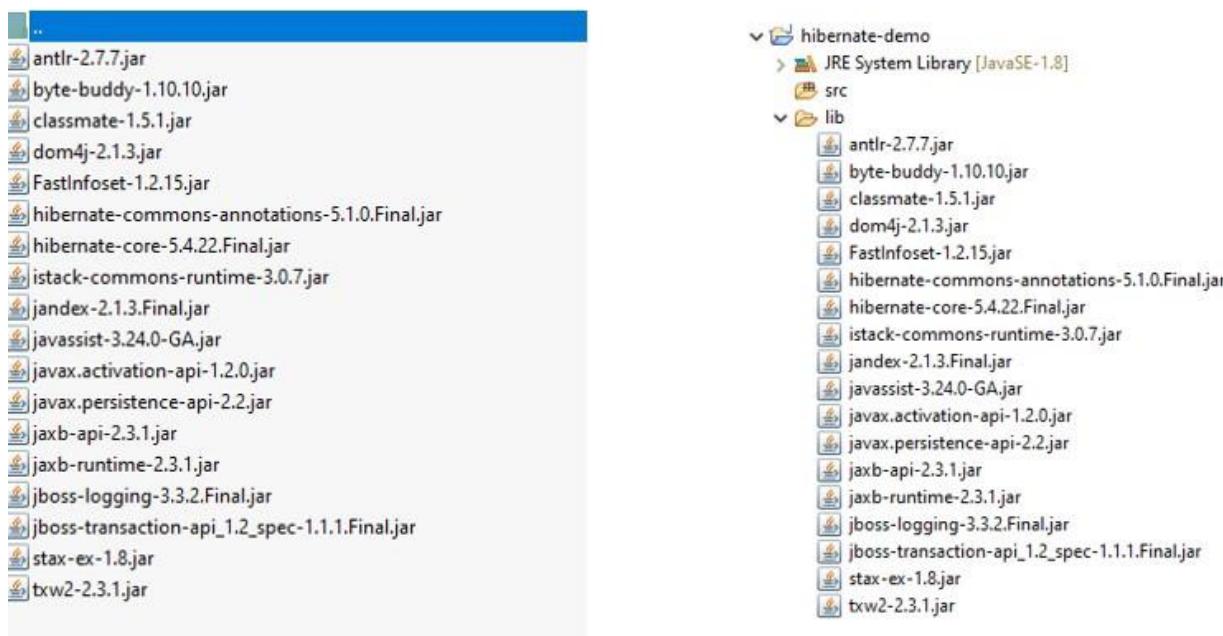
Maven artifacts of Hibernate ORM are published to [Maven Central](#) and to [Bintray](#).  
You can find the Maven coordinates of all artifacts through the link below:

[Maven artifacts](#)

**Quick links**

- Compatibility
- Documentation
- How to get it
- What's new
- Migrate
- Releases in this series

- Las librerías a importar en el proyecto se encuentran en *lib/required*, dentro del archivo comprimido descargado de la página web de Hibernate.



- Hibernate hace uso de las librerías de JDBC específicas para la base de datos utilizar (por ejemplo, MySQL).
- En la [página web oficial de descargas de MySQL](#) pueden descargarse estas librerías.

## ④ MySQL Community Downloads

- MySQL Yum Repository
- MySQL APT Repository
- MySQL SUSE Repository
  
- MySQL Community Server
- MySQL Cluster
- MySQL Router
- MySQL Shell
- MySQL Workbench
  
- MySQL Installer for Windows
- MySQL for Visual Studio
  
- C API (libmysqlclient)
- Connector/C++
- **Connector/J**
- Connector/.NET
- Connector/Node.js
- Connector/ODBC
- Connector/Python
- MySQL Native Driver for PHP
  
- MySQL Benchmark Tool
- Time zone description tables
- Download Archives

- ◆ Las librerías están disponibles para distintos sistemas operativos, pero es recomendable la opción *Platform Independent*. Seguidamente se pulsa en descargar el comprimido ZIP.

The screenshot shows the MySQL Connector/J 8.0.21 download page. At the top, there are tabs for "General Availability (GA) Releases" (selected), "Archives", and a help icon. Below the tabs, it says "Select Operating System:" and shows a dropdown menu with "Platform Independent" selected. This dropdown is circled in red. To the right, there's a link "Looking for previous GA versions?". The main content area lists two download options:

Platform	Version	File Size	Action
Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.21.tar.gz)	8.0.21	3.8M	<b>Download</b>
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.21.zip)	8.0.21	4.5M	<b>Download</b>

At the bottom, there's a note: "We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download." The "Download" button for the ZIP archive is also circled in red.

- ◆ No será necesario registrarse para comenzar la descarga.

## ④ MySQL Community Downloads

Login Now or Sign Up for a free account.

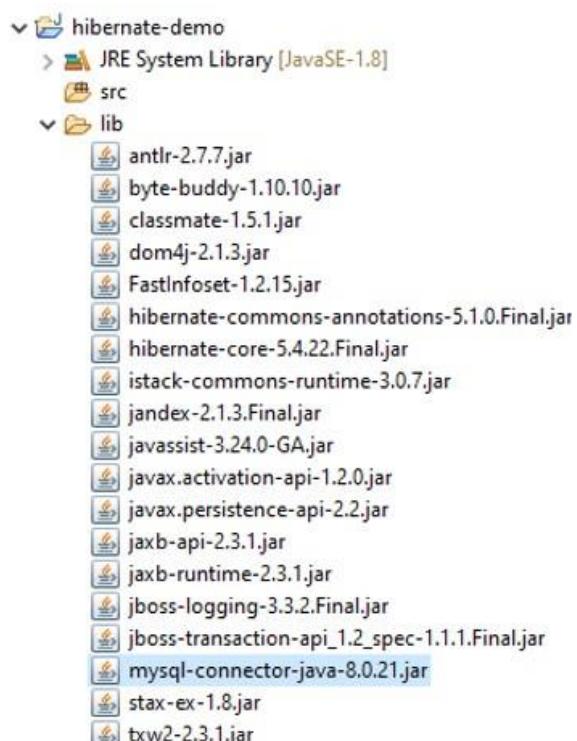
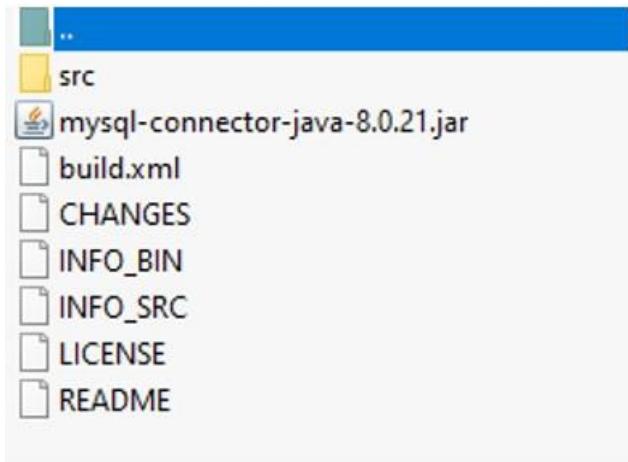
An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

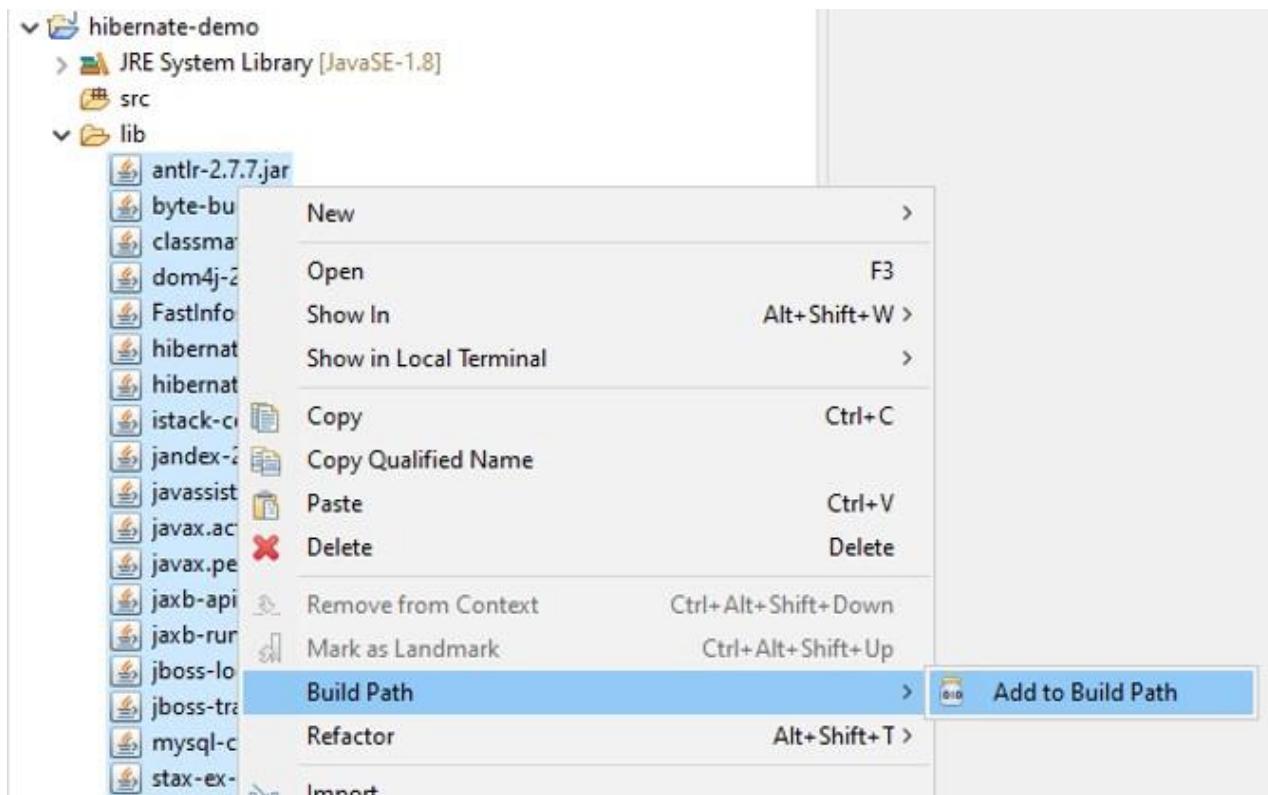
MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

- El único archivo que será necesario copiar será el JAR de MySQL.



- Finalmente, todos estos archivos formarán parte del *Build Path*.



- ◆ A continuación es necesario instalar MySQL y crear un nuevo usuario y base de datos asociada.
- ◆ Los datos necesarios para conectarse a una base de datos son:
  - Host
  - Puerto
  - Usuario
  - Password
  - Nombre de la base de datos
- ◆ Para testear que la base de datos se ha creado correctamente puede iniciarse una conexión mediante JDBC.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        String jdbcUrl = "jdbc:mysql://localhost:3306/hibernate-demo";  
        String usuario = "hibernate-demo";  
        String password = "hibernate-demo";  
  
        try {  
  
            System.out.println("Conectando a la base de datos: " + jdbcUrl);  
  
            Connection con = DriverManager.getConnection(jdbcUrl, usuario, password);  
  
            System.out.println("Conexión exitosa");  
  
        }  
        catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# Tu carrera digital ~



INVESTIGACIONES  
DEPARTAMENTO  
DE INVESTIGACIONES  
DE LA ESTADÍSTICA  
Y DEL ESTUDIO  
DE LA Población

red.es



"El FSE invierte en tu futuro"  
Fondo Social Europeo

Adecco