



red.es

en el Centro de Competencias
para la Transformación Digital



Adecco

Tu carrera digital ~

PROGRAMACIÓN WEB FULL STACK



Índice

| | |
|--|-----|
| Módulo 1 Fundamentos del desarrollo web | |
| Introducción | 3 |
| Breve introducción a la línea de comandos | 26 |
| Introducción a la pila de protocolos TCP/IP | 33 |
| Git | 50 |
| Módulo 2 Diseño de páginas interactivas frontend (HTML) | |
| HTML5 (I) | 61 |
| Nuevas etiquetas | 72 |
| Diseño de páginas interactivas frontend (CSS) | |
| CSS básico (I) | 83 |
| CSS básico (II) | 99 |
| CSS Flexbox | 118 |
| Diseño de páginas interactivas frontend (JavaScript) | |
| Fundamentos | 134 |
| Variables | 148 |
| Control de flujo | 166 |
| Funciones | 173 |
| Objetos | 189 |
| String | 198 |
| Array | 205 |
| Diseño de páginas interactivas frontend (TypeScript) | |
| Fundamentos | 213 |
| Variables | 225 |
| Funciones | 243 |
| Diseño de páginas interactivas frontend (Angular) | |
| Fundamentos (I) | 249 |
| Fundamentos (II) | 260 |
| Data binding (fundamentos) | 282 |
| Módulo 3 Java básico | |
| Introducción a Java | 294 |
| Fundamentos | 317 |
| Control del flujo | 341 |
| Orientación a objetos | 366 |
| Strings | 406 |
| Arrays | 422 |
| JSON | 430 |
| Módulo 4 Bases de datos (SQL) | |
| Introducción las bases de datos, MariaDB y SQL | 440 |
| JDBC | 480 |
| Módulo 5 Spring e Hibernate | |
| Introducción a Spring | 500 |
| Inversión de control | 516 |
| Inyección de dependencias | 524 |
| Ámbito y ciclo de vida de un bean | 539 |
| Configuración con anotaciones | 544 |
| Introducción a Spring MVC | 552 |
| Spring MVC – Fundamentos | 570 |
| Introducción a Hibernate | 586 |
| Hibernate básico | 594 |
| Spring MVC con Hibernate | 609 |
| Spring con Maven | 625 |

Tu carrera digital ~

Módulo 1

Fundamentos del desarrollo web

Introducción

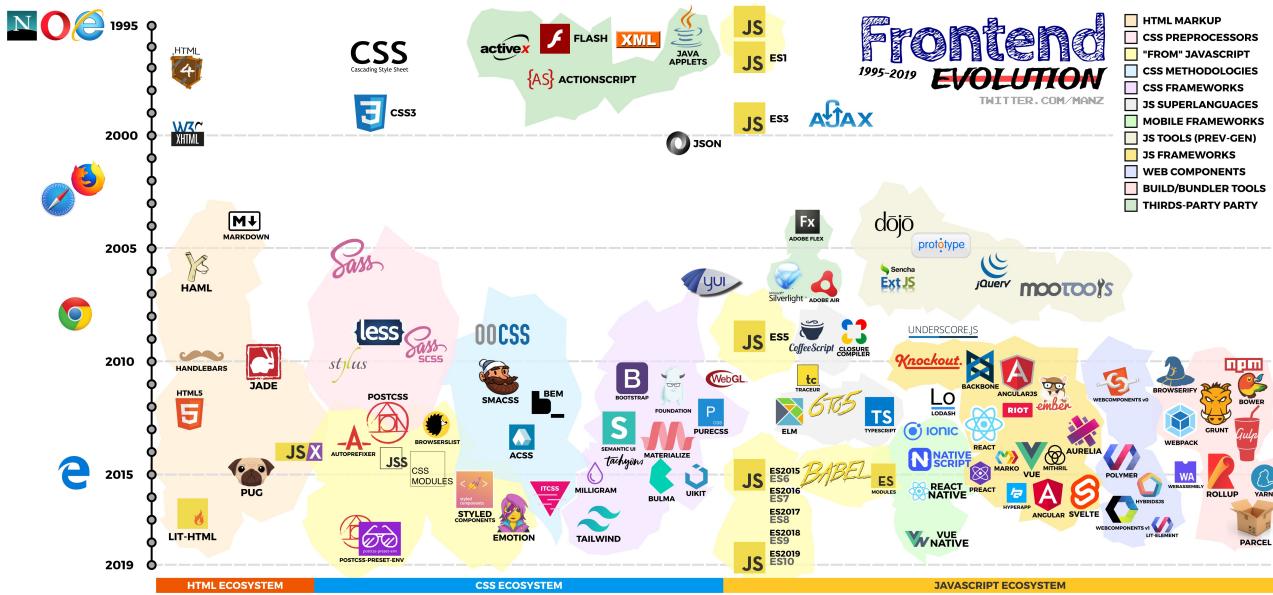


Introducción

- ◆ 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, ...).



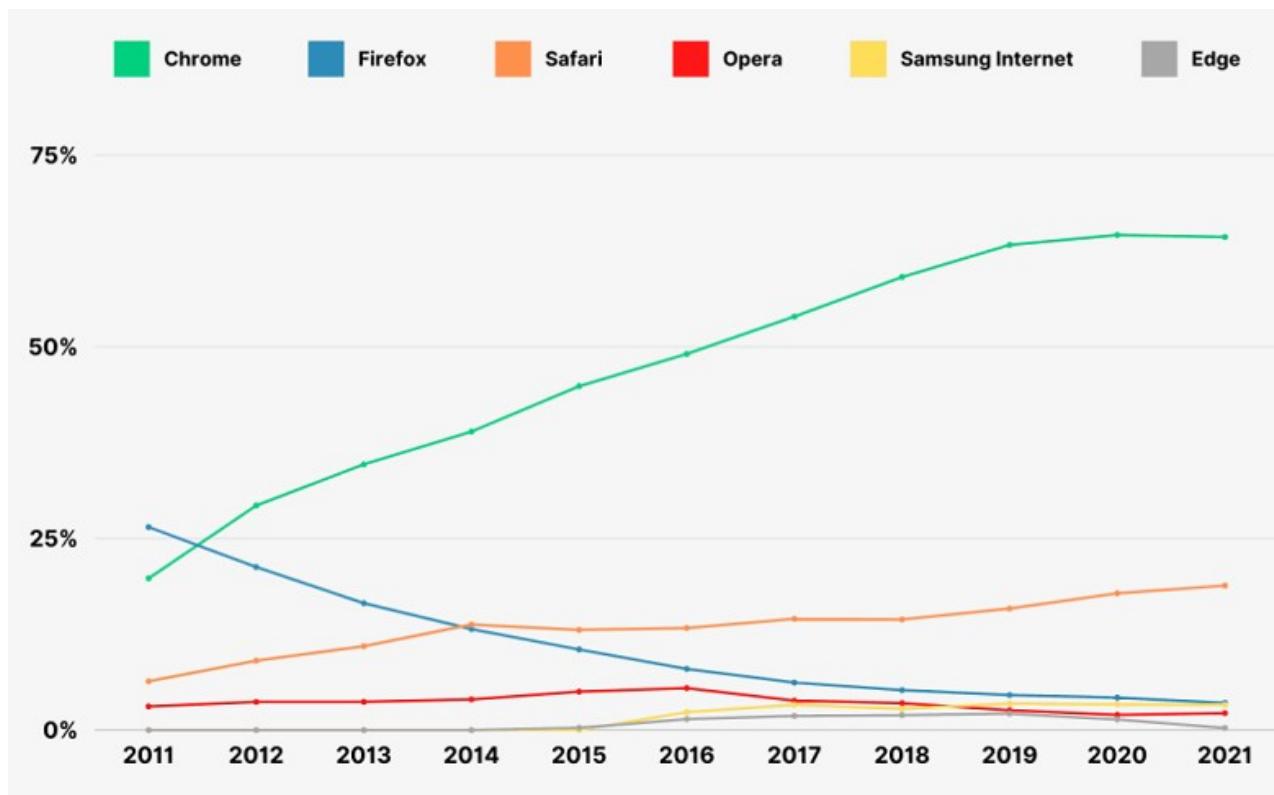
- El funcionamiento básico de la web se fundamenta en el modelo de comunicación cliente/servidor, siendo el cliente quien representa al componente consumidor del servicio, mientras que el servidor es quien provee el servicio.
- En esta comunicación cliente/servidor, el cliente es habitualmente quien 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 libra 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.

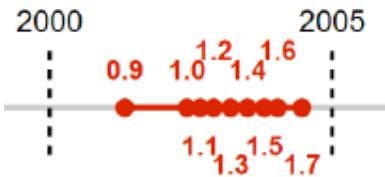
- Microsoft publicó Edge 12 en 2015 para Windows 10. De esta forma, Internet Explorer es abandonado en la versión 11, al igual que su motor de renderizado Trident. EdgeHTML pasó a convertirse en el motor de renderizado de Edge.
- En diciembre de 2018, Microsoft anunció que las futuras versiones de Edge 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 (pestanas, 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

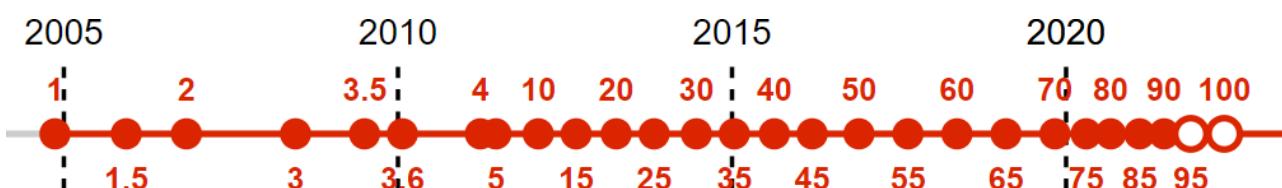
- creció hasta casi el 25% a principios de 2009. Sin embargo, la aparición de Google Chrome por esas fechas detuvo su crecimiento.

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.

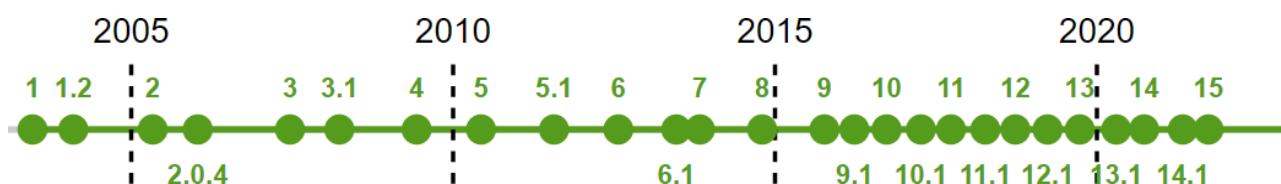
Otras iniciativas para incluir publicidad en el navegador tampoco han ayudado a Firefox a

- tratar de recuperar cuota de mercado.



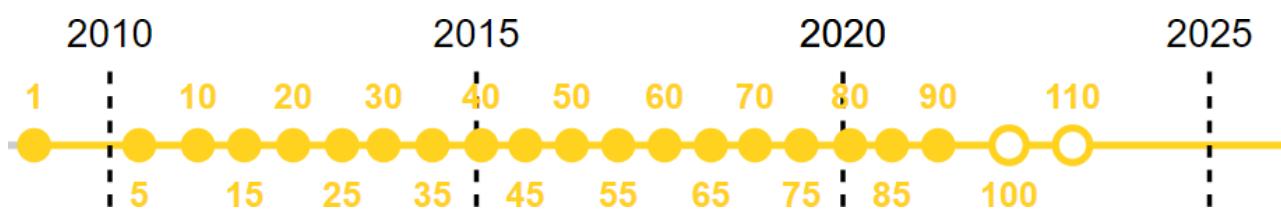
Safari

- 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

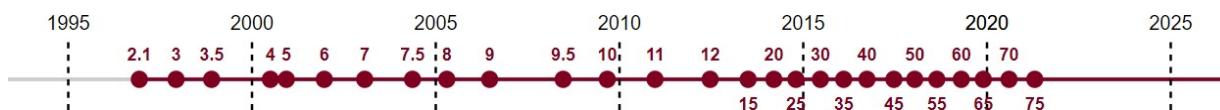
- Chrome es un navegador creado en 2008 por Google a partir de WebKit, el motor de renderizado del navegador Safari. WebKit fue sustituido por Blink a partir de la versión 28 (2013).
- Chrome ha destacado siempre por su interfaz minimalista y por la velocidad de ejecución del código Javascript.
- Los 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

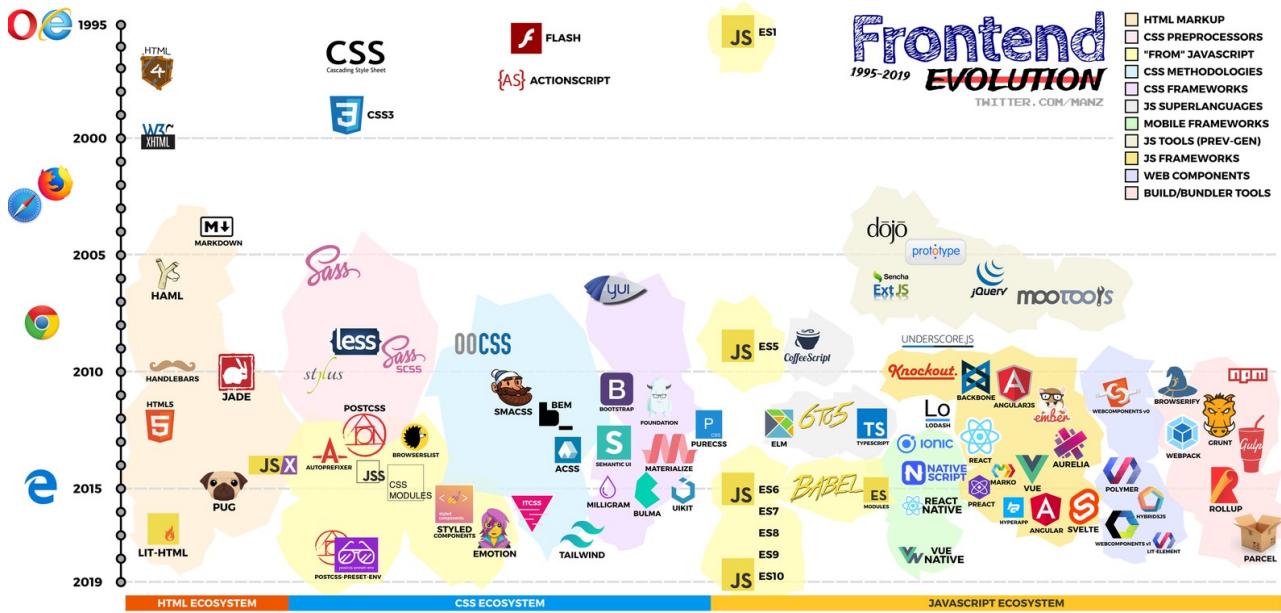
- Opera comenzó su andadura en 1994 como un proyecto de investigación de Telenor, una compañía telefónica Noruega. A partir de 1995 se constituye como Opera Software.
- La primera versión, Opera 2.1, se publicó en diciembre de 1996 y desde entonces ha ido publicando versiones tanto para PCs como para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



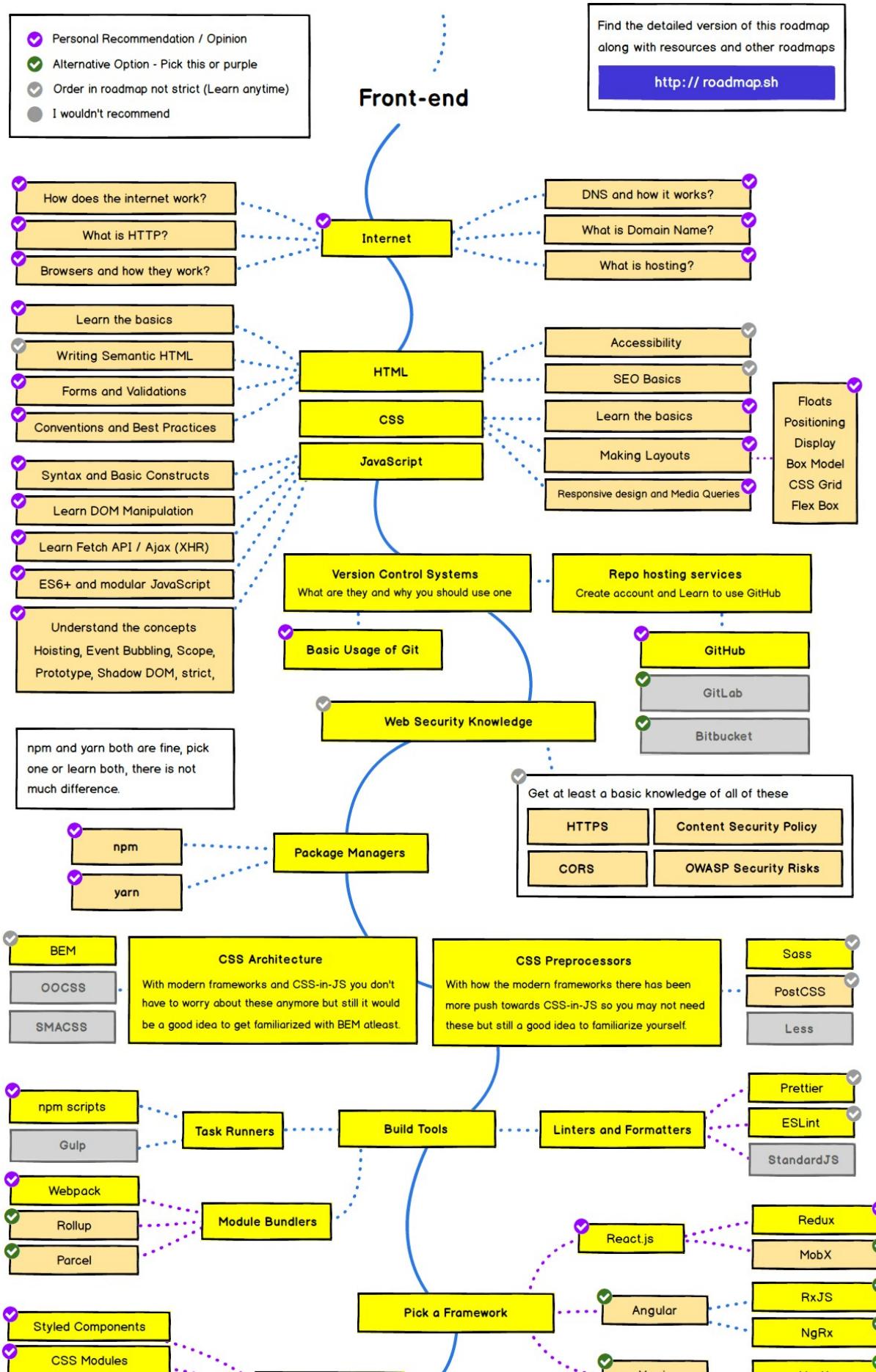
- HTML/CSS/JavaScript evolucionan con el tiempo con la inclusión de nuevas funcionalidades. La adopción de estas mejoras es más o menos rápida en función del navegador web.
 - El Backend (o lado del servidor) está compuesto por tres grandes bloques:
 - Los datos: almacenados en un sistema de almacenamiento 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).

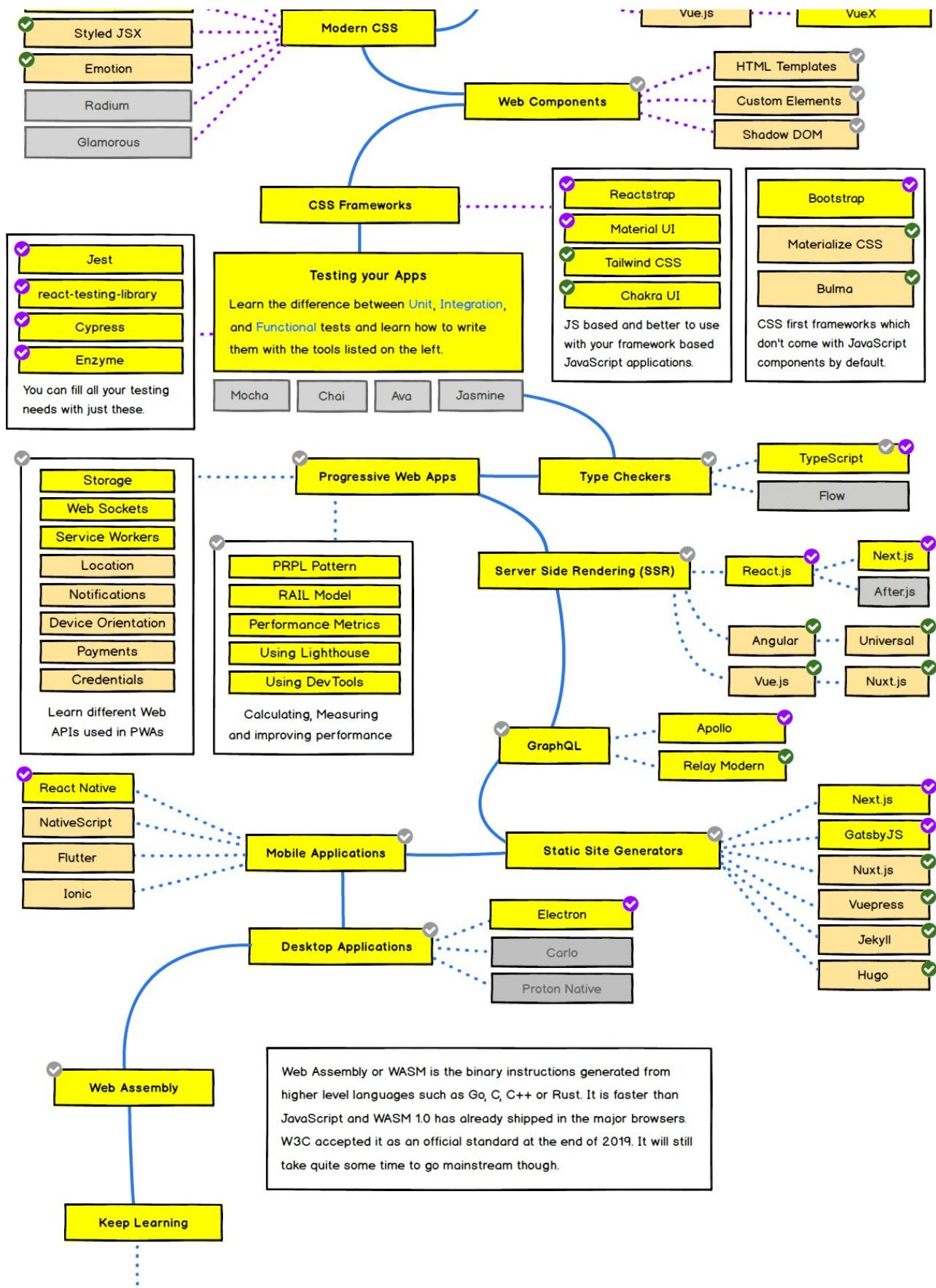
| | | | | |
|---------|--|--|--|-----------------------|
| Backend | | | | |
| Data | | | | redis |
| Infra | | | | Terraform |
| | | | | Grafana |
| | | | | Prometheus |
| | | | | Jenkins |
| | | | | Google Cloud Platform |

- 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.

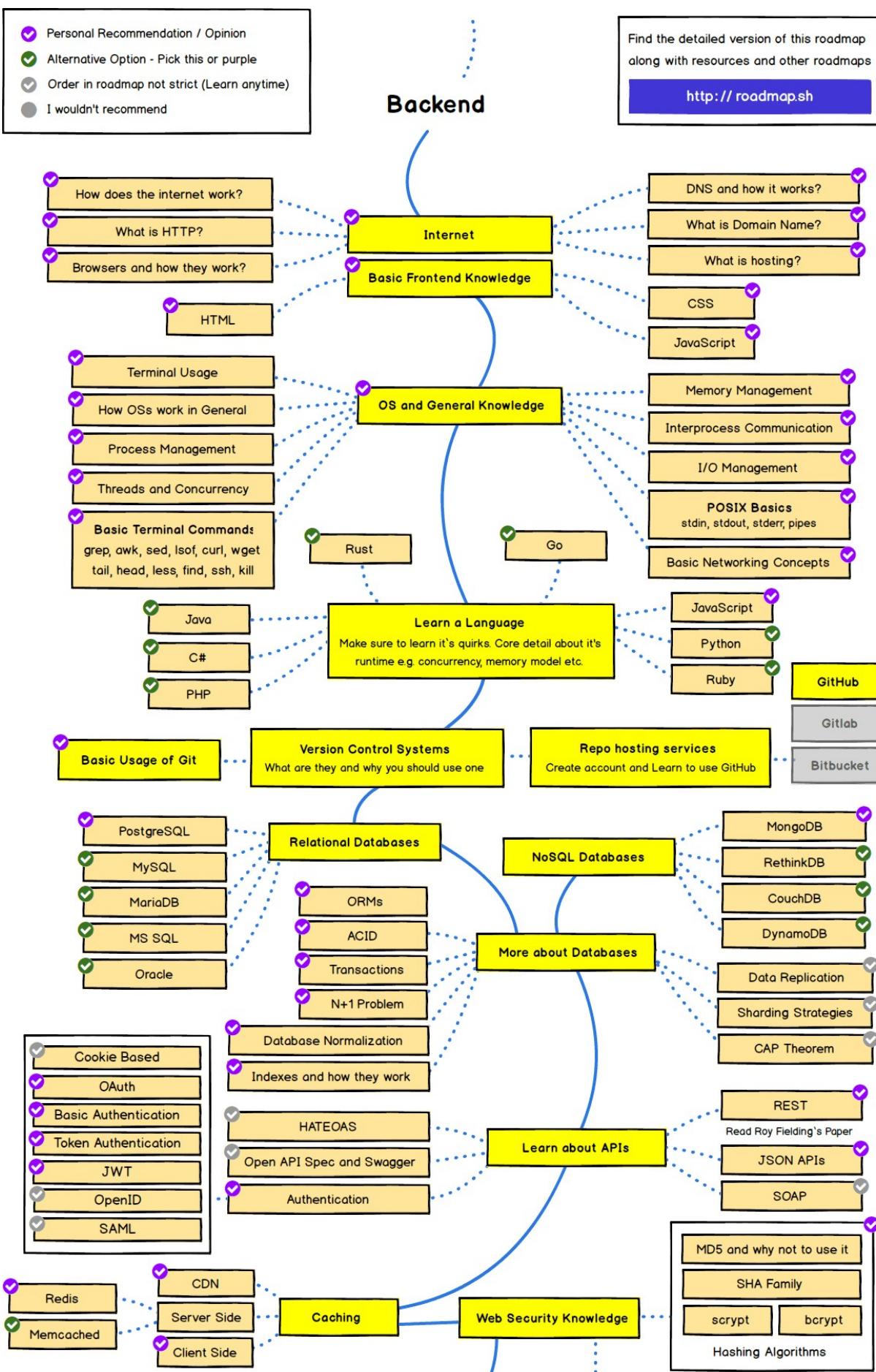
Estado actual del Frontend, Backend y Devops en la industria tecnológica

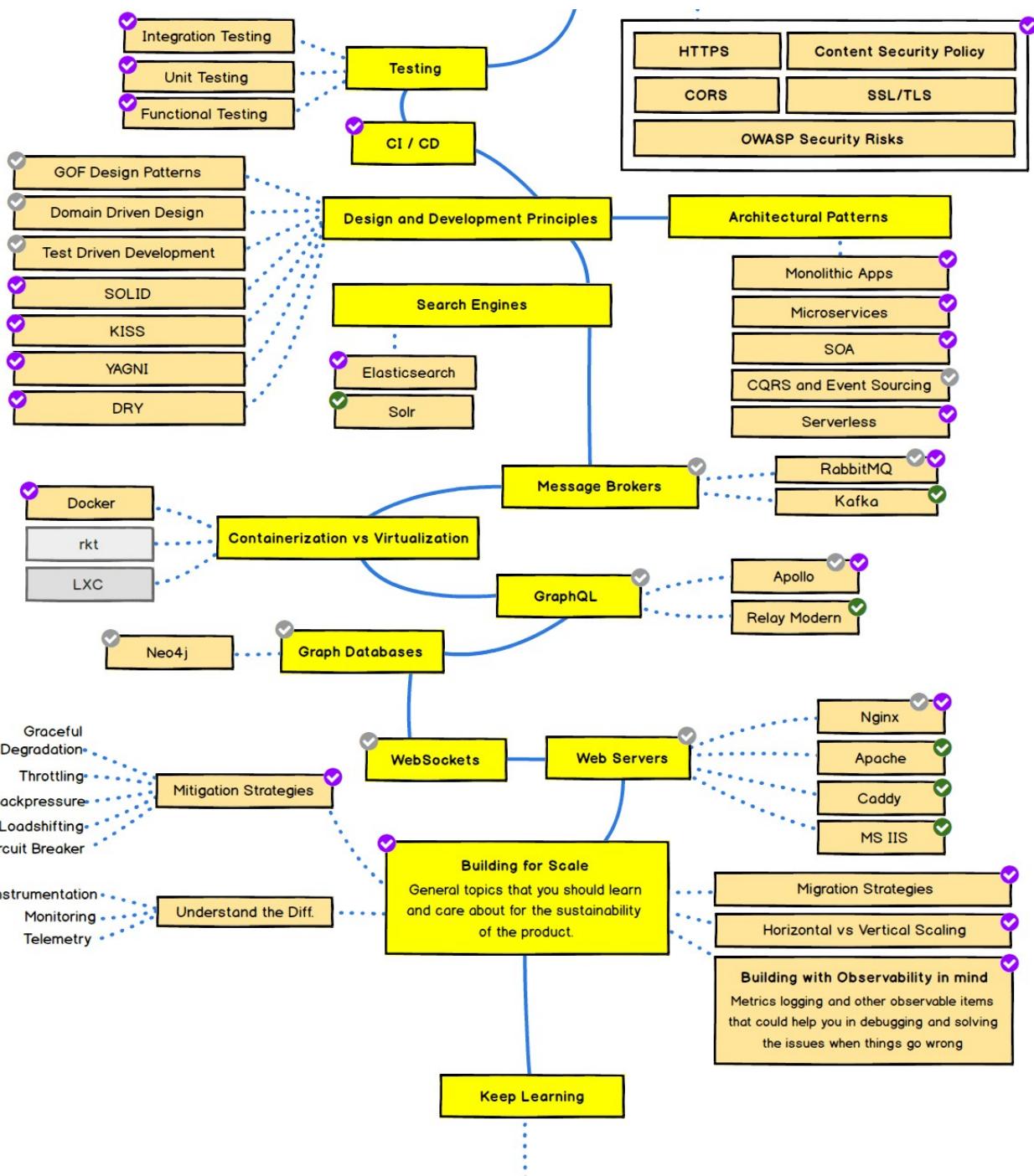
- Las rutas de aprendizaje del Frontend, Backend y Devops han ido variando a lo largo del tiempo en función de las necesidades de la industria tecnológica. Existen también importantes diferencias entre países.
- Sin embargo, hay un conjunto de tecnologías que han sido indiscutibles desde hace muchos 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





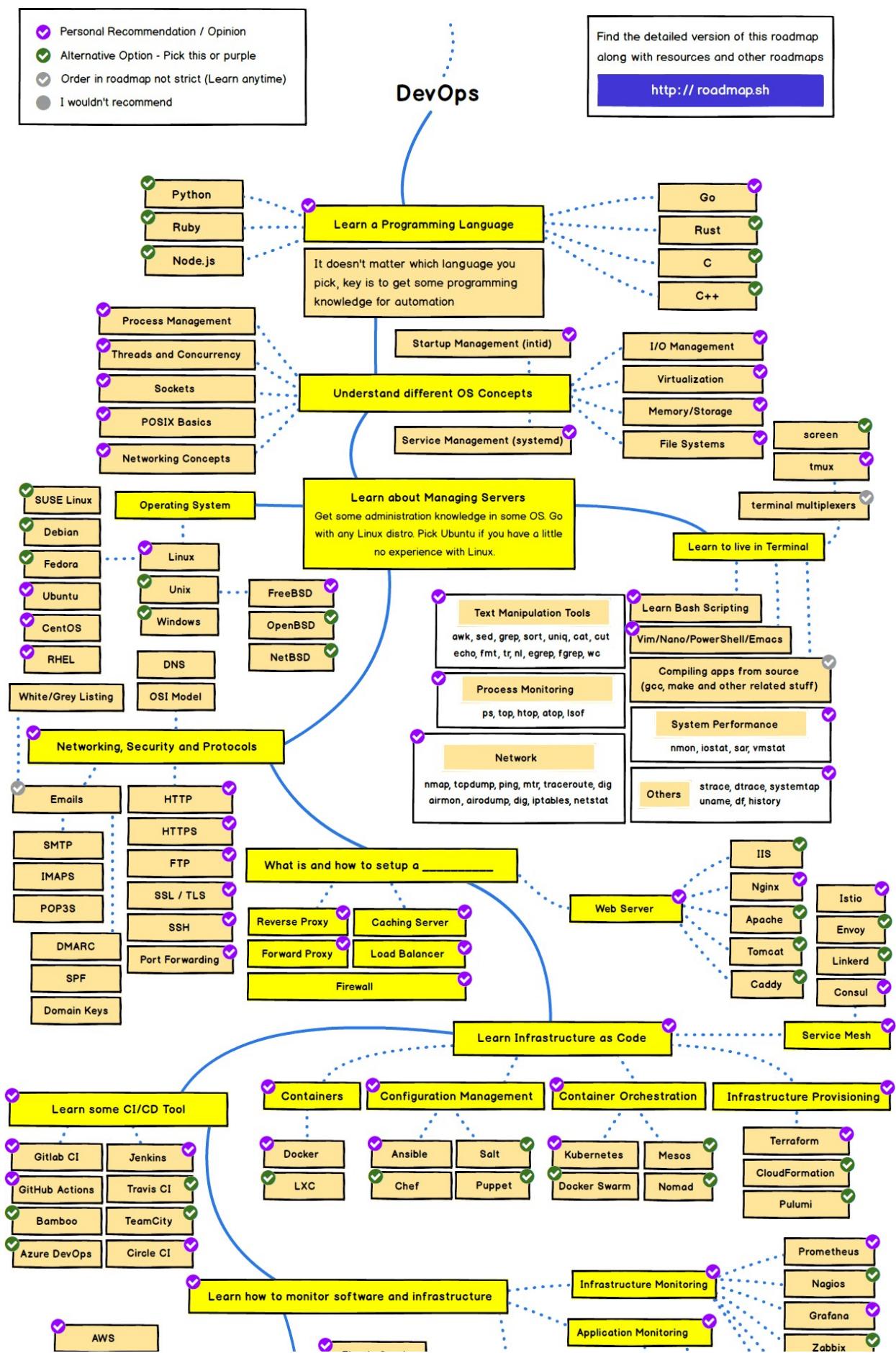
- Entre los conceptos más importantes en la ruta de aprendizaje del programador Backend se encuentran:
 - Internet.
 - Terminal de comandos.
 - Sistemas de control de versiones: Git y GitHub.
 - Seguridad de la información y los sistemas.
 - Protocolo HTTP.
 - Lenguajes de programación Backend: PHP,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

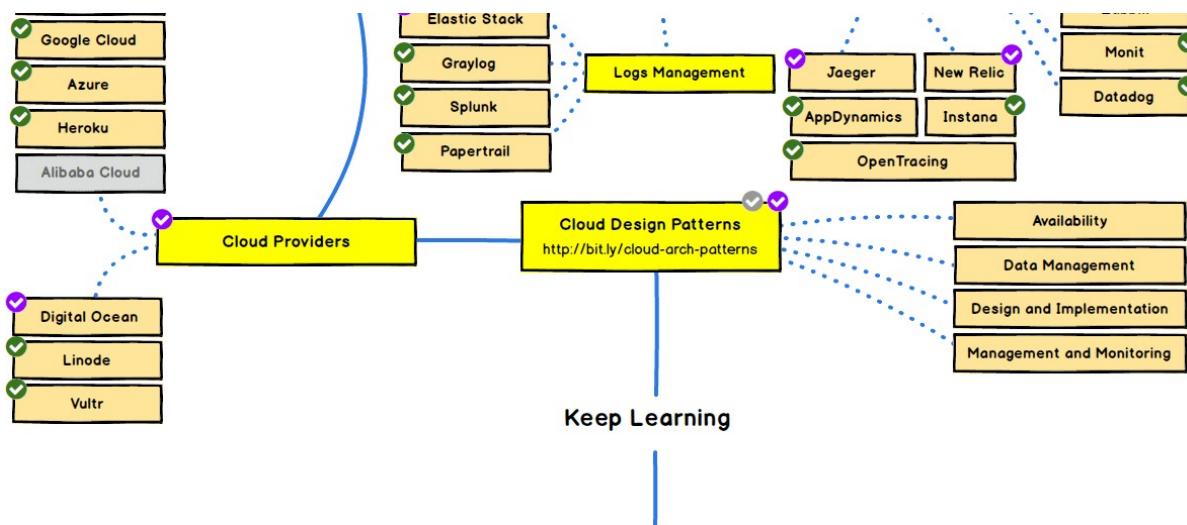




- Entre los conceptos más importantes en la ruta de aprendizaje del Devops se encuentran:
 - Internet.
 - Terminal de comandos.
 - Sistemas de control de versiones: Git y GitHub.
 - Seguridad de la información y los sistemas.
 - Pila de protocolos TCP/IP
 - Lenguajes de programación para administración 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](#).

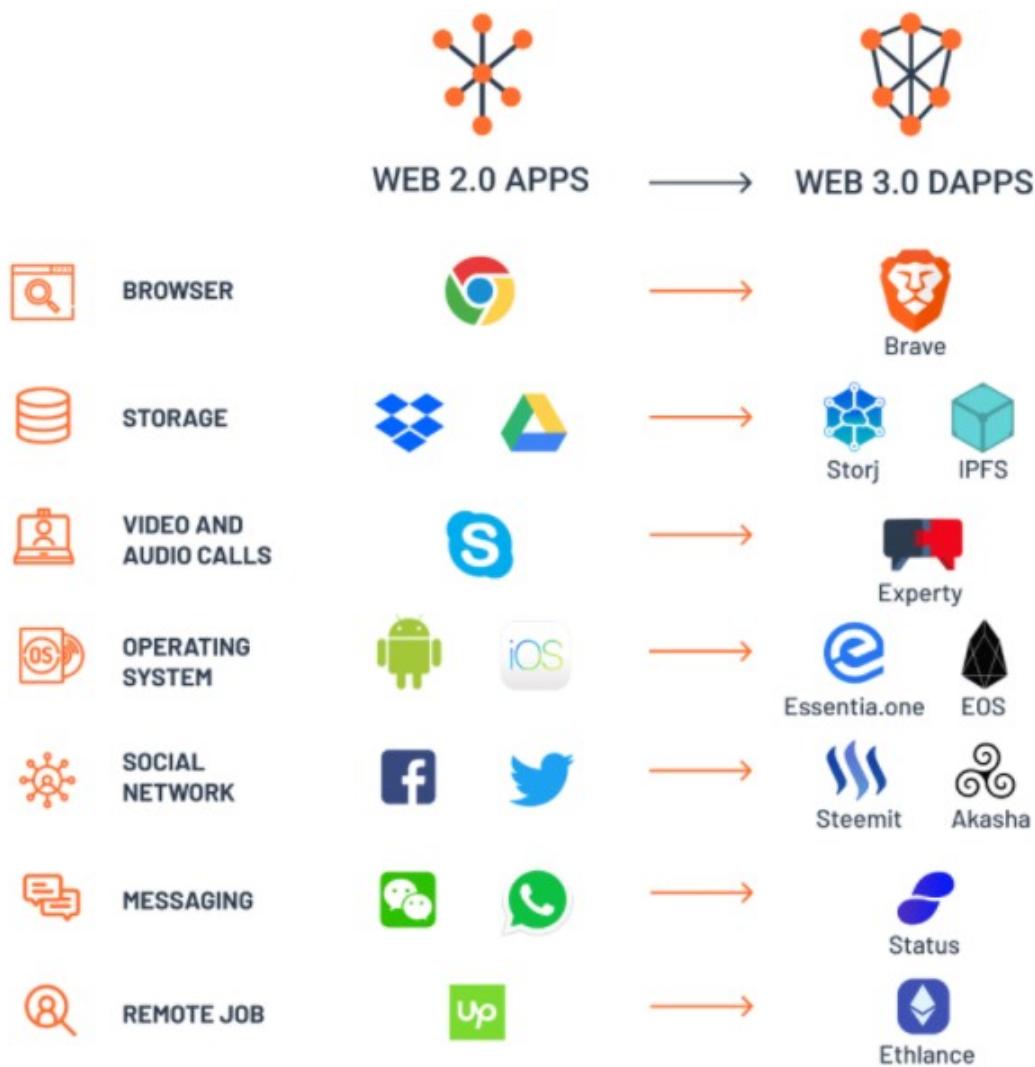
- Web Neural Network API: define una capa de abstracción independiente del hardware para hacer uso de las capacidades de aprendizaje automático y aprendizaje profundo.
 - La capa de abstracción aborda los requisitos de los marcos clave de JavaScript de aprendizaje automático y también permite a los desarrolladores 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.



— MGZ —

- 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](#)

Tu carrera digital ~

Módulo 1

Fundamentos del desarrollo web

Breve introducción a la línea de comandos



Breve introducción a la línea de comandos

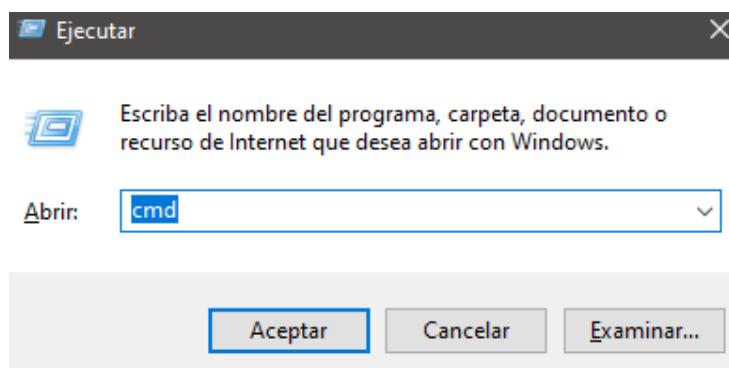
- Abrir una terminal de comandos
- Comando *cd*
- Comando *mkdir*
- Comando *rmdir*
- Comando *dir*
- Otros comandos

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.

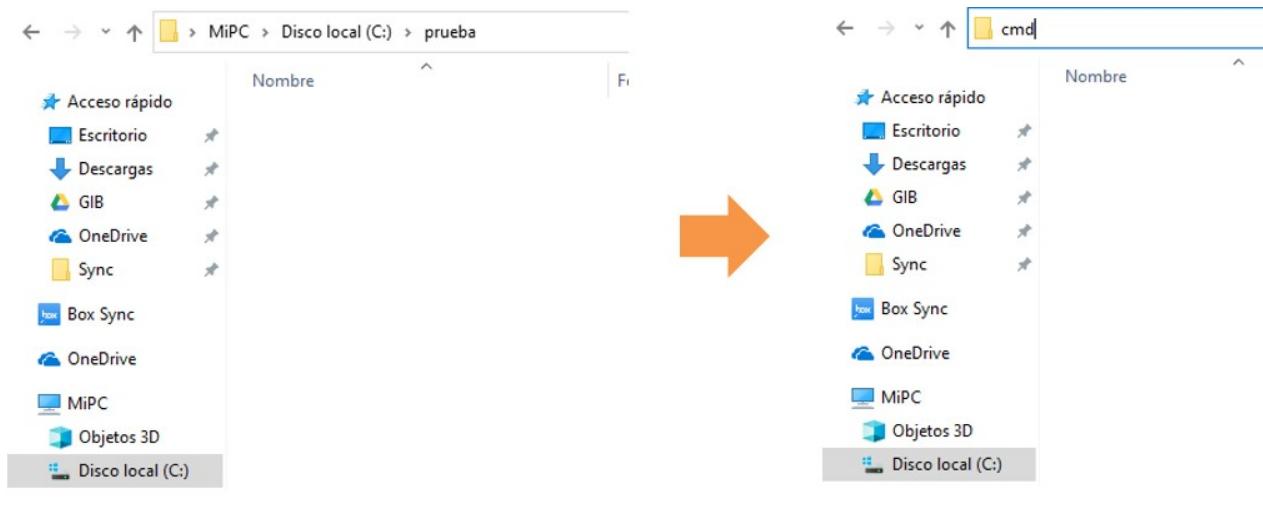


- La terminal de comandos se abre por defecto en la ruta c:\Users\USUARIO

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

C:\>
```

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

C:\>
```

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*.

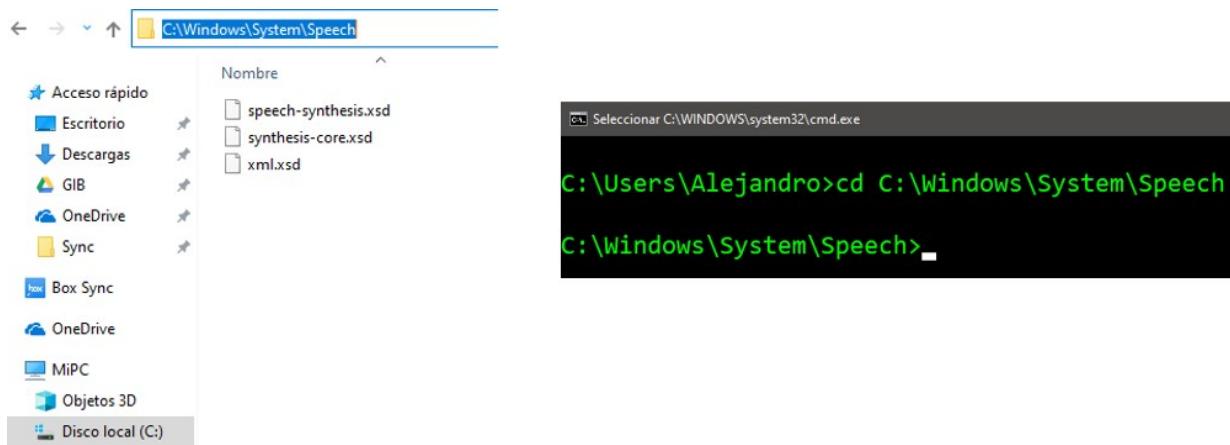
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

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\System32\cmd.exe
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

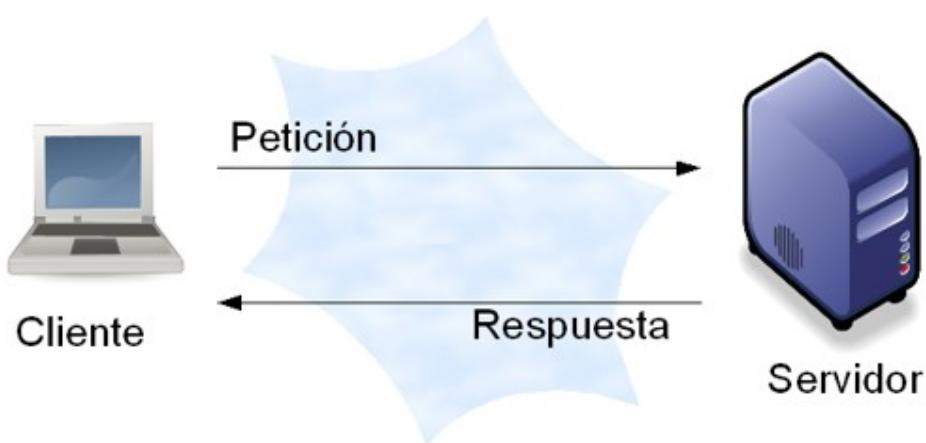


Introducción a la pila de protocolos TCP/IP

- ◆ 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

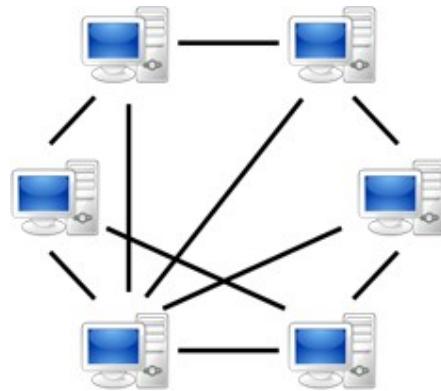
- ◆ La comunicación en Internet se basa en una arquitectura muy simple denominada cliente/servidor.
- ◆ El cliente inicia la comunicación mediante una petición.
- ◆ El servidor se encuentra "escuchando" peticiones y responde cuando recibe una.
- ◆ La comunicación finaliza cuando el servidor responde.



- ◆ 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

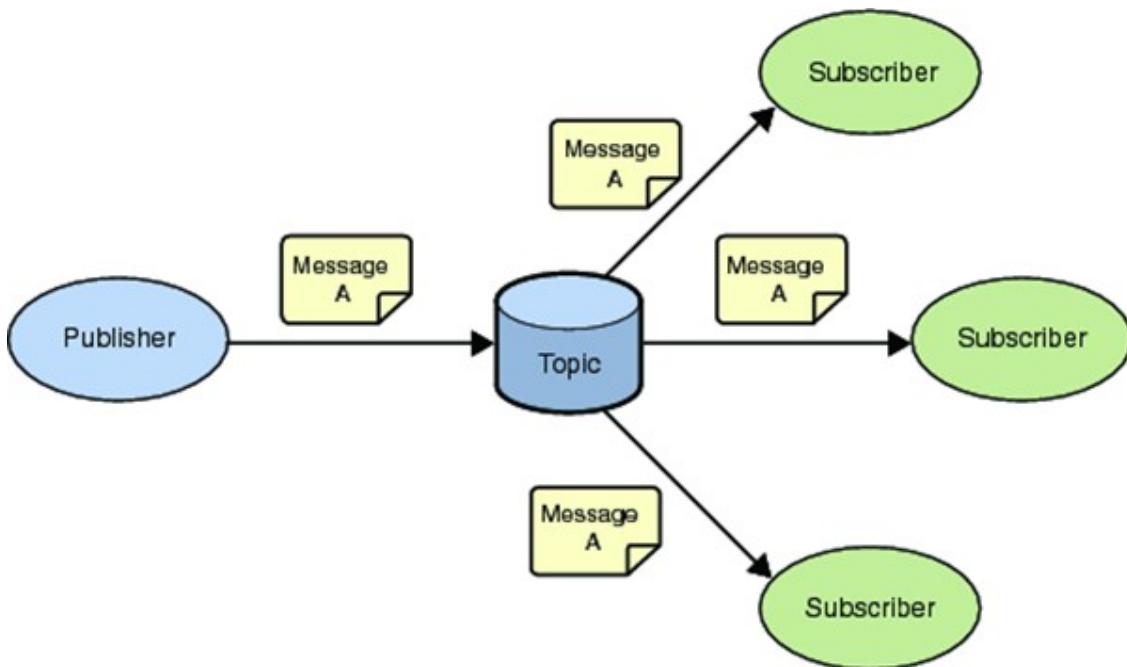
- Arquitecturas más complejas son construidas a partir del cliente/servidor.
- Por ejemplo, en las redes por pares (Peer-to-peer o P2P), todos los nodos de comunicación ejercen de cliente y servidor al mismo tiempo.



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

Arquitectura de comunicación publicador/suscriptor

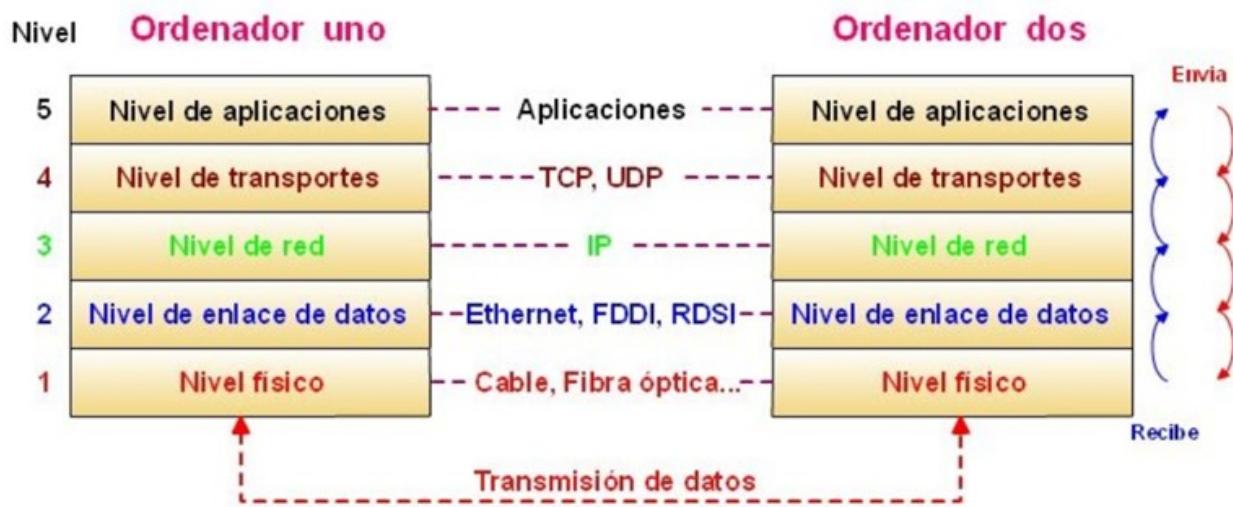
- Otra arquitectura importante que ha tenido gran auge en los últimos años es la Publicador/Suscriptor.



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

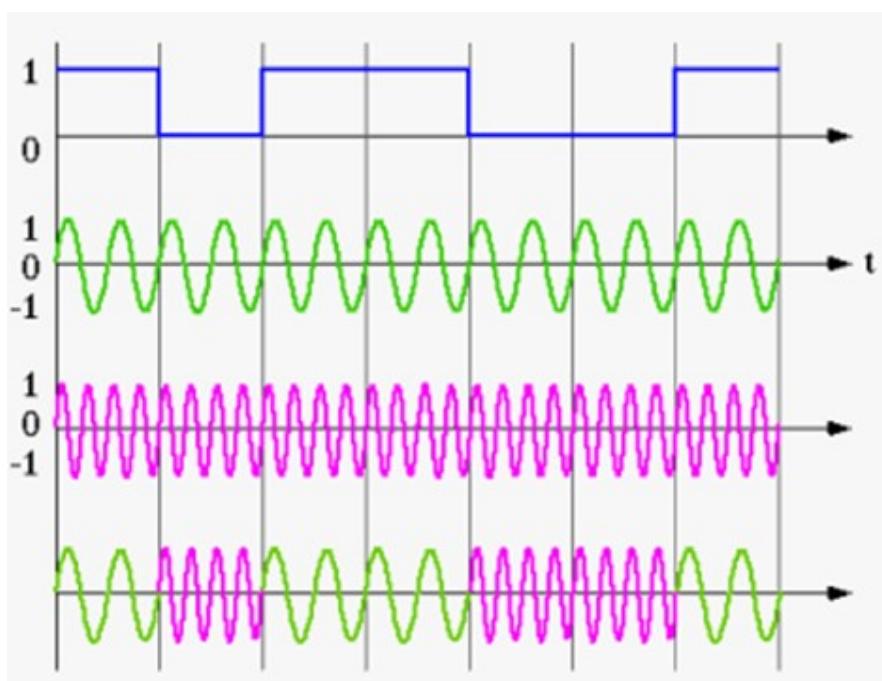
Modelo de capas TCP/IP

- 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.).

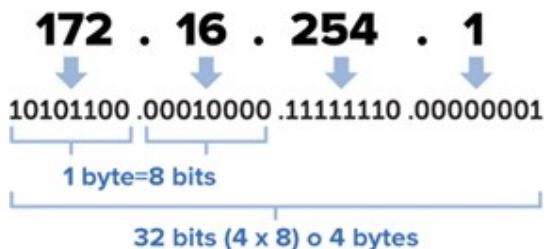


Nivel de enlace de datos

- 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.



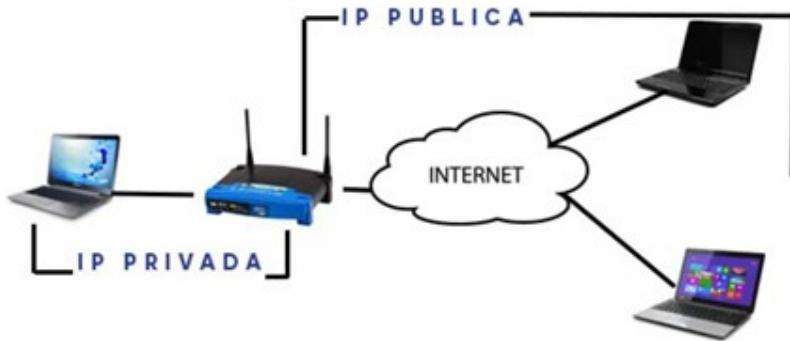
- El rango completo de direcciones IP abarca desde la dirección IP 0.0.0.0 a la 255.255.255.255.
- Existen direcciones especiales como la 127.0.0.1 (dirección de loopback o retorno), utilizada para realizar comunicaciones hacia el propio equipo.
- La dirección de loopback se utiliza principalmente para:
 - 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.

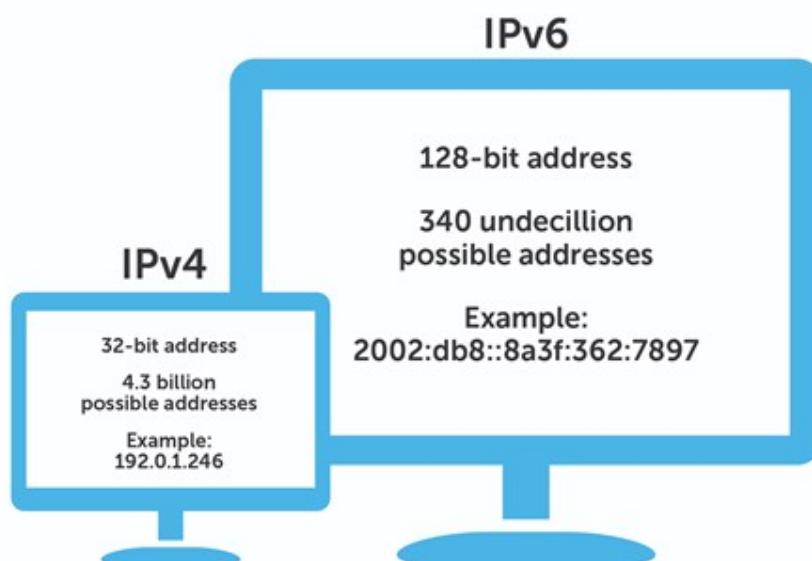
Dirección IP de www.corenetworks.es

```
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:

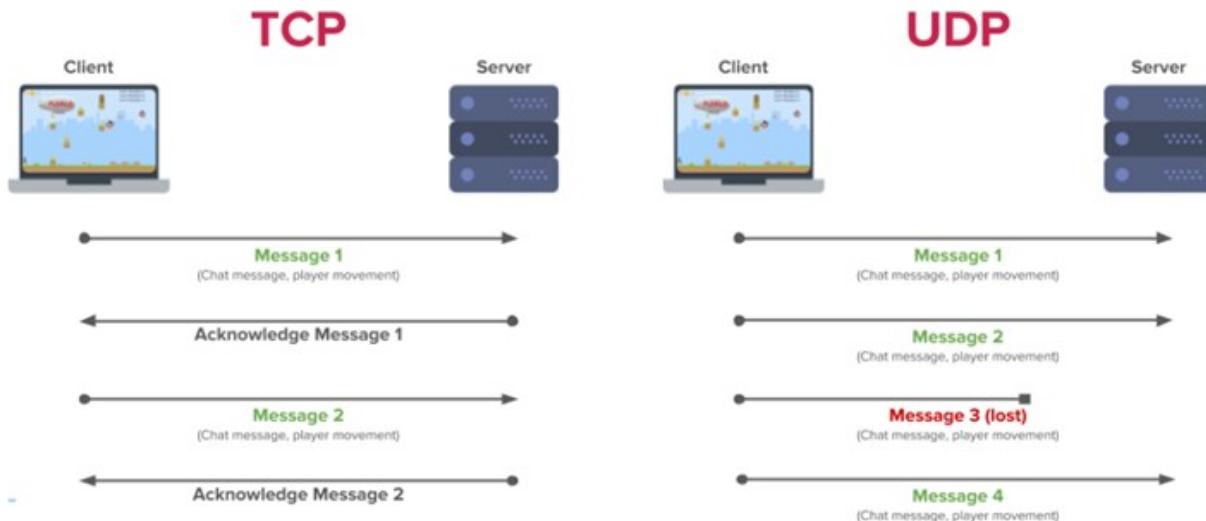
```
Sufijo DNS específico para la conexión. . . :  
Vínculo: dirección IPv6 local. . . : fe80::691e:16f6:96ea:e27%15  
Dirección IPv4. . . . . : 192.168.178.1  
Máscara de subred . . . . . : 255.255.255.0  
Puerta de enlace predeterminada . . . . . :
```

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.

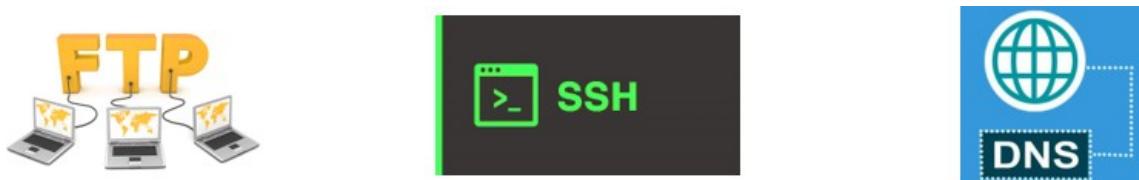


- Un puerto es un número comprendido entre 0 y 65535.
- Cualquier aplicación puede abrir un puerto TCP o UDP y esperar las peticiones entrantes de clientes.
- El rango de puertos puede dividirse en tres grupos.
 - Puertos bien conocidos (0-1023):
 - 21 (TCP): FTP.
 - 23 (TCP): Telnet.
 - 25 (TCP): SMTP.
 - 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).

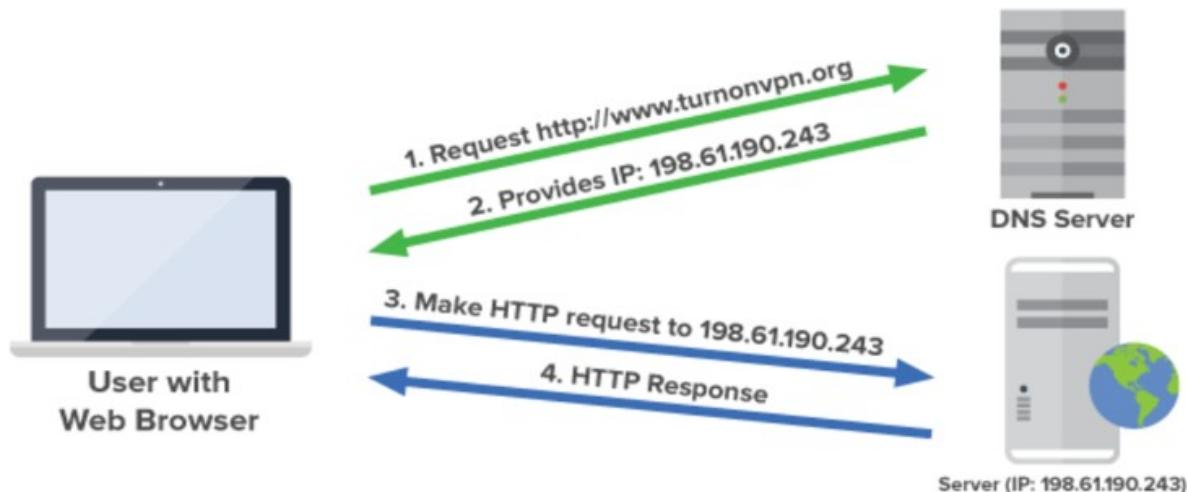
Nivel de aplicación

- 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.

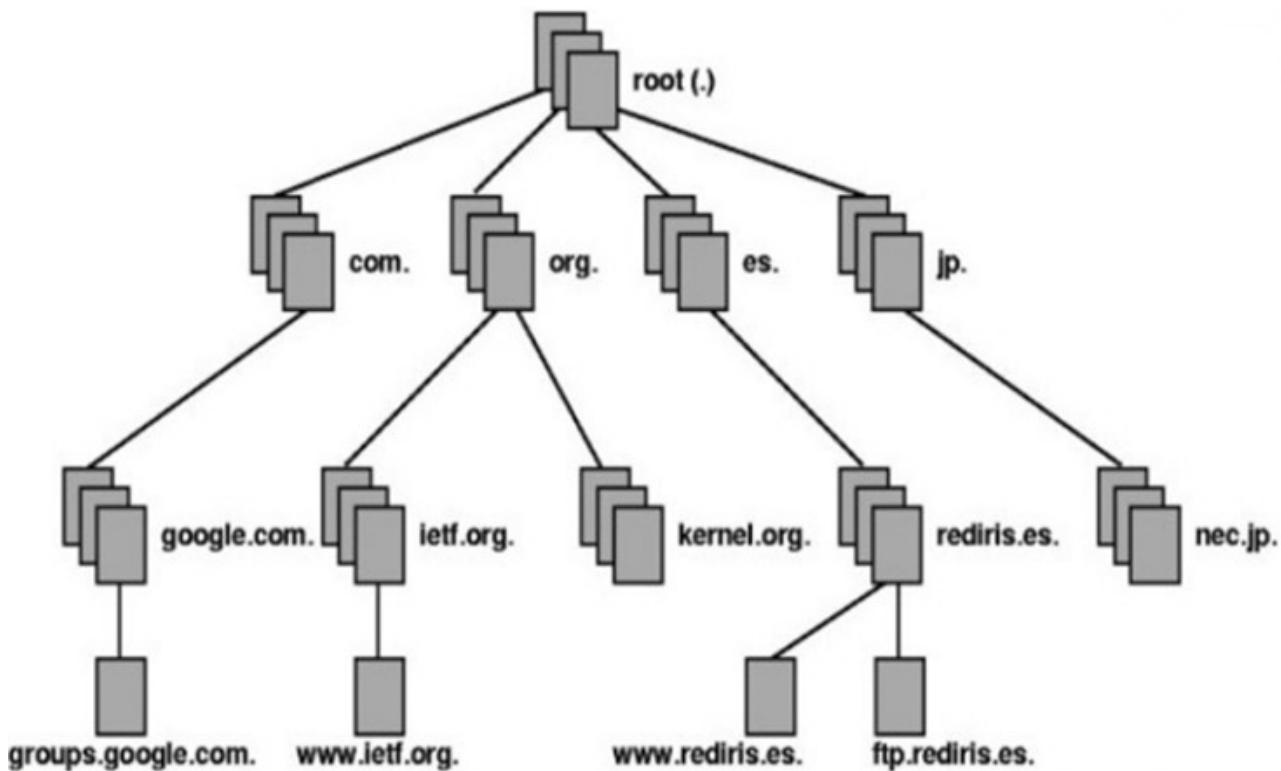


El protocolo DNS

- 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 | acae.us.es | General DNS re |
|-------------------------|---|--------------|--|---|
| ASN – 6 | acae.us.es | | IP: 31.200.242.174 | Title: |
| 198096 (7,671) | IP: 31.200.242.174 | Spain 60494 | Geo: Spain | // There is no <title> tag on the site |
| 766 (495) | Unelink Telecom, S.A. | | AS Number: 60494 | |
| + more | | | AS Organization: Unelink Telecom, S.A. | |
| IP – 30+ | gestioneventos.us.es - Eventos - Universidad de Sevi... | | Subdomains: 1 | |
| 150.214.230.47 (55) | IP: 87.253.228.167 | Spain 48846 | | Meta description: |
| 150.214.9.141 (15) | Informatica El Corte Ingles SA | | | // There is no meta-description on the site |
| + 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 | vpnclient9.us.es | | | |
| | IP: 150.214.9.5 | Spain 198096 | | |
| | Junta de Andalucía | | | |

Subdomains

Total – 1

Search FILTER 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.

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

General headers

Entity headers

- ◆ 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
```

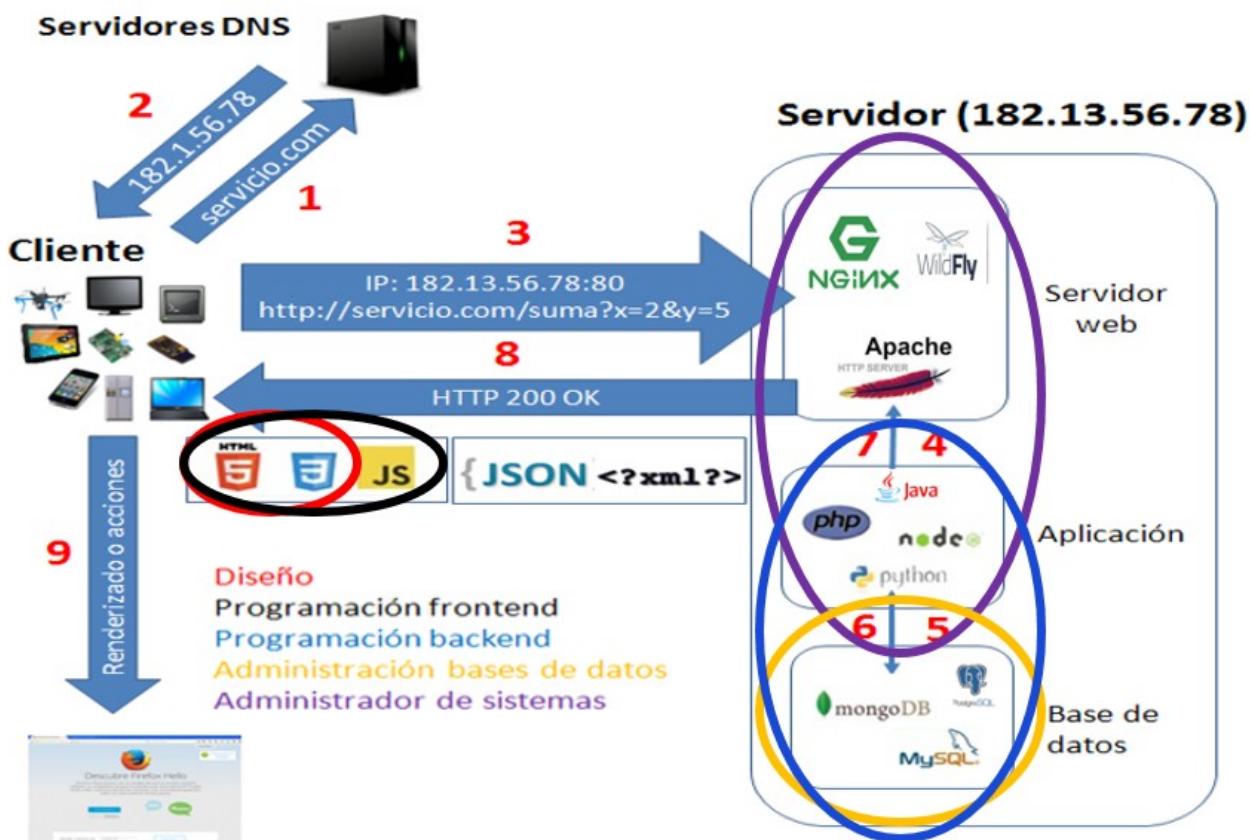
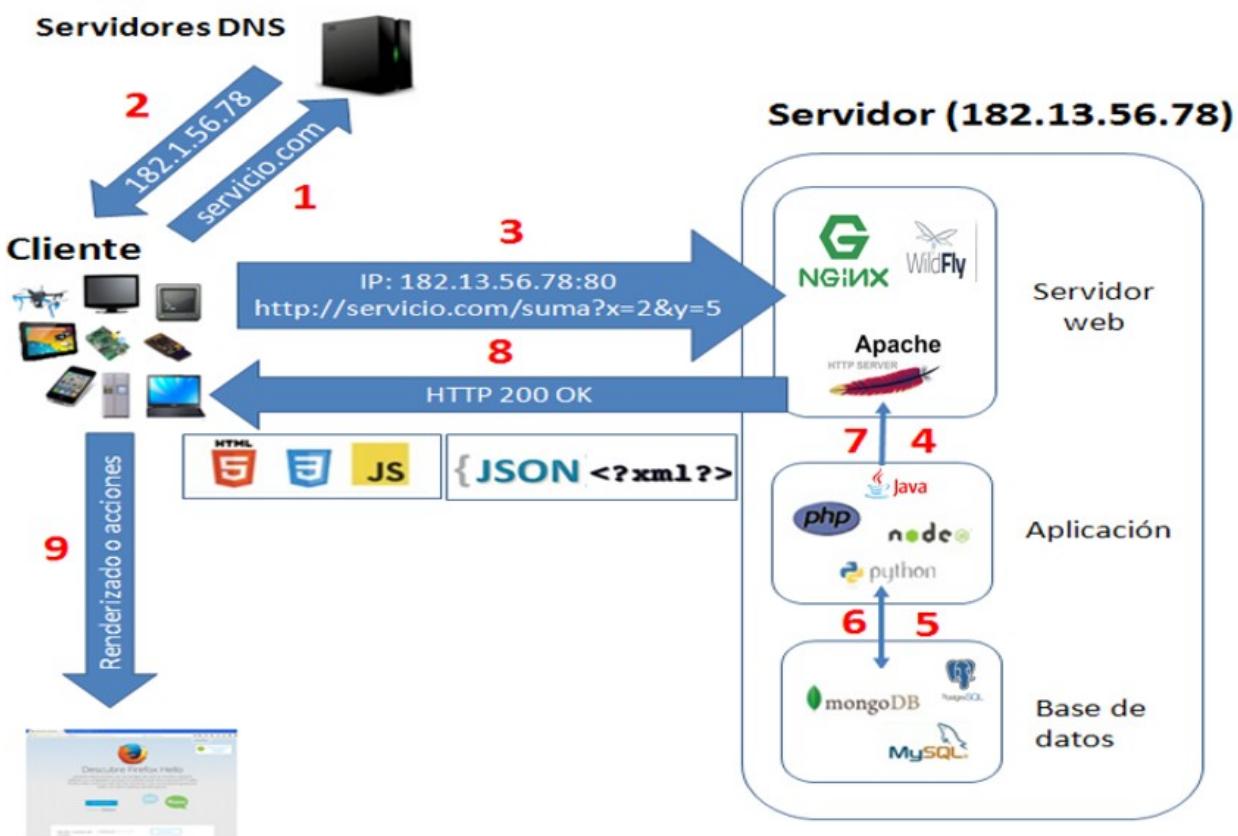
```
POST /webmail/login.phtml HTTP/1.1
Host: www.abv.bg
Accept: */*
Accept-Language: bg
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 59
Body:
<CRLF>
LOGIN_USER=mente
DOMAIN_NAME=abv.bg
LOGIN_PASS=top*secret!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

The screenshot shows the Postman application interface. On the left, there's a sidebar with a history of API requests. The main area is titled "Untitled Request" and shows a GET request to "https://swapi.co/api/films/2/". Under the "Params" tab, there is a single parameter "a" with value "b". The "Body" tab shows the JSON response:

```

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 pursue them across\nthe galaxy.\n\nInvading 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/",
```

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 sobreescibir 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 *Use 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 install 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

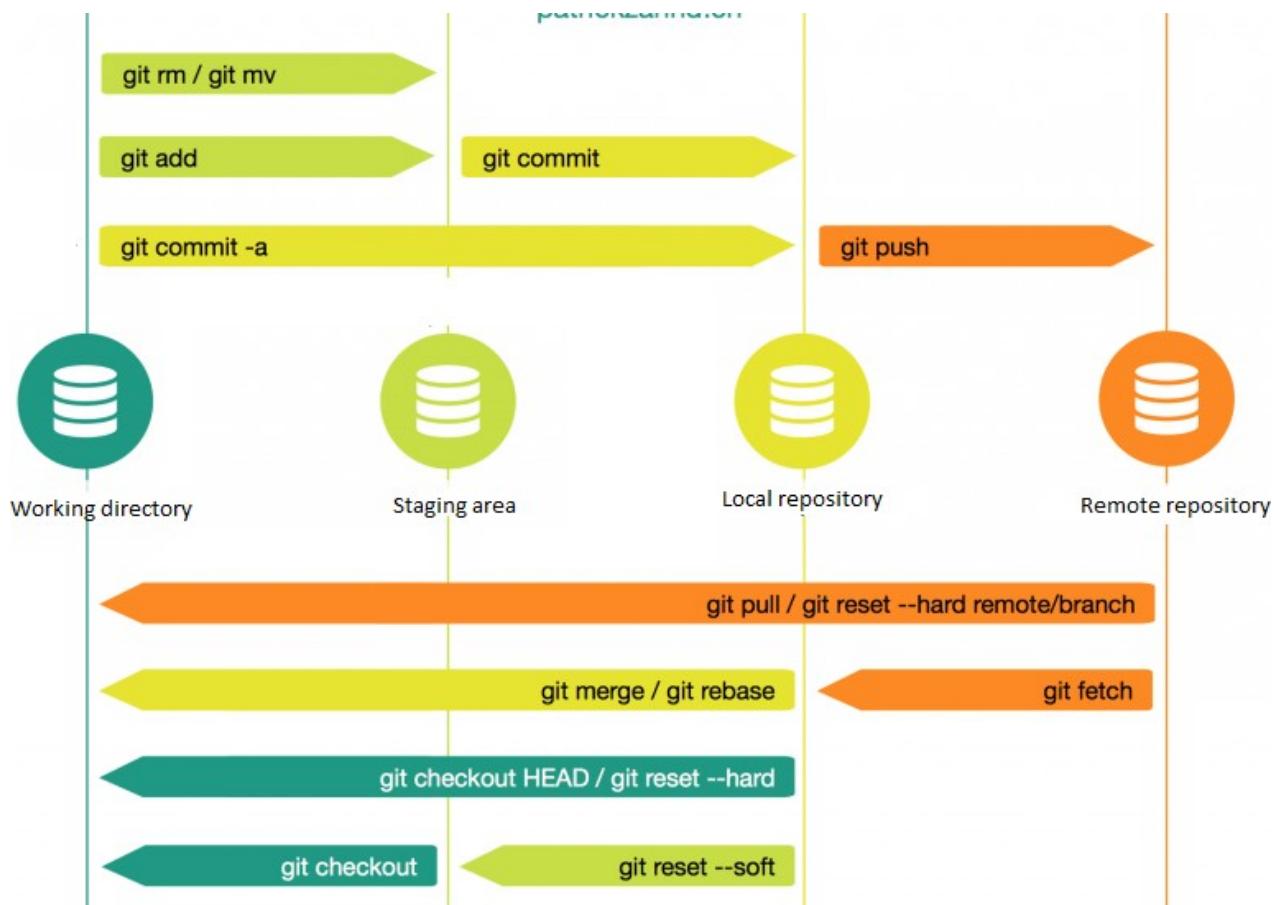
- Usuarios (global): ajustes específicos del usuario (con --global). Se almacena en `~/.gitconfig`
- Todo el sistema (sistema): ajustes de todo el sistema (con --system). Se almacena en `/etc/gitconfig`

```
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.

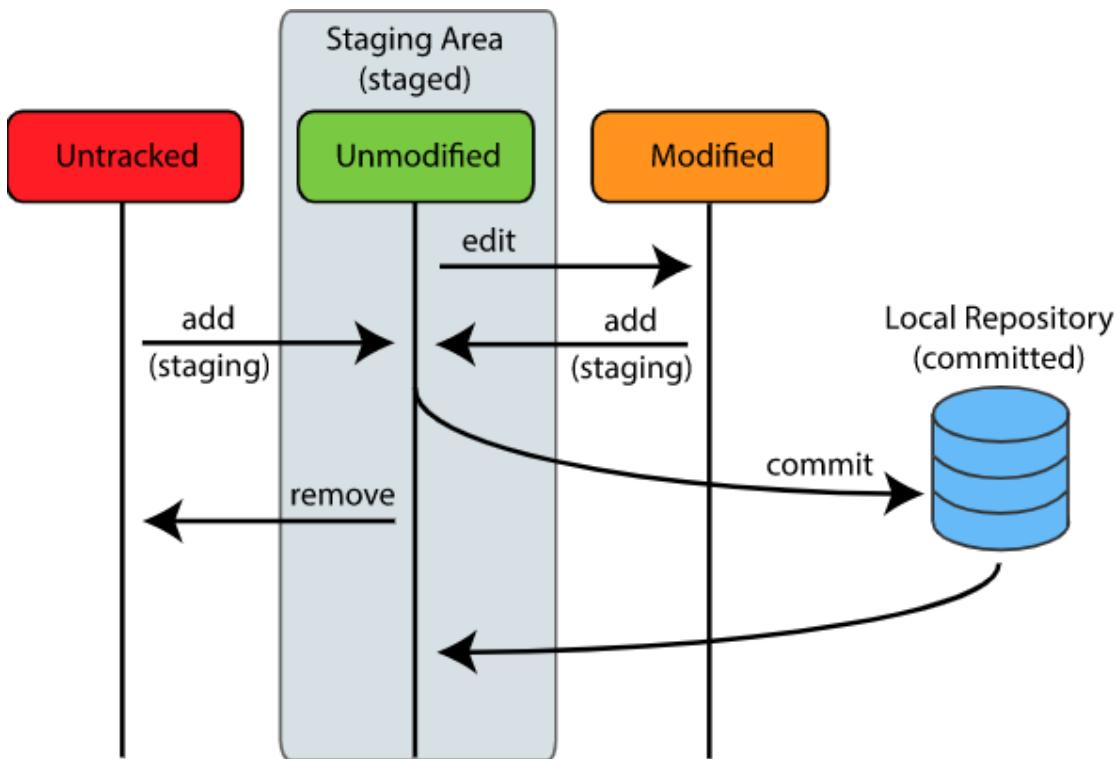
```
# inicializa un repositorio de Git nuevo  
git init
```

- 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 el historial de Git hasta el commit establecido, pero mantiene los cambios en el *Staging Area*
 - *git reset --hard TAG_COMMIT*: borra 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:

Menor que: <

Mayor que: >

Ampersand: &

Comillas dobles: "

Comillas simples: '

Ejemplo de etiqueta: <body>

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 indentado.
- 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 -->

[Saltar al capítulo 4 de la página index.html](index.html#C4)

- 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 vídeo, puede utilizarse el atributo autoplay (parece funcionar en Firefox, pero no en Chrome).
 - Para reproducir el vídeo 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 vídeo (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">
  Tu navegador no es compatible con la etiqueta audio.
</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)

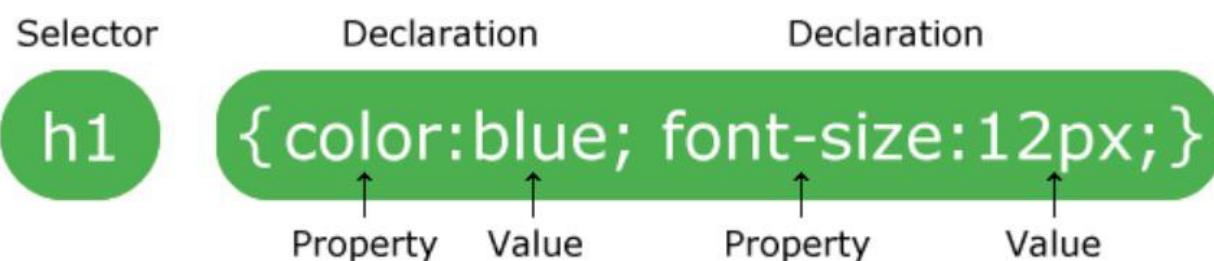


CSS básico (I)

- ◆ 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.



```
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>
```

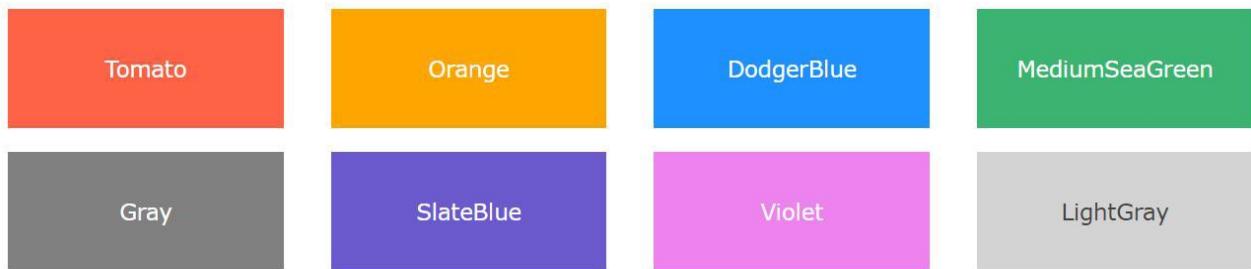
El selector class

- 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;
}
```

```


- 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>
</pre>

```



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



```

<style>

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

div {
 color: red;
}

```



90


```

```
}
```

```
</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 {  
    font-weight: 100;  
    color: white;  
}  
  
.red-and-bold {  
    font-weight: bold;  
    color: green;  
}  
  
div {  
    color: red;  
}  
  
div {  
    display: block;  
}
```

a.html:4 a.html:10 a.html:16 user agent stylesheet

- 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>
```

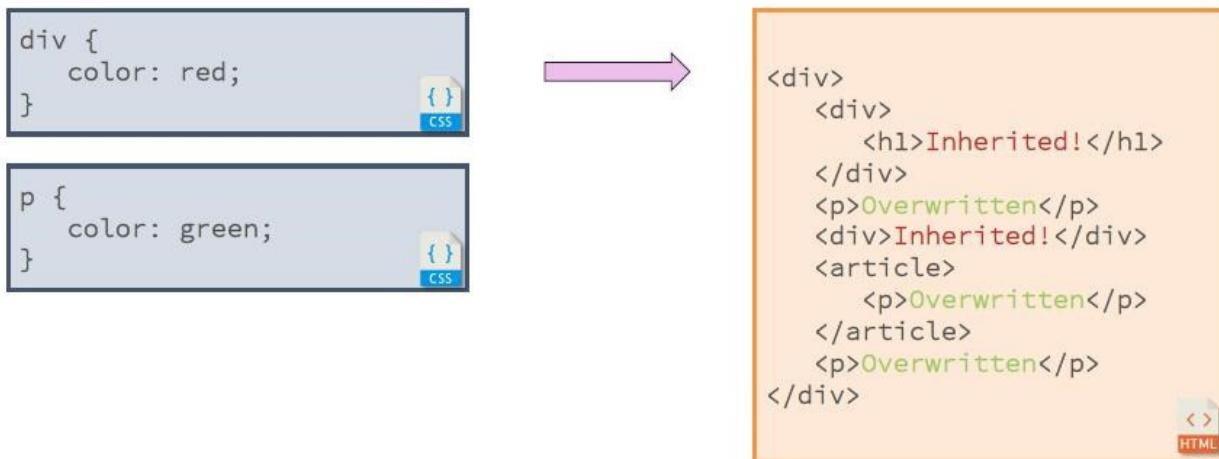
- Sin embargo, las propiedades que se heredan tienen una especificidad muy baja, más aún que los selectores establecidos automáticamente por el navegador. Cualquier selector sobre un elemento tiene más especificidad que uno que se hereda.

```
<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>
```



- 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 > div > #green-and-lighter -->
<!-- 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 -->
<div id="red-and-bold">
    Hola
</div>
</div>
```

- 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 --&gt;
&lt;p class="centrar red"&gt;Hola&lt;/p&gt;

<!-- aplican los estilos del selector .centrar --&gt;
&lt;p class="centrar"&gt;Buenos días&lt;/p&gt;</pre>

```

- 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">
```

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

```
<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>
</div>

```

```
<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 (CSS)

CSS básico (II)

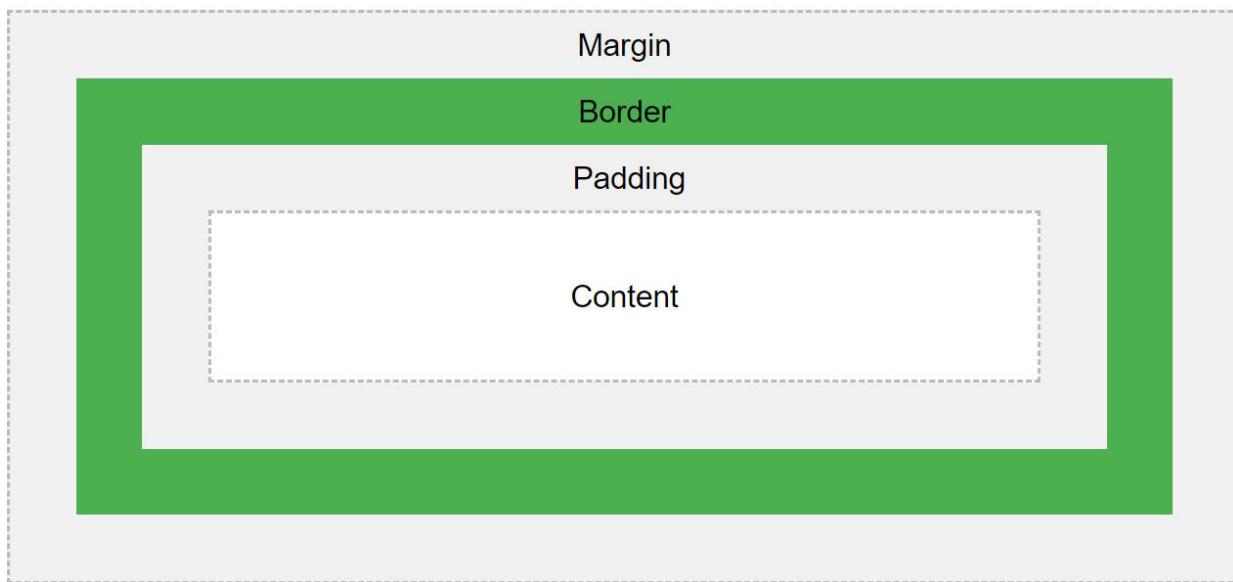


CSS básico (II)

- ◆ Modelo de caja
- ◆ Colapso del margen
- ◆ Propiedades width/height
- ◆ Propiedad display (block e inline)
 - Propiedad display (inline-block)
 - Propiedad display (none) y propiedad visibility (hidden)
- ◆ Propiedades CSS para texto
 - Propiedad text-align (texto)
 - Propiedad text-decoracion
 - Propiedad text-transform
 - Propiedad text-indent
 - Propiedad letter-spacing
 - Propiedad line-height
 - Propiedad text-direction
 - Propiedad word-spacing
 - Propiedad text-shadow
- ◆ Propiedades CSS para fuentes
 - Propiedad font-family
 - Propiedad font-style
 - Propiedad font-size
 - Propiedad font-weight
 - Propiedad font-variant
- ◆ Google Fonts
- ◆ Propiedad vertical-align

Modelo de caja

- ◆ Todos los elementos HTML se pueden considerar como cajas. En CSS, el término "modelo de caja" se utiliza en la maquetación.
- ◆ El modelo de caja CSS es esencialmente una caja que envuelve cada elemento HTML. Una caja está formado por: márgenes, bordes, relleno (padding) y el contenido.



- Explicación de las diferentes partes:

- Contenido: es el contenido de la caja, es decir, lo que contiene el elemento HTML.
- Padding: área entre el borde y el contenido. Es transparente.
- Borde: borde que rodea el padding y el contenido.
- Margen: área fuera del borde. Es transparente.
- El ancho total del un elemento = width (ancho del contenido) + left padding + right padding + left border + right border + left margin + right margin.
- La altura total del un elemento = height (altura del contenido) + top padding + bottom padding + top border + bottom border + top margin + bottom margin.

```
<style>

/* Anchura total = 320px (width) + 20px (left + right padding) + 10px (left + right border) + 0px (left + right margin) = 350px */
div {
    width: 320px;
    padding: 10px;
    border: 5px solid gray;
    margin: 0;
}

</style>

<div>Hola mundo</div>
```

- Estas medidas se observan bien en las herramientas del desarrollador de Chrome.
- La etiqueta body generalmente tiene un margin por defecto establecido por el navegador.

Colapso del margen

- Cuando dos elementos se encuentran pegados, los márgenes se colapsan y la distancia entre los dos elementos corresponde al mayor borde.

```
<style>

#id1 {
    width: 320px;
    padding: 10px;

    /* forma simplificada de utilizar las propiedades border-width, border-style y border-
    color */
    border: 5px solid gray;
    margin: 20px;
}

#id2 {
    width: 320px;
    padding: 10px;
    border: 5px solid gray;
    margin: 5px;
}

</style>

<div id="id1">Hola mundo</div>
<div id="id2">Hola mundo</div>
```

Hola mundo

Hola mundo

Propiedades width/height

- Las propiedades width y height se usan para establecer el ancho y el alto de un elemento. Su valor puede ser establecido automáticamente por el navegador o puede especificarse en valores de longitud (px, cm, porcentajes, etc).

- Los valores por defecto son:
 - width: 100% (ocupa toda el ancho de su contenedor ancestro).
 - height: se adapta al contenido de la caja y el navegador calcula el valor automáticamente. Si no tiene contenido, entonces la caja no aparece.

```
<style>
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
</style>

<div>Texto</div>
```

- Las propiedades de width y height no incluyen relleno, bordes o márgenes, únicamente se refieren al contenido.
- Como a la anchura de un elemento hay que añadirle la anchura del margin, y del padding, existe una propiedad denominada box-sizing para mantener el tamaño del width como el ancho de todo el elemento. Posibles valores:
 - content-box: las propiedades width y height incluyen solo el contenido. Los bordes y el padding no están incluidos. Es el valor por defecto.
 - padding-box: las propiedades width y height incluyen el padding, pero no incluyen el border y el margin.
 - border-box: las propiedades width y height incluyen el padding y el border, pero no el margin.
 - Suele ser la opción más recomendada porque tiene sentido que el ancho representa a la suma del ancho del contenido, el padding y el borde. Se establece mediante el selector universal para que se aplique a todos los elementos.
 - margin-box: no existe.

```
<style>

div.ex1 {
  width: 300px;
  background-color: blue;
}

div.ex2 {
  width: 300px;
  padding: 25px;
```

```

/* Si se elimina esa propiedad, la anchura del elemento es 300+25+25=350px */
box-sizing: border-box;
background-color: red;
}

</style>

<div class="ex1">Texto</div>
<br>
<div class="ex2">Texto</div>

```

Ejercicio 1 del proyecto: escribir los estilos CSS para la cabecera de la página: el logo o nombre de la aplicación (Banco) dentro de un div y el menú en una lista desordenada dentro de un nav. Establece la propiedad box-sizing: border-box con el selector universal y margin: 0 para el elemento body

Propiedad display (block e inline)

- Es la propiedad de CSS más importante para controlar el diseño.
- Especifica si el elemento es mostrado y de qué forma.
- Cada elemento HTML tiene un valor de display predeterminado según el tipo de elemento que sea. El valor de visualización predeterminado para la mayoría de los elementos es block o inline, aunque algunos elementos HTML especiales tienen none, como por ejemplo, script.
- Un elemento block siempre comienza en una nueva línea y ocupa todo el ancho disponible (se extiende hacia la izquierda y hacia la derecha al máximo). Ejemplos: div, h1-h6, p, form, section, article, nav-
- Un elemento inline no comienza en una nueva línea y solo ocupa el ancho que necesite. Cualquier propiedad width, height, padding o margin no tendrá efecto alguno. Ejemplos: span, a, img.

```

<style>

span {
  border: 2px solid red;

  /* el atributo width no tiene efecto porque span es de tipo inline */
  width: 1420px;
}

</style>

<span>Texto 1</span>

```

- Sin embargo, es posible convertir un elemento block a elemento inline y viceversa.

```
<style>

div {
    border: 2px solid red;
    display: inline;
    width: 1120px
}

</style>

<div>Texto 1</div>
<div>Texto 2</div>
```

Propiedad display (inline-block)

- Los elementos que tienen la propiedad display establecida a inline-block no ocupan todo el ancho de la página por defecto y pueden situarse uno detrás de otro (como sucede con los elementos de tipo inline).
- Sin embargo, a diferencia de los elementos inline, en los elementos inline-block pueden establecerse el valor de las propiedades width, height, padding y margins.
- Los elementos inline-block tienen una pequeña separación si se encuentran en distintas líneas porque consideran el salto de línea entre elemento y elemento.

```
<style>

#div {
    border: 2px solid red;
    width: 300px;
    display: inline-block;
}

</style>

<!-- entre elemento span y elemento span hay una pequeña separación -->
<span id="div">Texto 1</span>
<span id="div">Texto 2</span>
```

Ejercicio 2 del proyecto: añade al header (utilizando la clase la main-header) las propiedades background-color: #2ddf5c; padding: 8px 16px; width:100% (width toma el valor 100% por defecto en elementos de tipo block como el header, por lo que no sería necesario establecer esta propiedad).

Ejercicio 3: añade la propiedad display:inline-block a todos los elementos li de la cabecera utilizando la clase main-nav__item para todos ellos. Añade también la clase main-nav__items a la etiqueta ul que engloba a todos los elementos li de la cabecera, la clase main-nav a la etiqueta nav y la clase main-header a la etiqueta header.

Ejercicio 4 del proyecto: muestra todos los elementos de la cabecera en una única línea, estableciendo el div del logo y el nav (clase main-nav) a display:inline-block. Para el div del logo utiliza el selector de combinación .main-header > div

Ejercicio 5 del proyecto: alinea los elementos del menú a la derecha estableciendo las propiedades width: 100% y text-align: right en la clase main-nav (los elementos del menú volverán a estar en una línea distinta al logo porque el menú ocupa ahora el 100% de su contenedor). Aplica también estilos a la clase main-nav__items: margin: 0; padding: 0; list-style: none;

Ejercicio 6 del proyecto: disminuye el valor de la propiedad width a 90% o mejor aún calc(100% - 49px) en el nav para eliminar la suma del pequeño espacio (unos pocos píxeles) entre elementos inline-block y el espacio ocupado por el div del logo (alrededor de 41px para la palabra Banco). Ahora el menú se establecerá a la derecha en la misma línea que el logo, aunque no es la mejor solución hacerlo de esta forma.

Propiedad display (none) y propiedad visibility (hidden)

- La propiedad display también permite esconder un determinado elemento mediante el valor none. Esto es útil para esconder determinada información de la página y solamente mostrarla cuando se produzca un evento (por ejemplo, un click en un botón).

```
<style>
h2 {
  display: none;
}

</style>

<h1>Título1</h1>
<h2>Título2</h2>
<h3>Título3</h3>
```

- La propiedad visibility también permite esta función. Sin embargo, en este caso el elemento ocupará el mismo espacio que antes. El elemento estará oculto, pero aún afectará el diseño.

```
<style>
h2 {
```

```

    visibility: hidden;
}

</style>

<h1>Título1</h1>
<h2>Título2</h2>
<h3>Título3</h3>

```

Propiedades CSS para texto

Propiedad text-align (texto)

- Establece la alineación de un texto. Sus posibles valores son: left, right, center, o justify.

```

<style>

h1 {
    text-align: center;
}

h2 {
    text-align: left;
}

h3 {
    text-align: right;
}

p {
    text-align: justify;
}

</style>

<h1>Cabecera 1 (center)</h1>
<h2>Cabecera 2 (left)</h2>
<h3>Cabecera 3 (right)</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin in aliquet elit. Duis nec odio nisi. Donec a venenatis lectus. Duis commodo tincidunt posuere. Suspendisse luctus non nulla eu pulvinar. Cras eleifend dignissim iaculis.</p>

```

Propiedad text-decoration

- Se usa para establecer o eliminar decoraciones del texto.
- No se recomienda subrayar el texto que no es un enlace, ya que a menudo confunde al usuario.

```
<style>

h1 {
    /* superrayado */
    text-decoration: overline;
}

h2 {

    /* tachado */
    text-decoration: line-through;
}

h3 {

    /* subrayado */
    text-decoration: underline;
}

</style>

<h1>Cabecera 1</h1>
<h2>Cabecera 2</h2>
<h3>Cabecera 3</h3>
```

Propiedad text-transform

- Transforma el texto en minúscula, mayúscula y capitalización (primera letra en mayúscula).

```
<style>

p.uppercase {

    /* Mayúscula */
    text-transform: uppercase;
}

p.lowercase {

    /* Minúscula */
    text-transform: lowercase;
}

p.capitalize {

    /* Capitalización */
    text-transform: capitalize;
}

</style>
```

```
<p class="uppercase">Texto</p>
<p class="lowercase">Texto</p>
<p class="capitalize">Texto</p>
```

Propiedad text-indent

- Permite identar los párrafos.

```
<style>
```

```
p {  
    text-indent: 50px;  
}
```

```
</style>
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin in aliquet elit. Duis nec odio nisi. Donec a venenatis lectus. Duis commodo tincidunt posuere. Suspendisse luctus non nulla eu pulvinar. Cras eleifend dignissim iaculis.</p>
```

Propiedad letter-spacing

- Permite definir un valor concreto para el espacio entre letras:

```
<style>
```

```
h1 {  
    letter-spacing: 3px;  
}
```

```
h2 {  
    letter-spacing: -3px;  
}
```

```
</style>
```

```
<h1>Cabecera 1</h1>
<h2>Cabecera 2</h2>
```

Propiedad line-height

- Permite definir espacio entre líneas.

```
<style>

p.small {
    line-height: 0.7;
}

p.big {
    line-height: 1.8;
}

</style>

<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
</p>

<p class="small">
Lorem ipsum dolor sit amet, consectetur adipiscing elit.<br>
Lorem ipsum dolor sit amet, consectetur adipiscing elit.<br>
</p>

<p class="big">
Lorem ipsum dolor sit amet, consectetur adipiscing elit.<br>
Lorem ipsum dolor sit amet, consectetur adipiscing elit.<br>
</p>
```

Propiedad text-direction

- ◆ Funciona de forma similar a text-align.
- ◆ Posibles valores:
 - ltr: alineado a la izquierda (predeterminado).
 - rtl: alineado a la derecha.
 - inherit: heredado del elemento padre.

```
<style>

p.rtl {
    direction: rtl;
}

</style>

<p>Texto a la izquierda</p>
<p class="rtl">Texto a la derecha</p>
```

Propiedad word-spacing

- Permite definir el espacio entre palabras.

```
<style>

h1 {
    word-spacing: 10px;
}

h2 {
    word-spacing: -5px;
}

</style>

<h1>Texto Texto Texto Texto Texto </h1>
<h2>Texto Texto Texto Texto Texto </h2>
```

Propiedad text-shadow

- Añade sombra al texto:
 - El primer parámetro es la distancia de la sombra en el eje x.
 - El segundo parámetro es la distancia de la sombra en el eje y.
 - El tercer parámetro es el color.

```
<style>

h1 {
    text-shadow: 3px 2px red;
}

</style>

<h1>Texto con sombra</h1>
```

Propiedades CSS para fuentes

Propiedad font-family

- Permite definir el tipo de fuente.
- Se pueden especificar varias fuentes.
- Si el navegador no es compatible con la primera fuente, intenta con la siguiente fuente, y así sucesivamente.

- Si el nombre de una familia de fuentes es más de una palabra, entonces deben utilizarse comillas simples obligatoriamente. Ejemplo: "Times New Roman".
- Para especificar varias tipografías se utiliza la coma como separador.

```
<style>

p.serif {
    font-family: "Times New Roman", Times, serif;
}

p.sansserif {
    font-family: Arial, Helvetica, sans-serif;
}

</style>

<p class="serif">Times New Roman font</p>
<p class="sansserif">Arial font</p>
```

Propiedad font-style

- Especifica el estilo de un texto: itálica u oblicua.

```
<style>

p.normal {
    font-style: normal;
}

p.italic {
    font-style: italic;
}

p.oblique {
    font-style: oblique;
}

</style>

<p class="normal">normal</p>
<p class="italic">italic</p>
<p class="oblique">oblique</p>
```

Propiedad font-size

- Especifica tamaño de la letra.

- Si no especifica un tamaño de fuente, entonces el tamaño predeterminado para el texto normal (como los párrafos) es de 16 px.
- El valor del tamaño de fuente puede ser un tamaño absoluto o relativo.
- El tamaño absoluto es útil cuando se conoce el tamaño físico del dispositivo.

```
<style>

h1 {
    font-size: 40px;
}

h2 {
    font-size: 30px;
}

p {
    font-size: 2px;
}

</style>

<h1>Cabecera 1</h1>
<h2>Cabecera 2</h2>
<p>Párrafo 1</p>
<p>Párrafo 2</p>
```

- Muchos desarrolladores usan la unidad de medida em en lugar de píxeles. La unidad de tamaño em es recomendada por la W3C.
- 1em es igual al tamaño de fuente actual. El tamaño de texto predeterminado en los navegadores es de 16px. Entonces, el tamaño predeterminado de 1em es 16px.

```
<style>

/* 16 * 2.5em = 40px */
h1 {
    font-size: 2.5em;
}

/* 1.875em *16 = 30px*/
h2 {
    font-size: 1.875em;
}

/* 0.875em * 16 = 14px */
p {
```

```

        font-size: 0.875em;
    }

</style>

<h1>Cabecera 1</h1>
<h2>Cabecera 2</h2>
<p>Párrafo 1</p>
<p>Párrafo 2</p>
```

Propiedad font-weight

- Especifica el peso o intensidad (negrita) de la letra.

```

<style>

p.normal {
    font-weight: normal;
}

p.light {
    font-weight: lighter;
}

p.thick {
    font-weight: bold;
}

p.thicker {
    /* El máximo valor es 900 */
    font-weight: 900;
}

</style>

<p class="normal">Texto</p>
<p class="light">Texto.</p>
<p class="thick">Texto</p>
<p class="thicker">Texto</p>
```

Propiedad font-variant

- Especifica si un texto se debe mostrar o no en una fuente de mayúsculas pequeñas.
- Todas las letras minúsculas se convierten en letras mayúsculas. Sin embargo, estas letras mayúsculas convertidas aparecen en un tamaño de letra más pequeño que las letras mayúsculas originales en el texto.

```
<style>

p.normal {
    font-variant: normal;
}

p.small {
    font-variant: small-caps;
}

</style>

<p class="normal">Texto</p>
<p class="small">Texto</p>
```

Ejercicio 7 del proyecto: añade la clase .main-header__brand en la etiqueta a que se encuentra dentro del div del logo. Seguidamente agregar las siguientes propiedades: color: #0e4f1f; text-decoration: none; font-weight: bold; font-size: 22px; (será necesario modificar la función calc para mantener el logo y el menú en la misma línea).

Google Fonts

- El tipo de fuente que se mostrará en el navegador de un cliente dependerá de las fuentes instaladas en su sistema.
- Se puede forzar a que un usuario descargue una fuente específica utilizando Google Fonts.
- Para ello es necesario agregar un archivo CSS proporcionado por Google Fonts y establecer el nuevo tipo de fuente utilizando el selector body.

Ejercicio 8 del proyecto: agrega al CSS la fuente Montserrat de Google Fonts utilizando el selector body (será necesario modificar la función calc para mantener el logo y el menú en la misma línea). La alineación vertical de los elementos del menú no será del todo correcta porque no se encontrará en el medio (disminuir el tamaño de la letra para comprobar la posición).

Propiedad vertical-align

- Especifica el alineado vertical de un elemento inline, inline-block o una celda de una tabla. Sus valores más importantes son: top, bottom o middle.

```
<style>

.middle > * {
    vertical-align: middle;
}
```

```
.top > * {  
    vertical-align: top;  
}  
  
.bottom > * {  
    vertical-align: bottom;  
}  
  
h2 {  
    display: inline-block;  
    margin: 0;  
    width: 150px;  
    text-align: right;  
    font: bold 20px Sans-Serif;  
}  
  
div {  
    margin: 0 0 30px 0;  
    padding: 10px;  
}  
  
</style>  
  
<div class="middle">  
    <h2>Middle</h2>  
      
      
      
</div>  
  
<div class="top">  
    <h2>Top</h2>  
      
      
      
</div>  
  
<div class="bottom">  
    <h2>Bottom</h2>  
      
      
      
</div>
```

- El valor por defecto de vertical-align es baseline. Con este valor el elemento se alinea en la base del elemento contenedor.

Ejercicio 9 del proyecto: establece el valor de vertical-align:middle para los elementos del menú (clase main-nav) y el logo (selector main-header > div). Observa cómo se desplaza

verticalmente los elementos del menú con los diferentes valores de vertical-align (se aprecia mejor modificando el tamaño de la letra).

Ejercicio 10 del proyecto: realiza los siguientes cambios en los enlaces del menú a través de la clase main-nav__item: elimina el subrayado al pulsar los enlaces (no funcionará porque la propiedad text-decoration se heredará al elemento a y estos elementos ya tienen un valor por defecto para la propiedad text-decoration y es más específico), añade un margin lateral entre los elementos, aumenta la intensidad de la fuente, un color (#0e4f1f) y cambia su tamaño.

Tu carrera digital ~

Módulo 2

Diseño de páginas interactivas frontend (CSS)

CSS Flexbox

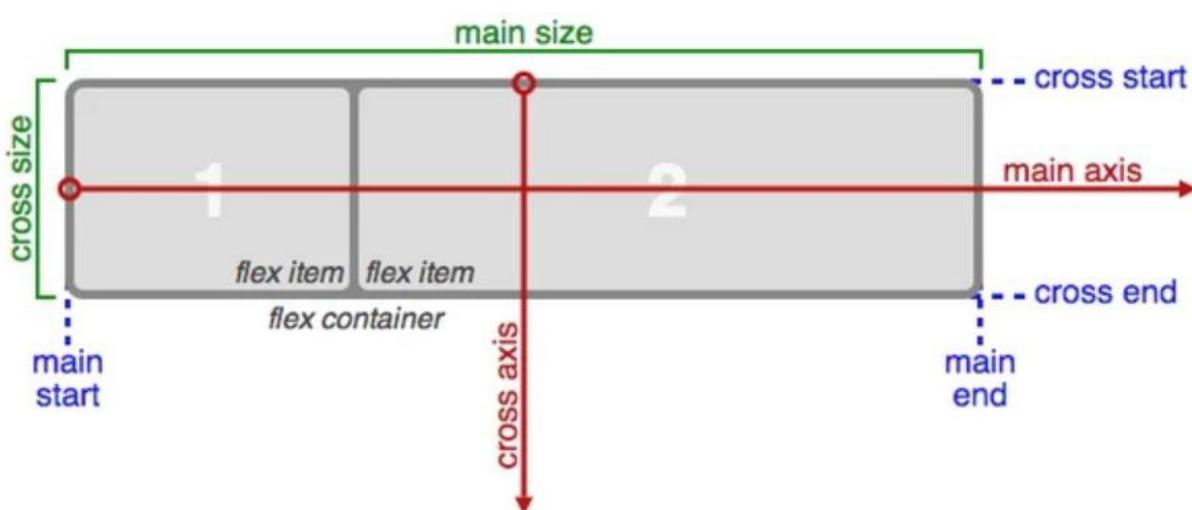


Flexbox

- ◆ Introducción
- ◆ Propiedades del contenedor ancestro (flex-container)
 - Propiedad flex-direction
 - Propiedad flex-wrap
 - Propiedad flex-flow
 - Propiedad justify-content
 - Propiedad align-items
 - Propiedad align-content
- ◆ Propiedades de los elementos descendientes (flex-items)
 - Propiedad order
 - Propiedad flex-grow
 - Propiedad flex-shrink
 - Propiedad flex-basis
 - Propiedad flex
 - Propiedad align-self

Introducción

- ◆ Flexbox facilita el diseño de una estructura flexible sin tener que utilizar float o position.
- ◆ Los componentes de Flexbox se pueden anidar dentro de otros componentes de flexbox.
- ◆ El elemento flex es un contenedor con dos ejes, los ejes principal y transversal, y los marcadores que especifican el inicio y el final del elemento en ambos ejes.
- ◆ [Más info.](#)



- ◆ El primer paso en flexbox es crear un elemento con un atributo display:flex.

```

<style>
#flex-container {
  display: flex;
}

</style>

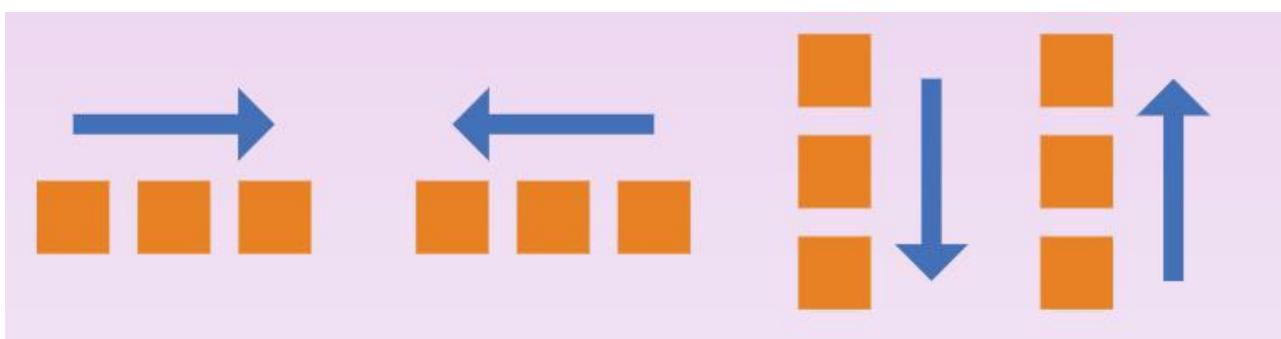
<div id="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

```

Propiedades del contenedor ancestro (flex-container)

Propiedad flex-direction

- Establece la dirección del eje principal en que se colocan los elementos flexibles en el contenedor. Flexbox es un layout de una única dirección.
- Posibles valores:
 - row (por defecto): de izquierda a derecha
 - row-reverse: de derecha a izquierda.
 - column: de arriba a abajo.
 - column-reverse: de abajo a arriba.
- En los elementos que contiene el contenedor se utiliza line-height en lugar de height para cambiar la altura de los elementos.
- Ejemplo



```

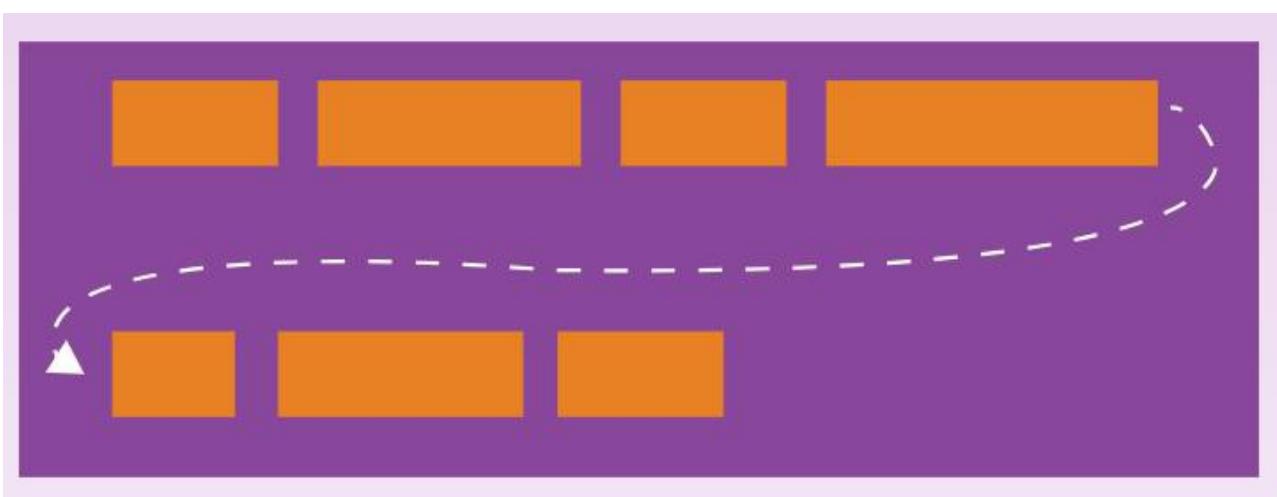
<style>
.flex-container {
  display: flex;
  flex-direction: column;
}

```

```
background-color: red;  
/* align-items: center; */  
}  
  
.flex-container > div {  
background-color: blue;  
width: 50%;  
margin: 10px;  
text-align: center;  
line-height: 75px;  
font-size: 30px;  
}  
  
</style>  
  
<div class="flex-container">  
<div>1</div>  
<div>2</div>  
<div>3</div>  
</div>
```

Propiedad flex-wrap

- Por defecto, todos los items tratarán de estar en una única línea. Pero esto puede cambiarse mediante la propiedad flex-wrap.
- Posibles valores:
 - nowrap (por defecto): todos los elementos estarán en una única línea.
 - wrap: los elementos estarán en varias líneas, de arriba a abajo.
 - wrap-reverse: los elementos estarán en varias líneas, de abajo a arriba.
- Ejemplo



```

<style>
.flex-container {
  display: flex;
  flex-wrap: wrap-reverse;
  background-color: blue;
}

.flex-container > div {
  background-color: red;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
  <div>11</div>
  <div>12</div>
</div>

```

Propiedad flex-flow

- Permite combinar las propiedades flex-direction y flex-wrap.

```

.container {
  flex-flow: row wrap;
}

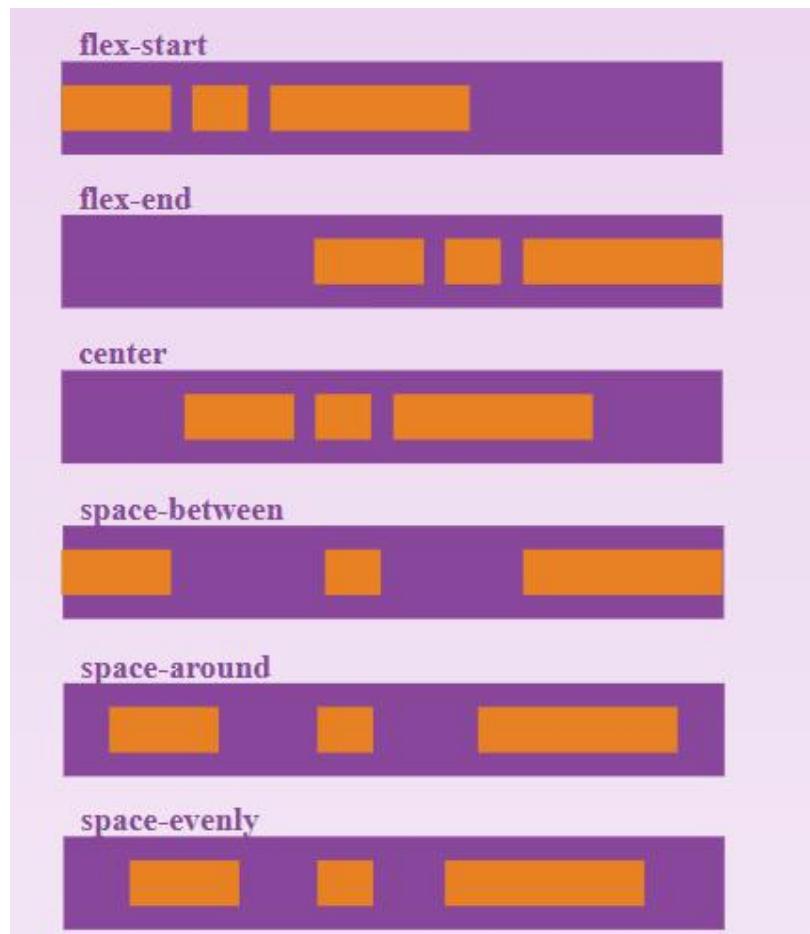
```

Propiedad justify-content

- Esto define la alineación de los elementos a lo largo del eje principal.
- Posibles valores:
 - flex-start (por defecto): los elementos son colocados al comienzo de la línea.

- flex-end: los elementos son colocados al final de la línea.
- center: los elementos son centrados.
- space-between: los elementos son distribuidos en la línea con el mismo espacio de separación, excepto el primer elemento y el último elemento, que se pegan a sus bordes.
- space-around: los elementos son distribuidos en la línea con un espacio semejante entre ellos, pero sin solaparse (por lo que el primer elemento y el último tienen menor espacio a la izquierda y a la derecha respectivamente).
- space-evenly: los elementos son distribuidos de tal forma que el espacio entre dos elementos es el mismo (incluido los espacios de los elementos de las esquinas).

♦ Ejemplo



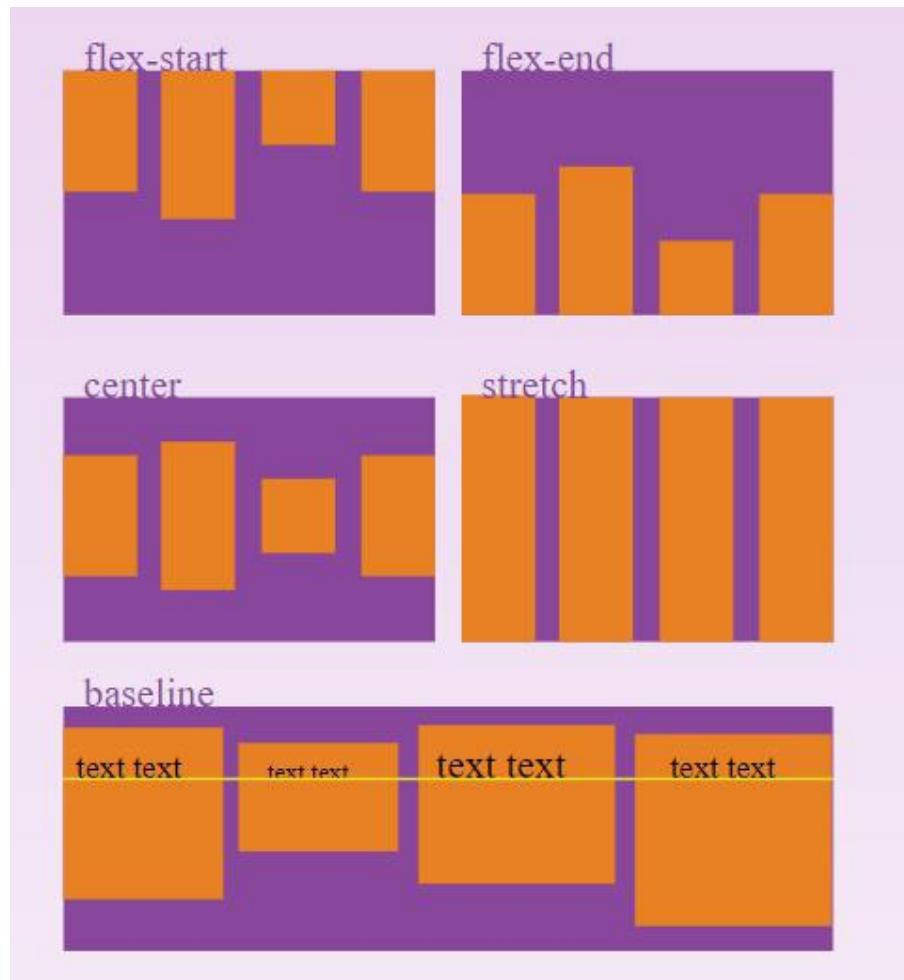
```
<style>
.flex-container {
  display: flex;
  justify-content: flex-end;
  background-color: blue;
}

.flex-container > div {
  background-color: red;
```

```
width: 100px;  
margin: 10px;  
text-align: center;  
line-height: 75px;  
font-size: 30px;  
}  
</style>  
  
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
</div>
```

Propiedad align-items

- Esta propiedad alinea los elementos cuando hay espacio adicional en el eje transversal, de forma similar a cómo justify-content alinea elementos individuales dentro del eje principal.
- [Ejemplos](#)
- Posibles valores:
 - flex-start: los elementos son colocados en la parte superior del contenedor.
 - flex-end: los elementos son colocados en la parte inferior del contenedor.
 - center: los elementos son colocados en la parte central del contenedor.
 - baseline: los elementos son colocados en proporción a sus bases (texto).
 - stretch (por defecto): los elementos ocupan todo el espacio del contenedor.



```
<style>
.flex-container {
  display: flex;
  height: 200px;
  align-items: center;
  background-color: blue;
}

.flex-container > div {
  background-color: red;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}

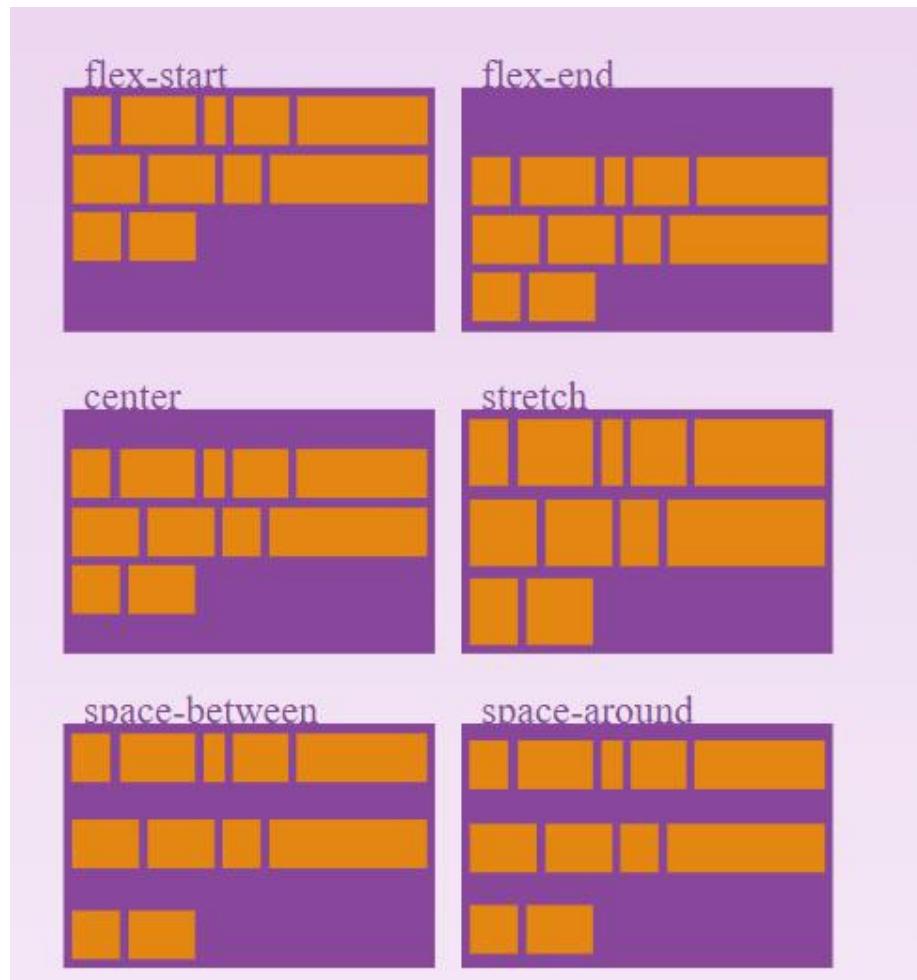
</style>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

</div>

Propiedad align-content

- ◆ Esta propiedad alinea las líneas de un contenedor cuando hay espacio adicional en el eje transversal, de forma similar a cómo justify-content alinea elementos individuales dentro del eje principal.
- ◆ Posibles valores:
 - ◊ flex-start: las líneas se ubican en la parte superior del contenedor.
 - ◊ flex-end: las líneas se ubican en la parte inferior del contenedor.
 - ◊ center: las líneas se ubican en la parte central del contenedor.
 - ◊ space-between: las líneas son distribuidas en el contenedor con el mismo espacio de separación, excepto la primera línea y la última línea, que se pegan a sus bordes).
 - ◊ space-around: las líneas son distribuidas en el contenedor con un espacio semejante entre ellos, pero sin solaparse (por lo que el primer elemento y el último tienen menor espacio a la izquierda y a la derecha respectivamente).
 - ◊ space-evenly:
 - ◊ stretch (por defecto): las líneas ocupan todo el espacio del contenedor.
- ◆ [Ejemplo](#)



```
<style>
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: space-around;
  background-color: blue;
}

.flex-container > div {
  background-color: red;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}

</style>

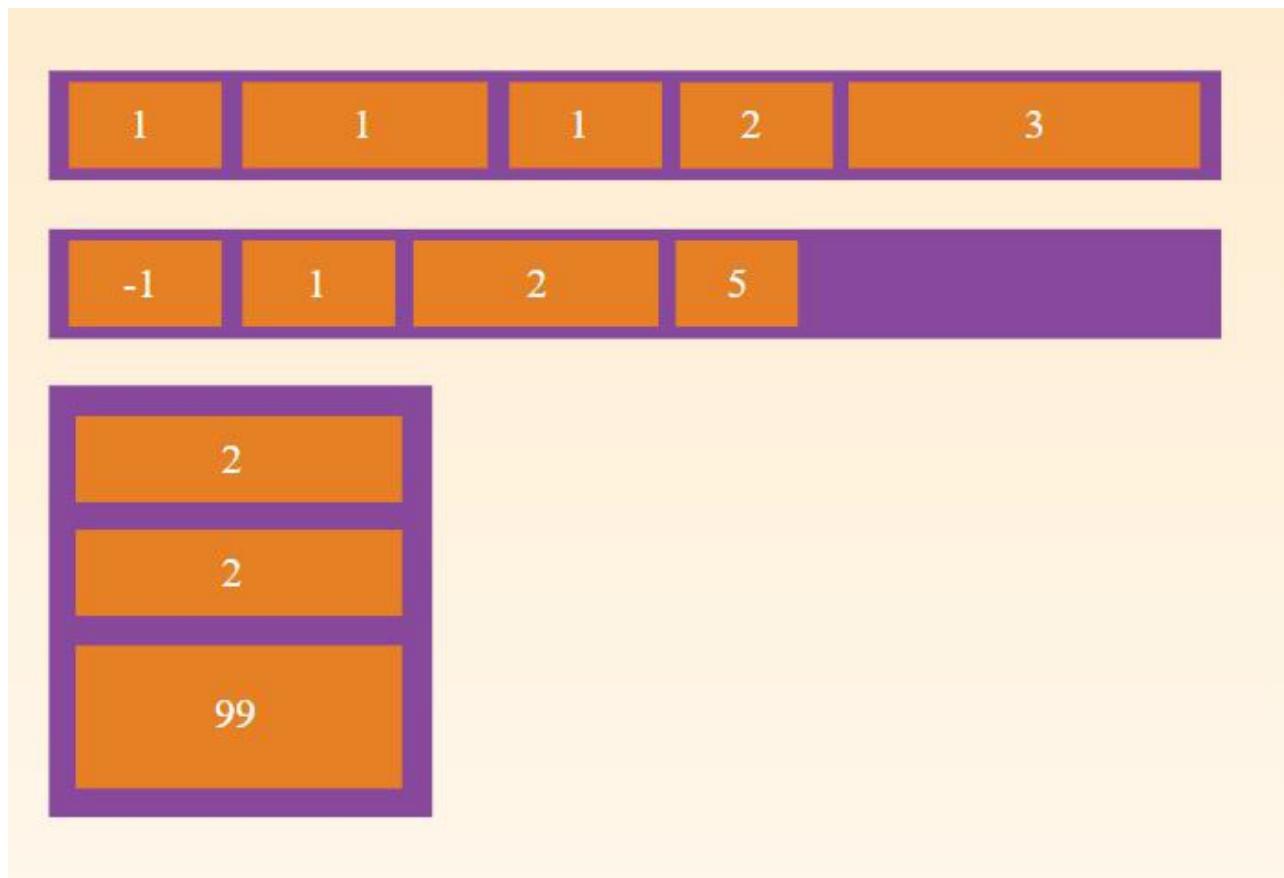
<div class="flex-container">
  <div>1</div>
  <div>2</div>
</div>
```

```
<div>3</div>
<div>4</div>
<div>5</div>
<div>6</div>
<div>7</div>
<div>8</div>
<div>9</div>
<div>10</div>
<div>11</div>
<div>12</div>
</div>
```

Propiedades de los elementos descendientes (flex-items)

Propiedad order

- Por defecto, los elementos flexibles se presentan en orden. Sin embargo, esta propiedad controla el orden en que aparecen en el contenedor. El valor por defecto es 0.



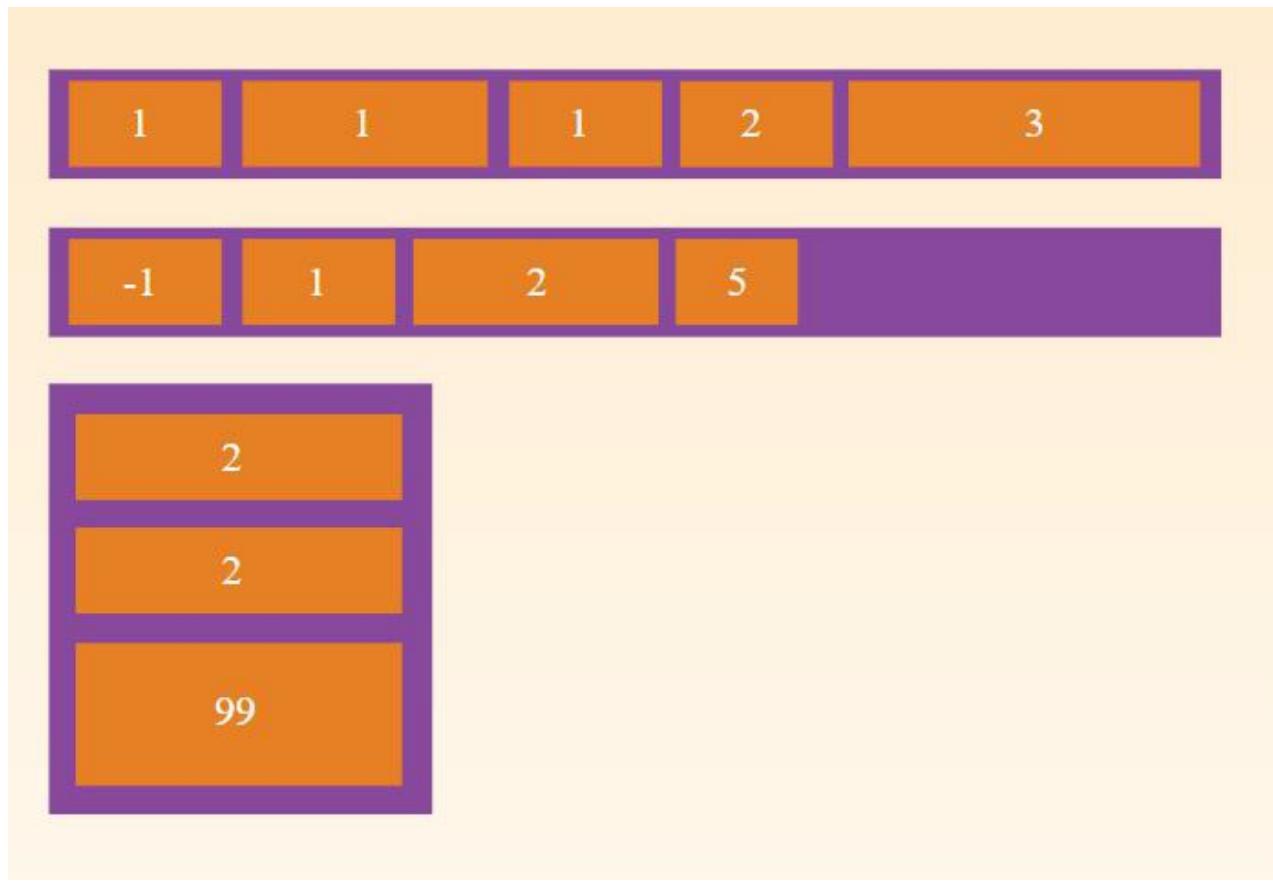
```
<style>
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: blue;
}
```

{}

```
.flex-container > div {  
    background-color: red;  
    color: white;  
    width: 100px;  
    margin: 10px;  
    text-align: center;  
    line-height: 75px;  
    font-size: 30px;  
}  
</style>  
  
<div class="flex-container">  
    <div style="order: 3">1</div>  
    <div style="order: 2">2</div>  
    <div style="order: 4">3</div>  
    <div style="order: 1">4</div>  
</div>
```

Propiedad flex-grow

- Define la capacidad de un elemento para crecer si es necesario. Acepta un valor sin unidades que sirve como una proporción. Indica qué cantidad de espacio disponible dentro del contenedor flexible debe ocupar el elemento. Por defecto, el valor es 0.
- Si todos los elementos tienen un crecimiento establecido en 1, el espacio restante en el contenedor se distribuirá por igual a todos los elementos. Si uno de los elementos tiene un valor de 2, el espacio restante ocuparía el doble de espacio que los demás (o al menos, lo intentará).



```
<style>
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}

.flex-container > div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}

</style>

<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 6">2</div>
  <div style="flex-grow: 1">4</div>
</div>
```

Propiedad flex-shrink

- Define la capacidad de un elemento para contraerse si es necesario (cuando el número de elementos en una línea es excesivo). Por defecto el valor es 1. Con 0 se estira el elemento y con 2 o superior, se contrae.

```
<style>

.flex-container {
  display: flex;
  align-items: stretch;
  background-color: #f1f1f1;
}

.flex-container>div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>

<div class="flex-container">
  <div style="flex-shrink: 2">1</div>
  <div style="flex-shrink: 1">2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
</div>
```

Propiedad flex-basis

- Define el tamaño predeterminado de un elemento. Puede establecerse una longitud (por ejemplo, 20%, 5%, 200px, etc.) o la palabra auto, que comprueba las propiedades width y height.

```
<style>
.flex-container {
  display: flex;
  align-items: stretch;
  background-color: red;
}

.flex-container > div {
  background-color: blue;
```

```

color: white;
width: 100px;
margin: 10px;
text-align: center;
line-height: 75px;
font-size: 30px;
}

</style>

<div class="flex-container">
<div>1</div>
<div>2</div>
<div style="flex-basis:500px">3</div>
<div>4</div>
<div>5</div>
<div>6</div>
<div>7</div>
</div>

```

Propiedad flex

- Esta es la abreviatura de flex-grow, flex-shrink y flex-basis combinados.

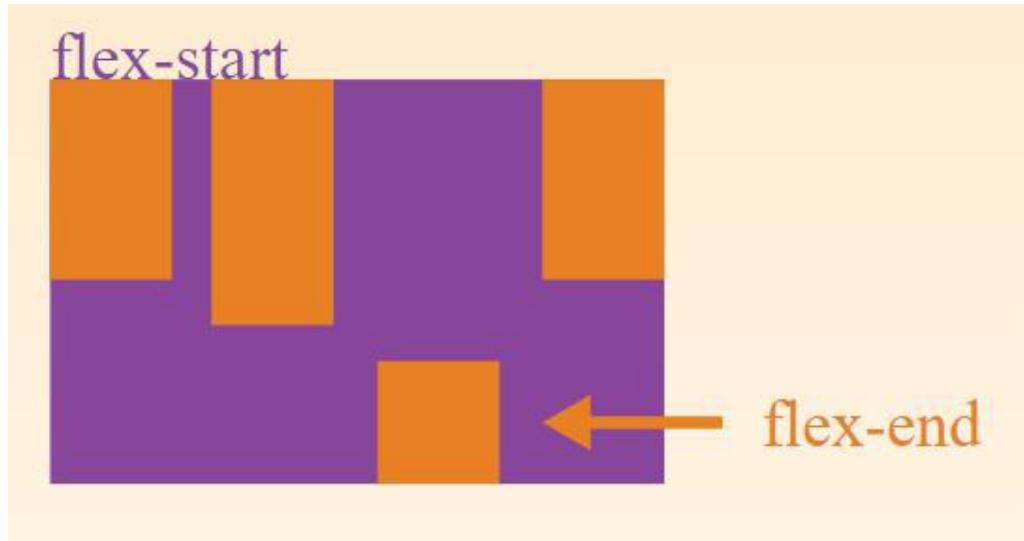
```

.item {
  flex: 0 0 200px;
}

```

Propiedad align-self

- Permite que la alineación predeterminada del contenedor (stretch) se anule para elementos individuales.
- Posibles valores:
 - flex-start: el elemento es colocado en la parte superior del contenedor.
 - flex-end: el elemento es colocado en la parte inferior del contenedor.
 - center: el elemento es colocado en la parte central del contenedor.
 - baseline: el elemento es colocado en proporción a sus bases (texto).
 - stretch (por defecto): el elemento es colocado ocupando todo el espacio del contenedor.



```
<style>
.flex-container {
  display: flex;
  height: 200px;
  background-color: red;
}

.flex-container > div {
  background-color: blue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}

</style>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="align-self: center">3</div>
  <div>4</div>
</div>
```

Tu carrera digital ~

Módulo 2

Diseño de páginas interactivas frontend (JavaScript)

Fundamentos

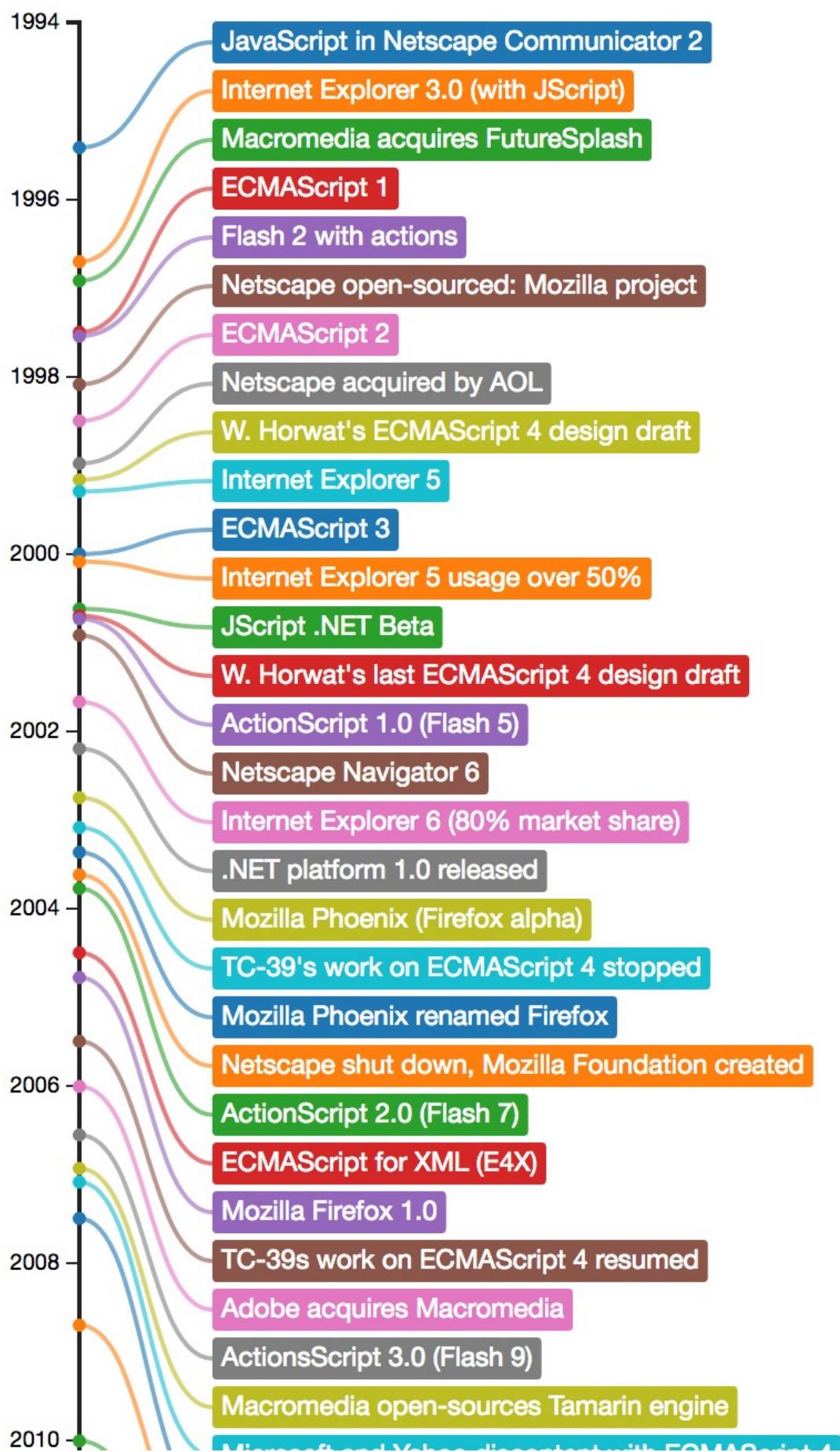


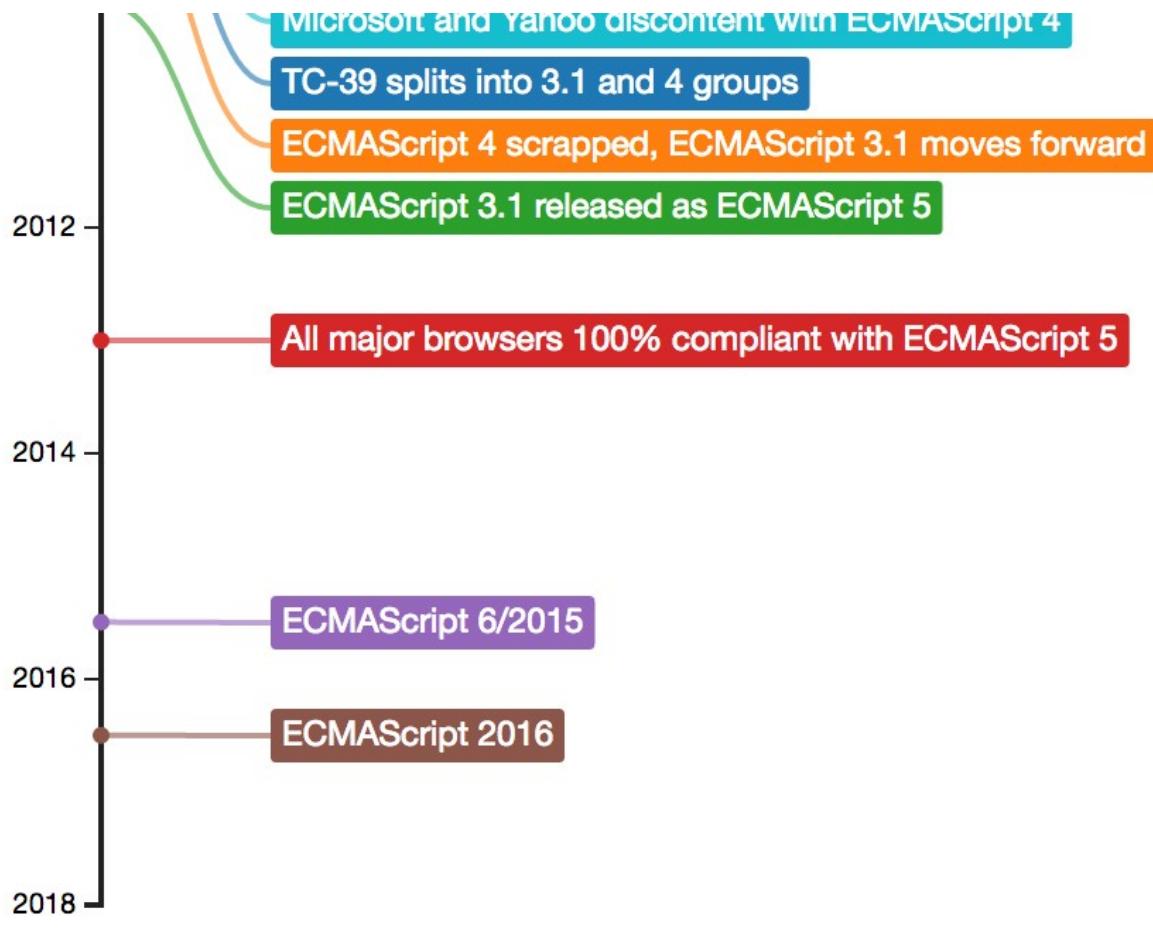
Fundamentos

- ◆ 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).

| JavaScript (frontend) | JavaScript (backend) |
|--|--|
| 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.

```
// 10000000000000000000000000000000  
console.log(99999999999999999999);  
  
// 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 sobreesciben 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 las 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 las 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 sobrescribe.
- 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 (JavaScript)

Objetos



Objetos

- ◆ Introducción
- ◆ Creación de objetos
- ◆ Operaciones con las propiedades de un objeto

Introducción

- ◆ Un objeto es una colección no ordenada de variables (atributos) y funciones (métodos).
- ◆ Al conjunto de atributos y métodos de un objeto se les llama propiedades.
- ◆ Todo en JavaScript es un objeto, con excepción de null y undefined.
- ◆ JavaScript realiza una conversión temporal a objeto cuando se utiliza algún método o atributo en variables no reconocidas como object por typeof.

```
// boolean
console.log(typeof false);

// false
console.log(false.toString());
```

Creación de objetos

- ◆ Existen cuatro formas de crear objetos.
- ◆ Primera forma (recomendada).

```
// declaración de un objeto con dos propiedades: un atributo y un método
const objeto = {
  name: 'Rebecca',
  diHola: () => {
    console.log('holo');
  }
};
objeto.diHola();
```

- ◆ Segunda forma.

```
// declaración de un objeto vacío (posteriormente son agregados un atributo y un
método)
const coche = {};
coche.contenido = 20;
```

```
coche.adelante = () => {
    console.log('Arrancando...');

};

coche.adelante();
```

- Tercera forma: con la partícula new sobre una función anónima (es obligatorio utilizar function en lugar de una función de flecha).

```
const Persona = function(name) {
    this.name = name;
};

const yo = new Persona('alejandro');

// alejandro
console.log(yo.name);
```

- Cuarta forma: con closures (se verá posteriormente).

Operaciones con las propiedades de un objeto

- El acceso a las propiedades un objeto se realiza mediante la notación punto y también con corchetes.

```
const persona = {
    nombre: 'Alejandro',
    'lugar de residencia': 'Madrid'
};

// Alejandro
console.log(persona.nombre);

// Alejandro
console.log(persona['nombre']);

// error
console.log(persona.lugar de residencia);

// en este caso son obligatorios los corchetes
persona['lugar de residencia'] = 'Sevilla';

// Sevilla
console.log(persona['lugar de residencia']);

const propiedad = 'nombre';

// undefined porque busca la variable propiedad dentro del objeto persona
console.log(persona.propiedad);
```

// Alejandro
`console.log(persona[propiedad]);`

- ◆ Anidación de elementos.

```
const persona = {
  nombre: 'Carmen',
  apellidos: 'González',
  datos: {
    dirección: 'Sevilla'
  }
};

// Sevilla
console.log(persona.datos.dirección);

// undefined
console.log(persona.datos.otrodato);
```

- ◆ En el espacio de memoria de un objeto no se almacena el propio objeto (sus propiedades), sino una dirección de memoria (referencia) que apunta al objeto. Debido a esta razón, no hay una forma simple de comparar un objeto con otro.

```
const coche1 = { marca: 'Ford', modelo: 'Focus' };
const coche2 = { marca: 'Ford', modelo: 'Focus' };

// devuelve false porque no comparten referencia
console.log(coche1 == coche2);

// devuelve false porque no comparten referencia
console.log(coche1 === coche2);

// devuelve true porque el valor es el mismo y son de tipo string
console.log(coche1.modelo === coche2.modelo);

const coche3 = coche1;

// devuelve true porque comparten referencia
console.log(coche1 === coche3);
```

- ◆ El borrado de propiedades en objetos se realiza con `delete`.

```
const x = {
  peras: 3,
```

```
fresas: 20
};

delete x.fresas;
```

- Para comprobar la existencia de alguna propiedad en un objeto puede utilizarse Object.keys y, a continuación el método includes de los arrays.
 - Object.keys(objeto) devuelve un array con todos los nombres de las propiedades de un objeto.
 - includes es un método de los arrays para comprobar si un determinado elemento existe o no dentro de un array

```
const x = {

  peras: 3,
  fresas: 20,

  // no se debe utilizar aquí la sintaxis de la función de flecha porque this en ese caso
  // apuntaría al objeto global
  contar: function () {
    console.log(this.peras + this.fresas);
  }
};

// se guarda en un array el nombre de todas las propiedades del objeto x
const nombresPropiedades = Object.keys(x);

// ["peras", "fresas", "contar"]
console.log(nombresPropiedades);

// true
console.log(nombresPropiedades.includes('fresas'));
```

- Otra forma alternativa de comprobar la existencia de una propiedad en un objeto es utilizando in.

```
const x = {

  peras: 3,
  fresas: 20,

  contar: function() {

    // no se puede utilizar aquí función de flecha porque this apuntaría al objeto global
    console.log(this.peras + this.fresas);
  }
};
```

```
// true
console.log('fresas' in x);
```

- Al contrario que en otros lenguajes de programación, en JavaScript los objetos pueden ser extendidos de forma dinámica.

```
// creación de un objeto vacío
const alejandro = {};

// agregando un método nuevo
alejandro.saludar = () => {
    console.log('Hola');
};
```

- Para iterar las propiedades de un objeto se utiliza el bucle for...in

```
const x = {
    peras: 3,
    fresas: 20
};

for (let key in x) {

    // peras y fresas (en distintas líneas)
    console.log(key);

    // string y string (en distintas líneas)
    console.log(typeof(key));

    // 3 y 20 (en distintas líneas)
    console.log(x[key]);

    // undefined y undefined porque busca la propiedad x.key (2 veces) y no existe
    console.log(x.key);
}
```

- Otra forma de iterar las propiedades de un objeto es mediante Object.keys
 - Sin embargo, Object.keys retorna un array y para recorrer arrays es mejor utilizar for...of
 - El bucle for...in en los array asigna la posición a la variable iteradora (0, 1,... hasta el final de array).

```
const x = {
    peras: 3,
    fresas: 20
```

```
};

// claves es un array
const claves = Object.keys(x);

// ["peras", "fresas"]
console.log(claves);

// bucle for...of para recorrer un array
for (let key of claves) {

    // peras y fresas (en distintas líneas)
    console.log(key);

    // 3 y 20 (en distintas líneas)
    console.log(x[key]);
}

// bucle for...in para recorrer un array
for (let i in claves) {

    // 0 y 1 (en distintas líneas)
    console.log(i);

    // peras y fresas (en distintas líneas)
    console.log(claves[i]);

    // 3 y 20 (en distintas líneas)
    console.log(x[claves[i]]);
}
```

- En JavaScript no hay una forma simple de crear atributos o métodos privados. Puede simularse este comportamiento con una función creadora de objetos, donde las propiedades declaradas con this son públicas y el resto declaradas con let o const son privadas.

```
// obligatorio utilizar function para posteriormente crear objetos con new
const Person = function(name) {

    // atributo público porque está declarado con this
    this.name = name || "";

    // atributo privado porque está declarado con const
    const lastname = 'González';

    // atributo privado porque está declarado con const
    const myName = this.name;

    // método privado porque está declarado con const
```

```
const fullName = () => {
    return myName + ' ' + lastname;
};

// método público porque está declarado con this
this.introduce = () => {
    return 'Hola, mi nombre es ' + fullName();
};
};

const oscar = new Person('Óscar');

// Óscar
console.log(oscar.name);

// Hola, mi nombre es Óscar González
console.log(oscar.introduce());

// undefined porque el atributo es privado
console.log(oscar.lastname);

// error porque el método no privado y no se puede acceder a él (es undefined) e invocar
// a undefined genera un error
console.log(oscar.fullName());
```

- En definitiva, existen tres diferentes formas de declarar funciones dentro de objetos (métodos).

```
const objeto1 = {
    nombre: 'Alejandro',

    // función clásica de JavaScript sin function (opción recomendada)
    mostrarNombre() {
        console.log(this.nombre);
    }
}

// Alejandro
objeto1.mostrarNombre();
```

```
const objeto2 = {
    nombre: 'Alejandro',

    // función clásica de JavaScript con function
    mostrarNombre: function() {
        console.log(this.nombre);
    }
}
```

```
// Alejandro
objeto2.mostrarNombre();  
  
const objeto3 = {
  nombre: 'Alejandro',  
  
  // función cde flecha
  mostrarNombre: () => {
    console.log(this.nombre);
  }
}  
  
// undefined
objeto3.mostrarNombre();
```

Ejercicio 1: escribe un programa que liste el nombre de las propiedades del siguiente objeto utilizando las dos formas comentadas con anterioridad: { name : "David Rayy", sclass : "VI", rollno : 12 }

Ejercicio 2: escribe un programa que declare el siguiente objeto: { x: 1, y: 2 }. A continuación, elimina la propiedad x y añade una propiedad nueva z con valor 5.

Ejercicio 3: escribe un programa que declare dos objetos: { x: 1, y: 2, z: 3 } y { a: 1, y: 2, z: 1 }. A continuación, comprueba qué nombres de propiedades son coincidentes y cuáles son también coincidentes en valor.

Ejercicio 1 del proyecto: escribe un programa que declare 3 objetos de cada modelo de datos considerado: gestor, cliente, mensaje y transferencia. Los valores de las propiedades de los objetos pueden ser arbitrarios.

Tu carrera digital ~

Módulo 2

Diseño de páginas interactivas frontend (JavaScript)

String



String

- ◆ Introducción
- ◆ Atributos y métodos

Introducción

- ◆ Posibles forma de definir un string.

```
// crea un objeto String
const a = new String('hola');

// crea una string con comillas dobles
const b = 'hola';

// crea una string con comillas simples
const c = 'hola';
```

- ◆ JavaScript convierte automáticamente de string a objeto temporal cuando se utiliza alguna propiedad de los string.

```
// más rápido
const x = 'John';

// más lento
const y = new String('John');

// string
console.log(typeof(x));

// object
console.log(typeof(y));

// true porque se comparan los valores (son iguales) y no los tipos (no son iguales)
console.log(x == y);

// false
console.log(x === y);

const z = new String('John');

// false porque es una comparación entre objeto y objeto (referencia)
console.log(y == z);

// false
console.log(y === z);
```

- Para escapar caracteres en JavaScript se utiliza la barra invertida o backslash.

```
// "hola"
const a = '\\"hola\\"';

// 'hola'
const b = '\'hola\'';

// \hola\
const c = '\\\\hola\\\\';
```

- Otros caracteres especiales de escapado:
 - \n: nueva linea.
 - \t: tabulador horizontal.
 - \v: tabulador vertical.

Atributos y métodos

- Atributo length: obtiene la longitud de un string.

```
const str = 'En un lugar de la mancha';

// 24
console.log(str.length);
```

- Método indexOf: devuelve la posición de la primera aparición de un string o -1 si no hay coincidencia.

```
const str = 'En un lugar de la mancha';

// 6
console.log(str.indexOf('l'));

// 6 (comienza a buscar a partir de la posición 6)
console.log(str.indexOf('l', 6));

// 18
console.log(str.indexOf('mancha'));

// -1 (comienza a buscar a partir de la posición 19)
console.log(str.indexOf('mancha', 19));

// -1
console.log(str.indexOf('noencontrado'));
```

- Método `lastIndexOf`: devuelve la posición de la última aparición de un string o -1 si no hay coincidencia.

```
const str = 'En un lugar de la mancha';

// 23
console.log(str.lastIndexOf('a'));

// 6 (considera solamente hasta la posición 6)
console.log(str.lastIndexOf('l', 6));

// -1 (considera solamente hasta la posición 5)
console.log(str.lastIndexOf('l', 5));

// -1
console.log(str.lastIndexOf('noencontrado'));
```

- Método `search`: busca un string y devuelve la posición de la coincidencia. Es similar a `indexOf` pero con dos diferencias:
 - No permite buscar a partir de una determinada posición.
 - Acepta el uso de expresiones regulares.

```
const str = 'En un lugar de la mancha';

// 9 (expresión regular que busca la primera coincidencia de a, b o c)
console.log(str.search(/[abc]/));
```

- Método `slice`: extrae una parte de un string en base a dos posiciones. El método puede tomar 2 parámetros:
 - El primer parámetro es la posición inicial.
 - El segundo parámetro es la posición final (sin tomarla).

```
const str = 'Apple, Banana, Kiwi';

// Banana (la posición 13 no se toma)
console.log(str.slice(7, 13));

// Banana (la posición -6 no se toma)
console.log(str.slice(-12, -6));

// Banana, Kiwi (toma desde la posición 7 hasta el final del string)
console.log(str.slice(7));
```

```
// Banana, Kiwi (toma desde la posición -12 hasta el final del string)
console.log(str.slice(-12));
```

- Método substring: similar a slice, pero no acepta valores negativos.
- Método substr: similar a slice, pero el segundo parámetro representa la longitud del string extraido.

```
const str = 'Apple, Banana, Kiwi';

// Banana (toma los 6 siguientes caracteres desde la posición 7)
console.log(str.substr(7, 6));

// Banana, Kiwi (toma desde la posición 7 hasta el final del string)
console.log(str.substr(7));
```

- Método replace: reemplaza un determinado fragmento de un string por otro. Únicamente se reemplaza la primera aparición y es case sensitive (aunque pueden utilizarse expresiones regulares).

```
const str = 'Visita MICROSOFT! Microsoft!';

// Visita MICROSOFT! Corenetworks! (case sensitive y solamente primera coincidencia)
console.log(str.replace('Microsoft', 'Corenetworks'));

// Visita Corenetworks! Microsoft! (no case sensitive (i) y solamente primera coincidencia)
console.log(str.replace(/Microsoft/i, 'Corenetworks'));

// Visita Corenetworks! Corenetworks! (no case sensitive (i) y todas las coincidencias (g))
console.log(str.replace(/Microsoft/ig, 'Corenetworks'));
```

- Método toUpperCase: convierte todo el string a mayúsculas.

```
const str = 'Hello World!';

// HELLO WORLD!
console.log(str.toUpperCase());
```

- Método toLowerCase: convierte todo el string a minúsculas.

```
const str = ' HELLO WORLD!';
// hello world!
console.log(str.toLowerCase());
```

- Método concat: permite concatenar strings. Es una alternativa al operador de concatenación +

```
const text1 = 'Hello';
const text2 = 'World';
const text3 = '!';

// HelloWorld
console.log(text1.concat(text2));

// Hello World
console.log(text1.concat(' ', text2));

// Hello World!
console.log(text1.concat(' ', text2, text3));
```

- Método charAt: devuelve el carácter de una posición específica.

```
const text = 'HELLO WORLD';

// H
console.log(text.charAt(0));

// H
console.log(text[0]);

// undefined
console.log(text[20]);

//
console.log(text.charAt(20));
```

- Método charCodeAt: devuelve el valor en Unicode del carácter de una posición específica.

```
const text = 'HELLO WORLD';

// 72
console.log(text.charCodeAt(0));
```

- Método split: convierte un string a un array. El parámetro que recibe split es el carácter separador.

```
const text = 'a,b,c,d,e';
const array = text.split(',');
// ["a", "b", "c", "d", "e"]
console.log(array);

// a,b,c,d,e
console.log(array.toString());
```

- Método trim: elimina los espacios en blanco a ambos lados de un string.

```
const text = ' Hola ';
// Hola (sin espacios a los lados)
console.log(text.trim());
```

Ejercicio 1: escribe un programa que dado dos string compruebe si los dos primeros caracteres son iguales.

Ejercicio 2: escribe un programa que dado dos string compruebe si los dos primeros caracteres y los dos últimos son iguales.

Ejercicio 3: escribe un programa que compruebe que el substring "abc" existe en un string dado, pero no puede encontrarse ni al comienzo, ni al final.

Ejercicio 4: escribe un programa que dado un string y una determinada posición, compruebe que el carácter anterior y el posterior son iguales o no.

Ejercicio 5: escribe un programa que muestre por pantalla la posición de la segunda y tercera ocurrencia del carácter 'a' en un string dado.

Ejercicio 6: escribe un programa que devuelva si un string es palíndromo (se escribe igual hacia delante y hacia detrás). Ejemplo: "sometemos".

Ejercicio 7: escribe un programa que elimine el último carácter de un string.

Ejercicio 8: escribe un programa que inserte el carácter 'b' cada tres posiciones en un string.

Ejercicio 9: escribe un programa que convierta en mayúsculas la primera letra de cada palabra de un string.

Tu carrera digital ~

Módulo 2

Diseño de páginas interactivas frontend (JavaScript)

Array



Array

- Introducción
- Atributos y métodos de los arrays

Introducción

- Las arrays se utilizan para almacenar múltiples valores ordenados.
- La diferencia principal entre un array y un objeto es que un array utiliza como posiciones los números, mientras que los objetos utilizan identificadores. Además, los arrays se encuentran ordenados, mientras que los objetos no lo están.
- Formas de declarar un array.

```
// buena práctica
const coches1 = ['Saab', 'Volvo', 'BMW'];

// mala práctica
const coches2 = new Array('Saab', 'Volvo', 'BMW');

// buena práctica
const coches3 = [];

// mala práctica
const coches4 = new Array();
```

- Es posible almacenar distintos tipos de datos en un array.

```
const datos = [23, 'Volvo', true];
```

- Formas de comprobar que una variable es de tipo Array.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];

// true
console.log(Array.isArray(frutas));

// true
console.log(frutas instanceof Array);
```

- La primera posición de un array en JavaScript es la 0.

```
const coches = ['Saab', 'Volvo', 'BMW'];

// Saab
console.log(coches[0]);
```

- Iterar elementos de un array.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];
const lon = frutas.length;

// for clásico
for (let i = 0; i < lon; i++) {
    console.log(frutas[i])
}

// for...of
for (let fruta of frutas) {
    console.log(fruta)
}
```

Atributos y métodos de los arrays

- Atributo length: longitud de un array.

```
const coches = ['Saab', 'Volvo', 'BMW'];

// 3
console.log(coches.length);
```

- Método toString: convierte un array en una variable de tipo string utilizando la coma como separación.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];

// Banana,Orange,Apple,Mango
console.log(frutas.toString());
```

- Método join: convierte un array en una variable de tipo string utilizando un determinado separador que es pasado por parámetro.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];
```

```
// Banana * Orange * Apple * Mango
console.log(frutas.join(' * '));
```

- Método push: añade un nuevo elemento al final del array y retorna la longitud.

```
const coches = ['Saab', 'Volvo', 'BMW'];
// 4
console.log(coches.push('Renault'));

// ["Saab", "Volvo", "BMW", "Renault"]
console.log(coches);

// otra forma de agregar elementos
const coche2 = ['Saab', 'Volvo', 'BMW'];
coche2[coche2.length] = 'Renault';

// ["Saab", "Volvo", "BMW", "Renault"]
console.log(coche2);

// aunque no es una buena práctica añadir elementos por su posición
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];
frutas[10] = 'Lemon';

// ["Banana", "Orange", "Apple", "Mango", empty × 6, "Lemon"]
console.log(frutas);
```

- Método unshift: añade un elemento al principio y retorna el número de elementos.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];
// 5
console.log(frutas.unshift('Lemon'));

// ["Lemon", "Banana", "Orange", "Apple", "Mango"]
console.log(frutas);
```

- Método pop: elimina el último elemento de un array y lo retorna.

```
const coches = ['Saab', 'Volvo', 'BMW'];
// BMW
console.log(coches.pop());
```

```
// ["Saab", "Volvo"]
console.log(coches);
```

- Método shift: elimina el primer elemento de un array y lo retorna.

```
const coches = ['Saab', 'Volvo', 'BMW'];
// Saab
console.log(coches.shift());

// ["Volvo", "BMW"];
console.log(coches);
```

- Método delete: borra elementos de un array, pero dejando "agujeros" (holes) en el array.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];
delete frutas[0];

// [empty, "Orange", "Apple", "Mango"]
console.log(frutas);
```

- Método splice: permite añadir elementos, eliminar elementos o ambas operaciones al mismo tiempo. El método devolvería los elementos eliminados (si los hubiera). Posee los siguientes parámetros:
 - El primer parámetro es la posición del elemento en el que se comenzará a añadir o eliminar elementos.
 - El segundo parámetro define el número de elementos que serán eliminados. Si el valor es 0, entonces ningún elemento del array es eliminado.
 - El tercer parámetro y los siguientes son los elementos a añadir a partir de la posición definida en el primer parámetro.

```
// eliminar elementos

const frutas1 = ['Banana', 'Orange', 'Apple', 'Mango'];

// ["Orange", "Apple"]
console.log(frutas1.splice(1, 2));

// ["Banana", "Mango"]
console.log(frutas1);
```

```
// añadir elementos

const frutas2 = ['Banana', 'Orange', 'Apple', 'Mango'];

// []
console.log(frutas2.splice(2, 0, 'Lemon', 'Kiwi'));

// ["Banana", "Orange", "Lemon", "Kiwi", "Apple", "Mango"]
console.log(frutas2);
```

```
// eliminar y añadir elementos (primero elimina y luego añade)

const frutas3 = ['Banana', 'Orange', 'Apple', 'Mango'];

// ["Apple", "Mango"]
console.log(frutas3.splice(2, 2, 'Lemon', 'Kiwi'));

// ["Banana", "Orange", "Lemon", "Kiwi"]
console.log(frutas3);
```

- Método concat: une dos arrays. Este método puede aceptar tantos parámetros como arrays quieran ser concatenados.

```
const myGirls = ['Cecilie', 'Lone'];
const myBoys = ['Emil', 'Tobias', 'Linus'];
const myChildren = myGirls.concat(myBoys);

// ["Cecilie", "Lone", "Emil", "Tobias", "Linus"]
console.log(myChildren);
```

- Método slice: extrae un fragmento de un array y lo convierte en un nuevo array. Posee los siguientes parámetros:
 - El primer parámetro es la posición del primer elemento del nuevo array.
 - El segundo parámetro es la posición del último elemento del nuevo array (si este parámetro es omitido, entonces el último elemento del nuevo array es el último elemento del array original).
 - El método slice sin parámetros permite obtener una copia completa de un array.

```
const frutas = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];

const frutas1 = frutas.slice(1,2);

// ["Orange"]
```

```
console.log(frutas1);

const frutas2 = frutas.slice(1);

// ["Orange", "Lemon", "Apple", "Mango"]
console.log(frutas2);
```

- Método reverse: invierte un array.

```
const frutas = ['Banana', 'Orange', 'Apple', 'Mango'];

// ["Mango", "Apple", "Orange", "Banana"]
console.log(frutas.reverse());
```

- Método Math.max y Math.min: calcula el valor mínimo y máximo de un array.
 - Math.max y Math.min no aceptan arrays como parámetros.

```
// 100
console.log(Math.max(40, 100, 1, 5, 25, 10));

// 1
console.log(Math.min(40, 100, 1, 5, 25, 10));
```

- Método sort: ordena un array de forma alfabética.
 - Este método no funciona para números enteros. Para ordenar números enteros es necesario que el método sort reciba un callback de comparación con dos parámetros (a y b, por ejemplo).
 - El callback será ejecutado múltiples veces por JavaScript utilizando internamente un algoritmo de ordenación, donde a y b tomarán los diferentes valores del array, de tal forma que:
 - Si se quiere una ordenación ascendente, entonces debe retornarse a - b
 - Si se quiere una ordenación descendente, entonces debe retornarse b - a

```
const frutas2 = ['Banana', 'Orange', 'Apple', 'Mango'];

// ["Apple", "Banana", "Mango", "Orange"]
console.log(frutas2.sort());

// ordenación ascendente de números enteros mediante una función de comparación
const puntos = [40, 100, 1, 5, 25, 10];

// función para ordenación ascendente
const puntos2 = puntos.sort((a, b) => a - b);
```

```
// [1, 5, 10, 25, 40, 100]
console.log(puntos2);

// esta forma de ordenar elementos es muy potente puesto que pueden ordenarse
// también objetos a partir de alguna propiedad
const coches = [
  { type: 'Volvo', year: 2016 },
  { type: 'Saab', year: 2001 },
  { type: 'BMW', year: 2010 }
];

// función para ordenar ascendente por la propiedad year
const cochesOrdenados = coches.sort((a, b) => a.year - b.year);

// [{type: "Saab", year: 2001}, {type: "BMW", year: 2010}, {type: "Volvo", year: 2016}]
console.log(cochesOrdenados);
```

Ejercicio 1: escribe un programa que genere un array de 6 valores de tipo number y sean mostrados en pantalla utilizando un bucle for clásico y un bucle for...of

Ejercicio 2 : escribe un programa que almacene los nombres de tres colores en un array y los muestre por pantalla mediante un bucle for...of

Ejercicio 3: escribe un programa con un array que almacene los nombres de tres colores. A continuación, crea otro array vacío e inserta en él todos los elementos del primer array mediante un for...of y el método push.

Ejercicio 4: escribe un programa que dado dos arrays, devuelva el número de elementos que son iguales.

Ejercicio 5: escribe un programa que extraiga una porción del array a partir de una posición inferior y una superior y lo almacene en otro array.

Ejercicio 2 del proyecto: escribe un programa que almacene los objetos creados en el ejercicio anterior del proyecto dentro de un array (un array por cada modelo de datos). A continuación, recorre cada uno de los arrays y muestra todas propiedades.

Módulo 2

Diseño de páginas interactivas frontend (TypeScript)

Fundamentos

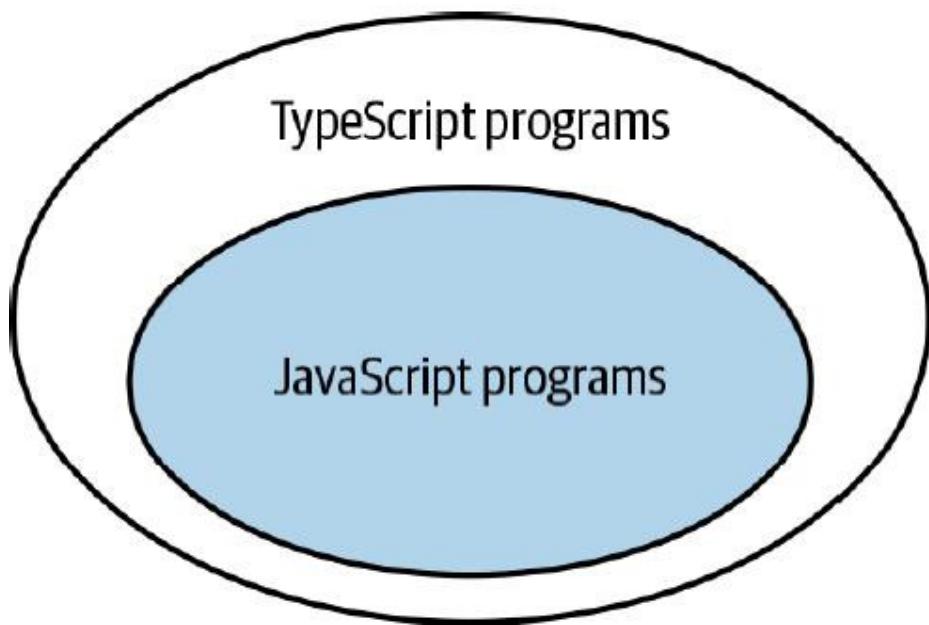


Fundamentos

- ◆ 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"
  }
}
```

```

    "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 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;
    }
}
```


Módulo 2

Diseño de páginas interactivas frontend (TypeScript)

Variables



Variables

- ◆ Introducción
- ◆ Tipos básicos
- ◆ Tipos complejos
 - Tipo *object*
 - Tipo *array*
 - Tipo *tupla*
 - Tipo *enum*
 - Tipo *any*
- ◆ Tipos especiales
 - Tipo *union*
 - Tipo *literal*
 - Tipo *alias*
 - Tipo *unknown*
 - Tipo *void*
 - Tipos *null* y *undefined*
 - Tipo *never*
- ◆ Detección de tipos
- ◆ Ámbito de variables

Introducción

- ◆ Una de las ventajas importantes de TypeScript es la posibilidad de tipar las variables.
- ◆ Las variables en TypeScript se declaran con *const* (si la variable no modificará nunca su valor) o con *let* (si la variable modificará su valor posteriormente), seguido del tipo de variable y su asignación.

```
const a: number = 1;
```

- ◆ Las variables declaradas con *const* deben ser asignadas a un valor obligatoriamente. No sucede lo mismo con las variables declaradas con *let*.

```
// error porque las variables declaradas con const requieren de la asignación de un valor
// const a: number;

// correcto porque se declara la variable y posteriormente se cambia el valor
let b: number;
b = 2;
c = 4;
```

- Visual Studio Code podría mostrar un aviso si la asignación de un valor a una variable infiere automáticamente su tipo, por lo que escribir el tipo es redundante.

```
const nombre: string = 'Mario Rodríguez';
```

- Por tanto, no es necesario establecer explícitamente el tipo cuando se puede inferir a partir de su valor.

```
const nombre = 'Mario Rodríguez';
```

- Visual Studio Code infiere el tipo automáticamente, aunque las variables declaradas con `const` las considera literales.

```
let nombre: string
let nombre = 'Mario Rodríguez';
```

```
const nombre: "Mario Rodríguez"
const nombre = 'Mario Rodríguez';
```

- En la declaración de las funciones, la inferencia de tipo automática no es posible, por lo que en este caso será necesario establecer el tipo para cada uno de los parámetros que recibe la función.

```
const sumar = (x: number, y: number): number => {
  return x + y;
}
```

- No es posible cambiar el tipo de una variable una vez ha sido inferido, excepto si se tipa con `any` (o un tipo similar ambiguo).

```
// f es una variable de tipo string
let f = 'holo';

// error porque se está asignando un valor numérico a una variable de tipo string
// f = 0;
```

Tipos básicos

- Tipo *boolean*: almacena valores *true* o *false*.

```
const estaEncendido = false;
```

- Tipo *number*: al igual que en JavaScript, en TypeScript solamente existe un tipo de variable numérica para almacenar valores enteros y flotantes.

```
const decimal = 6;
const hex = 0xf00d;
const binary = 0b1010;
const octal = 0o744;
```

- Tipo *string*: TypeScript también permite utilizar comillas simples, dobles e invertidas (plantillas de cadena de texto).
 - No existe consenso entre elegir comillas simples o dobles, si bien es cierto que Google recomienda el uso de comillas simples (ver documentación de Angular) para poder incluir strings con HTML (que puede contener comillas dobles como valor de los atributos de las etiquetas).

```
const nombre = 'Mario Rodríguez';
const sentencia = `Hola, mi nombre es ${nombre}`;

// las plantillas de cadena de texto pueden abarcar varias líneas
const sentencia: string = `Hola,
  mi nombre es ${nombre}`;
```

Tipos complejos

Tipo *object*

- En TypeScript existen tres conceptos relacionados con los objetos que suelen dar lugar a confusión:
 - *Object*
 - *object*
 - *{}*
- *Object* es un tipo de dato no primitivo para trabajar con objetos y que está presente también en JavaScript. Entre las propiedades de *Object* se encuentran: *Object.keys()*, *Object.values()*, *Object.freeze()*, etc.

```
const objeto1 = {a: 1, b: 2, c: 3}

// [1, 2, 3]
console.log(Object.values(objeto1));
```

- *object* y {} representan el mismo concepto en TypeScript. Una variable se puede tipar con este tipo, pero solamente se podrán acceder a las propiedades de los objetos nativos de JavaScript (*toString*, *valueOf*, *hasOwnProperty*, etc..), con independencia de las propiedades establecidas para el objeto.
- Por tanto, *object* y {} permiten crear un objeto genérico, sin propiedades adicionales.

```
const objeto2: object = {
  edad: 22,
  b: 2,
};

const objeto3: {} = {
  edad: 1,
  b: 2,
};

// error porque objeto2 y objeto3 son de tipo object y este tipo de dato no posee la
// propiedad edad
// console.log(objeto2.edad);
// console.log(objeto3.edad);
```

- En lugar de tipar con objetos genéricos, es importante ser más específico y crear un objeto tipando todas sus propiedades, una a una.

```
const objeto4: {
  edad: number;
  b: number;
} = {
  edad: 1,
  b: 2,
};

console.log(objeto4.edad);
```

- Aunque la mejor práctica es utilizar la inferencia de tipos comentada con anterioridad. En este caso, los tipos se infieren automáticamente a partir de la asignación de los valores de las propiedades.

```
const objeto5 = {
  edad: 1,
  b: 2,
};

console.log(objeto5.edad);
```

- Sin embargo, cuando el objeto contiene propiedades donde no se puede inferir el tipo (por ejemplo, una tupla), será necesario establecer los tipos para cada una de las propiedades.

```
const objeto6: {
  a: number;
  b: string;
  c: [number, boolean];
} = {
  a: 1,
  b: 'hola',
  c: [1, true]
};
```

- En TypeScript no puede definirse un objeto vacío y, a continuación, agregar propiedades de forma dinámica (excepto si se declara como *any* o se utiliza el tipo genérico *Partial* que se describe en la sección de Genéricos).

```
const objeto7 = {
  mostrarHola: () => {
    console.log("Hola");
  }
};

// error porque no se pueden agregar propiedades de forma dinámica
// objeto7.numero = 1;

const objeto8: any = {};

// correcto porque objeto8 se ha declarado como any
objeto8.numero = 1;
```

- Sin embargo, se pueden declarar propiedades de un objeto como opcionales (utilizando el carácter ?). El valor de una propiedad opcional puede ser asignado posteriormente.

```
const objeto9: {
  // la propiedad numero se declara como opcional
  numero?: number;
```

```
a: number
} = {
  // no es necesario asignar un valor a la propiedad numero porque es opcional
  a: 2
};

// undefined
console.log(objeto9.numero);

// correcto porque la propiedad numero se ha declarado como propiedad opcional
objeto9.numero = 1;

// 1
console.log(objeto9.numero);
```

Tipo array

- Funcionan como en JavaScript, pero almacenando valores del mismo tipo.

```
// lista1 solamente puede almacenar valores numéricos
const lista1: number[] = [1, 2, 3];

// otra forma de declarar un array
const lista2: Array<number> = [1, 2, 3];
```

- Si no se define ningún tipo, entonces se pueden almacenar valores de distinto tipo considerando los valores asignados inicialmente.

```
// cosas3 será tipado como (string | number | boolean)[]
const cosas3 = [true, 1, 'baloncesto'];

// error porque cosas3 únicamente puede almacenar valores de tipo string, number y
// boolean
// cosas3[0] = {}

// correcto porque los valores booleanos son permitidos
cosas3.push(true);

// cosas4 sí puede contener valores de cualquier tipo (aunque esto no es recomendable)
const cosas4: any[] = [true, 1];
```

Tipo tupla

- Es un array donde se establecen los tipos de valores permitidos.

- En la asignación de una tupla es obligatorio que el número de valores y los tipos coincidan con los establecidos en la declaración.

```
// declaración de una tupla de tres valores: el primero string, el segundo number y el tercero boolean
let tupla1: [string, number, boolean];

// correcto porque los tipos de los valores coinciden con su declaración: string, number y boolean
tupla1 = ['hello', 10, true];

// error porque el tercer elemento es de tipo number y no boolean
// tupla1 = ['hello', 22, 3];

// error porque la tupla declara tres elementos y no dos
// tupla1 = ['hello', 1];

// error porque la tupla declara tres elementos y no cuatro
// tupla1 = ['hello', 22, true, true];
```

- Pueden utilizarse los métodos de los arrays de JavaScript para insertar o eliminar elementos (*push* y *pop*) en una tupla, pero solamente pueden agregarse elementos del tipo declarado en la tupla. Esto sucede también con los arrays (excepto en arrays declarados como vacíos).

```
const tupla2: [string, number] = ['hello', 10];

// correcto
tupla2.push('bye');
tupla2.push(12);
console.log(tupla2);

// error porque el tipo booleano no es permitido para esta tupla
// tupla2.push(true);
```

```
const array2: number[] = [1, 2];

// correcto porque se inserta un valor de tipo numérico
array2.push(2);

// error porque solamente se pueden almacenar valores de tipo numérico
// array2.push('hello');

// declaración de un array vacío
const array3 = [];
```

```
// correcto porque el array está vacío (puede insertarse cualquier tipo de valor)
array3.push(2);
array3.push('hello');
array3.push(true);
```

- Por tanto, en la asignación de variables de tipo tupla se deben especificar tantos elementos como los establecidos en la declaración, pero con posterioridad el número de elementos puede variar si se utilizan métodos de los arrays como *pop* o *push*
- El acceso a elementos puede fallar si se invoca a un método que no existe para el tipo definido.

```
const tupla3: [string, number] = ['hello', 10];

// correcto
console.log(tupla3[0].substr(1));

// error, porque el tipo number no posee el método substr (es específico de los strings)
console.log(tupla3[1].substr(1));
```

- TypeScript arrojará un error cuando se acceda a la posición de un elemento que no existe en la tupla. No sucede lo mismo con los arrays (que devuelve *undefined*).

```
const tupla4: [string, number] = ['hello', 10];

// error, porque no existe valor para el elemento en la posición 2
// console.log(tupla4[2]);
```

```
const array4: number[] = [1,2];

// undefined
console.log(array4[3]);
```

- TypeScript no puede inferir automáticamente el tipo de una tupla y es necesario siempre declarar explícitamente los tipos de los elementos que contendrá.

Tipo *enum*

- Permite asignar nombres de variables más amigables a valores numéricos, de forma secuencial.
- Por defecto, el primer elemento de una enumeración comienza en 0 y los siguientes elementos toman los valores sucesivos.

```
enum Color {Rojo, Verde, Azul}
const c: Color = Color.Verde;

// 0
console.log(Color.Rojo);

// 1
console.log(Color.Verde);

// 2
console.log(Color.Azul);

// 1
console.log(c);
```

- Las enumeraciones pueden comenzar con otro valor.

```
enum Color {Rojo = 1, Verde, Azul}

// 1
console.log(Color.Rojo);

// 2
console.log(Color.Verde);

// 3
console.log(Color.Azul);
```

- Es posible cambiar manualmente el valor de cada uno de los elementos de la enumeración.

```
enum Color {Rojo = 1, Verde = 2, Azul = 4}

// 1
console.log(Color.Rojo);

// 2
console.log(Color.Verde);

// 4
console.log(Color.Azul);
```

- También se pueden asignar valores de tipo string a los elementos de la enumeración.
 - El elemento inmediatamente posterior al asignado a un string se le debe obligatoriamente asignar un valor.

```
// Azul obligatoriamente se le debe asignar un valor porque el anterior es de tipo string
enum Color {Rojo = 1, Verde = 'verde', Azul = 400, Negro}
```

```
// 1
console.log(Color.Rojo);

// green
console.log(Color.Verde);

// 400
console.log(Color.Azul);

// 401
console.log(Color.Negro);
```

- Puede accederse al valor del elemento de la enumeración utilizando el valor numérico asociado al elemento.

```
enum Color {Rojo = 1, Verde, Azul}
```

```
// no es el elemento ubicado en la posición 2, sino el elemento asociado al valor 2 en la
declaración de la enumeración (Verde)
console.log(Color[2]);
```

Tipo *any*

- La variable que es declarada como *any* acepta cualquier tipo de dato (supertipo universal). Es preferible evitar su uso.

```
let notSure: any = 4;
notSure = 'Hello world!';
notSure = false;
```

- Puede pensarse que el tipo *any* es similar al tipo *object*, pero no es cierto. Una variable de tipo *any* es mucho más permisiva que *object* y TypeScript no realiza ninguna comprobación sobre sus propiedades.

```
const cualquiera: any = 4;
const objeto: object = {};
```

```
// correcto porque el método ifItExists podría existir para el tipo any
cualquiera.ifItExists();
```

```
// correcto porque el métodotoFixed podría existir para el tipo any
// cualquiera.toFixed();
```

```
// error porque el métodotoFixed no existe para el tipo Object
// objeto.toFixed();
```

- El tipo *any* puede ser útil para definir un array de elementos de distinto tipo.

```
const lista: any[] = [1, true, 'free'];
lista[1] = 'hola';
```

Tipos especiales

Tipo *union*

- El tipo *union* permite declarar variables que admiten varios tipos, utilizando el carácter |

```
// la variable z admite valores de tipo string y number
let z: number | string = 4;
```

```
// correcto porque la variable admite valores de tipo string
z = 'hola';
```

```
// error porque la variable no admite valores de tipo booleano
// z = true;
```

- También se puede utilizar el carácter | para definir un array que almacene diferentes tipos de valores.

```
const a: (number | boolean)[] = [1, 2, 3, true];
```

- Aunque la inferencia de tipos también detecta automáticamente que se trata de una variable de tipo array para tipos de datos específicos.

```
const b = [1, 2, 3, true];
```

- TypeScript puede tener problemas para inferir determinadas acciones en tipos ambiguos como *union*

```
function combinar(input1: number | string, input2: number | string) {
    // error porque TypeScript desconoce si el tipo number | string puede ser concatenado
    // (no es específico)
    // return input1 + input2;
}

combinar('Marcos', 'Rodríguez');
```

- Sin embargo, la función `typeof` es detectada por TypeScript para detectar los tipos ambiguos de `union`

```
function combinar(input1: number | string, input2: number | string) {

    if(typeof(input1) == "number" && typeof(input2) == "number") {
        // number.toString() y number.toString() pueden ser concatenados
        return input1.toString() + input2.toString();
    }

    else if(typeof(input1) == "string" && typeof(input2) == "string") {
        // string y string pueden ser concatenados
        return input1 + input2;
    }
}

combinar('Marcos', 'Rodríguez');
```

Tipo literal

- El tipo literal permite que una variable solo admita valores específicos previamente establecidos en la declaración.
- Las variables de tipo literal no pueden ser declaradas con `const`

```
// la variable color únicamente puede tomar el valor rojo, verde o azul
let color: 'rojo' | 'verde' | 'azul';

// correcto
color = 'rojo';
color = 'verde';

// error porque el amarillo no es uno de los literales considerados en la declaración de la
// variable
// color = 'amarillo';
```

Tipo alias

- Los tipo alias permiten crear tipos personalizados mediante la palabra reservada `type` de TypeScript y utilizando tipos union y literales.

```
// creación del tipo Combinable, que admite valores de tipo string, number y el literal true
type Combinable = string | number | true;

// declaración de una variable de tipo Combinable
let cosa: Combinable;

// correcto porque esta variable admite todos estos tipos de valores
cosa = 'rojo';
cosa = 2;
cosa = true;

// error porque la variable no admite el valor false
// cosa = false;
```

- El tipo alias permite crear tipos personalizados de objetos

```
// declaración de tipo
type Usuario = { nombre: string; edad: number };

// declaración de variable
const usuario: Usuario = {
  nombre: 'Marcos',
  edad: 23
};
```

Tipo *unknown*

- Es similar a `any`, pero más restrictivo, lo que lo convierte en un tipo más recomendable.
- Al igual que `any`, también permite almacenar valores de cualquier tipo.
- Sin embargo, la asignación de una variable de tipo `unknown` a una variable de otro tipo más restrictivo (string, boolean, etc.) genera un error en TypeScript.

```
let a: unknown;
let b: any;
let c: string;

a = 'hola';
b = 4;
c = 'a';
```

```
// correcto porque a la variable c se le asigna una variable de tipo any (cualquier valor)
c = b;

// error porque a la variable c se le asigna una variable de tipo unknown
// c = a;

// es necesario esta comprobación para que el transpilador de TypeScript no arroje un
// error
if (typeof (a) === 'string') {
  c = a;
}
```

- Una variable de tipo *unknown* solo puede ser asignada a otra variable de tipo *unknown* o *any*, excepto si previamente se ha establecido alguna comprobación con *typeof*, *instanceof* (la comprobación será automáticamente detectada por el analizador sintáctico de TypeScript).

Tipo *void*

- Es el opuesto a *any*. Habitualmente utilizado en las funciones.

```
function warning1(): void {
  console.log('Esto es un mensaje de warning');
}
```

- Es posible también utilizarlo en variables, pero no tiene mucho sentido puesto que únicamente pueden asignarse valores como *undefined* o *null*

```
// correcto porque void acepta undefined o null solamente
const unusable1: void = undefined;

// error porque se asigna una variable numérica
// const unusable2: void = 1;
```

Tipos *null* y *undefined*

- Funcionan igual que en JavaScript, aunque no tiene mucho sentido declarar variables con estos tipos porque solamente pueden tomar este valor.

```
const u: undefined = undefined;
const n: null = null;
```

- En versiones antiguas de TypeScript, otros valores con tipos distintos podían tomar los valores `null` y `undefined`, pero ya no está permitido.

```
let a: string = "";
let b: number = 5;

// error
// a = undefined;

// error
// b = null;
```

- El comportamiento anterior puede evitarse cambiando la configuración del transpilador de TypeScript, concretamente asignando explícitamente el valor `false` al parámetro `strictNullChecks` (por defecto está establecido a `false`, excepto si el parámetro `strict` se establece a `true`).

Tipo `never`

- Utilizado por funciones que no retornan nunca (habitualmente generan excepciones o son bucles infinitos).

```
function error(message: string): never {
  throw { message };
}

function infiniteLoop(): never {
  while (true) { }
}

// error porque esta función finaliza
// function returnFunction(): never { }

// error porque esta función finaliza
// function returnFunction(): never { return; }
```

Detección de tipos

- Para detectar el tipo puede utilizarse la función `typeof`, teniendo en cuenta que los valores devueltos son los mismos que en JavaScript (`boolean`, `number`, `bigint`, `string`, `object`, `function` y `undefined`) y que algunos tipos de variables pueden devolver resultados inesperados.

```
const persona = {
  nombre: 'José',
```

```
mayorDeEdad: true,  
edad: 23,  
numeroGrande: 100n,  
domicilio: null,  
localidad: undefined,  
aficiones: ['Fútbol', 'Baloncesto'],  
hola: () => {  
    console.log('hola');  
}  
};  
  
// string  
console.log(typeof (persona.nombre));  
  
// boolean  
console.log(typeof (persona.mayorDeEdad));  
  
// number  
console.log(typeof (persona.edad));  
  
// bigint  
console.log(typeof (persona.numeroGrande));  
  
// object porque null es considerado como un objeto  
console.log(typeof (persona.domicilio));  
  
// undefined  
console.log(typeof (persona.localidad));  
  
// object porque los arrays son considerados como objetos  
console.log(typeof (persona.aficiones));  
  
// function  
console.log(typeof (persona.hola));  
  
// object  
console.log(typeof (persona));
```

- Para garantizarse que una variable es de tipo array puede utilizarse la propiedad *isArray* del objeto *Array*.

```
const a = [1, 2, 3];  
  
// object  
console.log(typeof (a));  
  
// true  
console.log(Array.isArray(a));
```

- Tambien se puede utilizar *instanceof* para comprobar las variables de tipo *Array* y otras especiales como las fechas.

```
const a = [1, 2, 3];  
  
// true  
console.log(a instanceof Array);  
  
const b = new Date();  
  
// true  
console.log(b instanceof Date);
```

Ámbito de variables

- *let* y *const* funcionan igual que en ECMAScript 6 o superior con respecto al ámbito de las variables. Su uso es recomendable respecto a *var* o a variables globales.

Módulo 2

Diseño de páginas interactivas frontend (TypeScript)

Funciones



Funciones

- ◆ Introducción
- ◆ Tipado de funciones
- ◆ void y undefined en las funciones
- ◆ Tipar callbacks
- ◆ Sobrecarga

Introducción

- ◆ Las funciones en TypeScript funcionan de forma similar a las funciones de JavaScript y poseen la misma sintaxis.
- ◆ Al igual que en JavaScript, las funciones en TypeScript pueden referirse a variables que están fuera del cuerpo de la función.

```
const z = 100;

const sumar = (x, y) => {
    return x + y + z;
}

console.log(sumar(1, 2))
```

Tipado de funciones

- ◆ Al igual que las variables, las funciones también pueden tiparse.

```
// función tipada
function sumar1(x: number, y: number): number {
    return x + y;
}

// función anónima tipada con inferencia de tipo
const sumar2 = function(x: number, y: number): number { return x + y; };

// función de flecha tipada con inferencia de tipo
const sumar3 = (x: number, y: number): number => x + y;

// función de flecha tipada sin inferencia de tipo (se utiliza el tipo Function)
const sumar4: Function = (x: number, y: number): number => x + y;

// creación de una variable que posteriormente almacenará una función
let sumar5: Function;
```

```
// error
// sumar5 = 5;

sumar5 = sumar4;
```

- Es posible definir parámetros opcionales mediante el carácter '?'. Por defecto, todos los parámetros son obligatorios.

```
// sin parámetros opcionales (ejemplo de dos parámetros obligatorios)
const construirNombre1 = (nombre: string, apellidos: string) => {
    return nombre + ' ' + apellidos;
};

// error porque solamente se ha pasado un parámetro
// const resultado1 = construirNombre1('Bob');

// error porque se han pasado tres parámetros
// const resultado2 = construirNombre1('Bob', 'Adams', 'Sr.');

// correcto
const resultado3 = construirNombre1('Bob', 'Adams');

// los parámetros opcionales no pueden ubicarse antes que los obligatorios. Si no se
// suministran, su valor es undefined
const construirNombre2 = (nombre: string, apellidos?: string) => {
    return apellidos ? nombre + ' ' + apellidos : nombre;
};

// correcto
const resultado4 = construirNombre2('Bob');

// error porque la función acepta uno o dos parámetros
// const resultado5 = construirNombre2('Bob', 'Adams', 'Sr.');

// correcto
const resultado6 = construirNombre2('Bob', 'Adams');
```

- También se pueden asignar parámetros por defecto a los parámetros de las funciones, que son considerados como parámetrosopcionales.

```
const construirNombre1 = (nombre: string, apellidos = 'Smith') => {
    return nombre + ' ' + apellidos;
};

// correcto, el segundo parámetro es opcional
const resultado1 = construirNombre1('Bob');
```

```
// correcto
const resultado2 = construirNombre1('Bob', undefined);

// error porque se han pasado tres parámetros
// const resultado3 = construirNombre1('Bob', 'Adams', 'Sr.');

// correcto
const resultado4 = construirNombre1('Bob', 'Adams');

// los parámetros con valores por defecto no necesariamente se ubican después de los
// parámetros obligatorios
const construirNombre2 = (nombre = 'Will', apellidos: string) => {
    return `${nombre} ${apellidos}`;
};

// error porque la función espera dos parámetros
// const resultado5 = construirNombre2('Bob');

// error porque la función espera dos parámetros
// const resultado6 = construirNombre2('Bob', 'Adams', 'Sr.');

// correcto
const resultado7 = construirNombre2('Bob', 'Adams');

// correcto
const resultado8 = construirNombre2(undefined, 'Adams');

// error porque el primer parámetro debe ser tipo string
// const resultado9 = construirNombre2(true, 'Adams');
```

- Por último, el número de parámetros no necesariamente tiene que ser fijo. Puede utilizarse el operador de propagación o rest (...)

```
const construirNombre = (nombre: string, ...resto: string[]) => {
    return `${nombre} ${resto.join(' ')}`;
};

const persona = construirNombre('Joseph', 'Samuel', 'Lucas', 'MacKinzie');
```

void y undefined en las funciones

- Tanto en JavaScript como en TypeScript, las funciones que no retornan nada realmente devuelven *undefined* (incluso aunque estén declaradas para que retornen void en TypeScript).

```
const mostrarHola1 = (): void => {
    console.log('Hola');
```

```
}
```

```
// undefined
console.log(mostrarHola1());
```

- Sin embargo, en TypeScript se producirá un error si se declara una función para que retorne *undefined* si no se ha incluido *return* dentro de la función.

```
const mostrarHola2 = (): undefined => {
  console.log('Hola');
}

// error porque la función mostrarHola2 no incluye la cláusula return
// console.log(mostrarHola2());
```

- Será necesario incluir *return* dentro de la función para que la función sea sintácticamente válida para TypeScript.

```
const mostrarHola3 = (): undefined => {
  console.log('Hola');
  return;
}

// correcto
console.log(mostrarHola3());
```

Tipar callbacks

- Los callbacks se tipan en TypeScript en la declaración de la función que la recibe.
 - Generalmente el callback es siempre el último parámetro que recibe una función.
 - El callback generalmente no retorna nunca ningún valor (*void*).

```
// el tercer parámetro que recibe la función es un callback (recibe una variable de tipo
// numérica y no retorna nada)
const suma = (a: number, b: number, cb: (result: number) => void) => {
  const result = a + b;
  cb(result);
};

// no es necesario declarar la variable result de tipo number porque TypeScript la infiere
// del tipado del callback
suma(1, 2, (result) => {
  console.log(result);
});
```

Sobrecarga

- Al contrario que en JavaScript, la sobrecarga en TypeScript es permitida.
- Para ello es necesario declarar la función tantas veces como se necesite (pudiendo cambiar el número y los tipos de datos de los argumentos y el tipo de la variable devuelta). A continuación es necesario implementar una función con todos los argumentos establecidos a any (o combinación de tipos que comprenda las posibles opciones de las declaraciones de las funciones), incluyendo también el tipo de la variable a devolver.
- No se puede utilizar funciones de flechas para establecer funciones sobrecargadas. Es necesario emplear la sintaxis clásica de JavaScript con *function*

```
function pickCard(x: string): string;
function pickCard(x: number): number;

function pickCard(x: number | string): number | string {
    return (typeof x === 'string') ? x+1 : x+2;
}

// correcto
pickCard(4);

// correcto
pickCard('Hola');

// error porque la función sobrecargada no recibe valores booleanos
// pickCard(true);
```

- Con la sobrecarga de funciones se puede evitar la ambigüedad de los tipos (por ejemplo, *any* o tipo *union*), si bien es cierto que será necesario diferenciarlos dentro de la implementación de la función.

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-CanActivateRoutingModule: Create a CanActivate routing guard snippet
 ag-CanDeactivateRoutingModule: 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

Módulo 2

Diseño de páginas interactivas frontend (Angular)

Fundamentos (II)

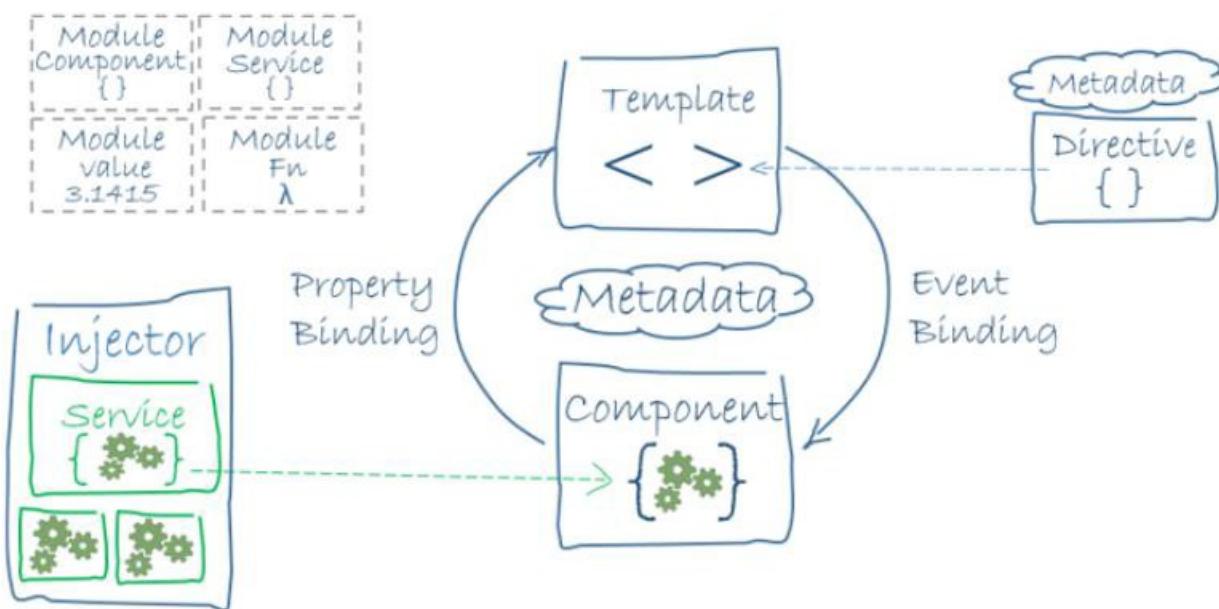


Fundamentos (II)

- ◆ 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.

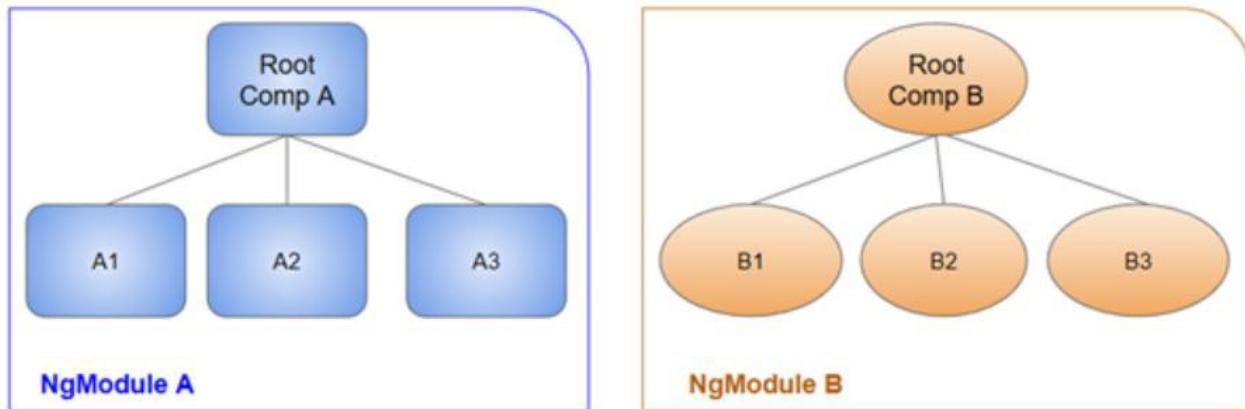
```
// src/main.ts
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

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.

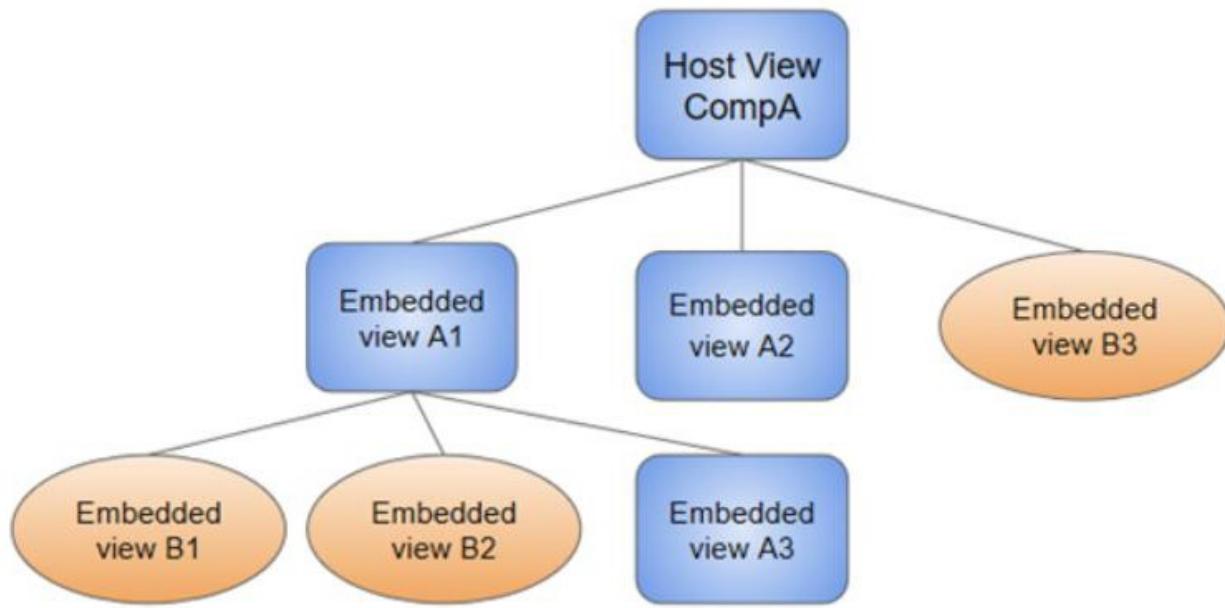
// exports: [AppComponent]

})

// 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>

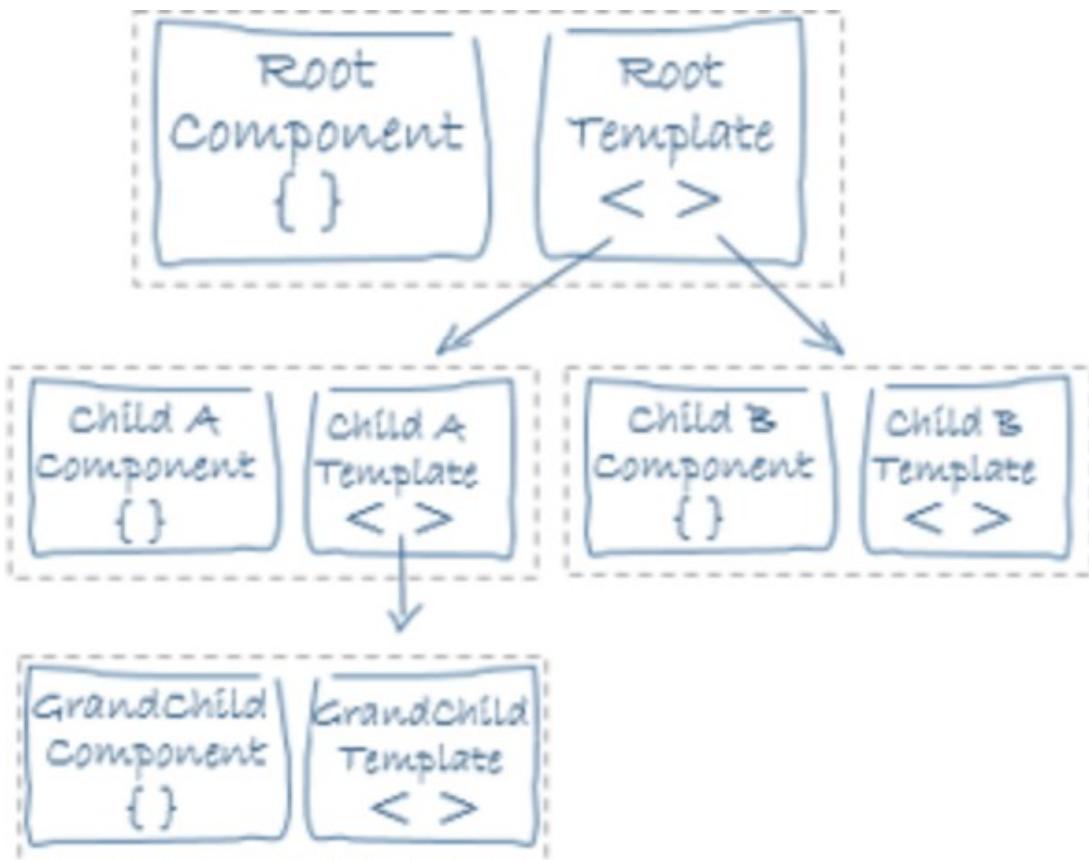
<!-- binding de tipo interpolación -->
<p>My current hero is {{currentHero.name}}</p>

<!-- 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>

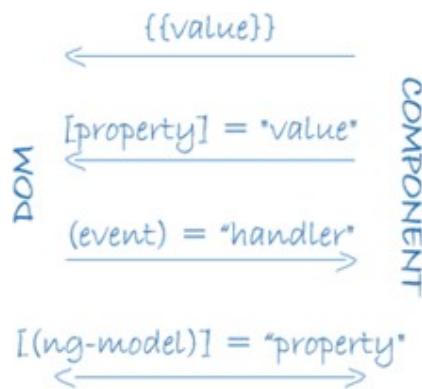
<!-- selector de otro componente combinado con la directiva *ngIf y un tipo de binding entre componentes -->
<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

- 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) -->

`{{hero.name}}`

<!-- 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) -->

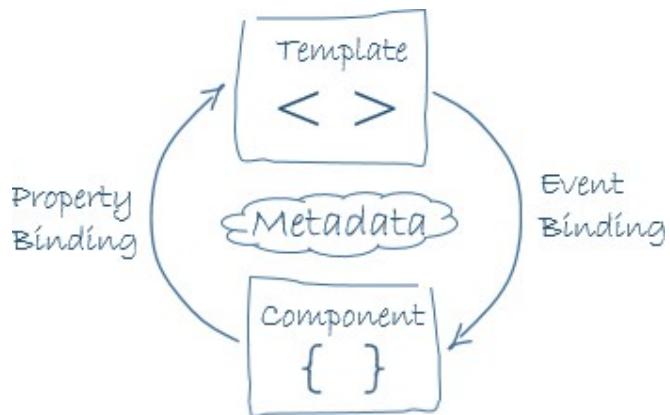
`<button [disabled]="isUnchanged">Disabled Button</button>`

<!-- 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 -->

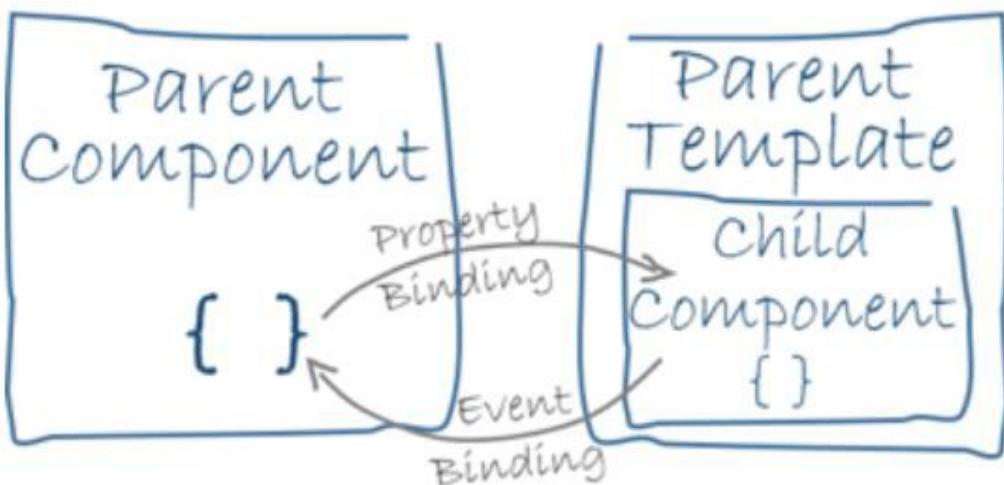
`<li (click)="selectHero(hero)">`

<!-- 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>
```

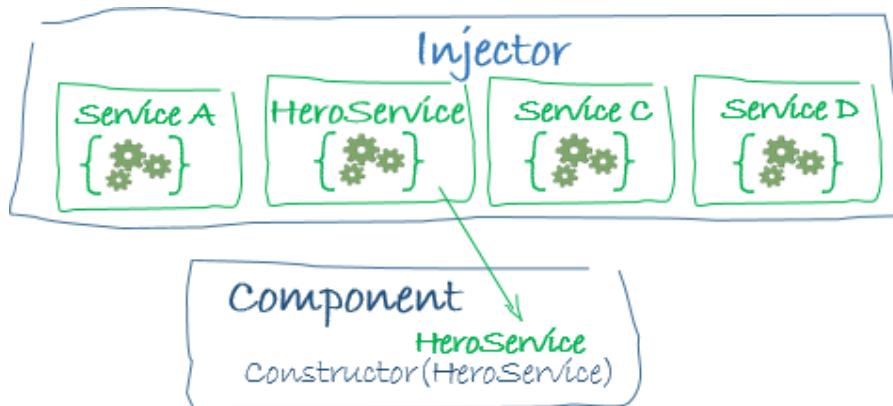
- Las directivas de atributo alteran la apariencia o el comportamiento de un elemento HTML. En las plantillas se muestran como atributos HTML normales (de ahí el nombre).

```
<!-- 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 injectado 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 injectado.
- 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.

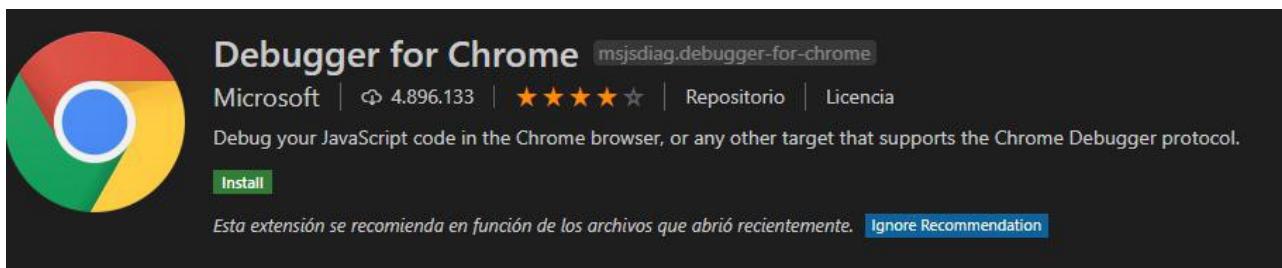
```
export class Hero {
  constructor(public id: number, public name: string) { }
}
```

- Y en otro componente distinto se puede utilizar la clase creada.

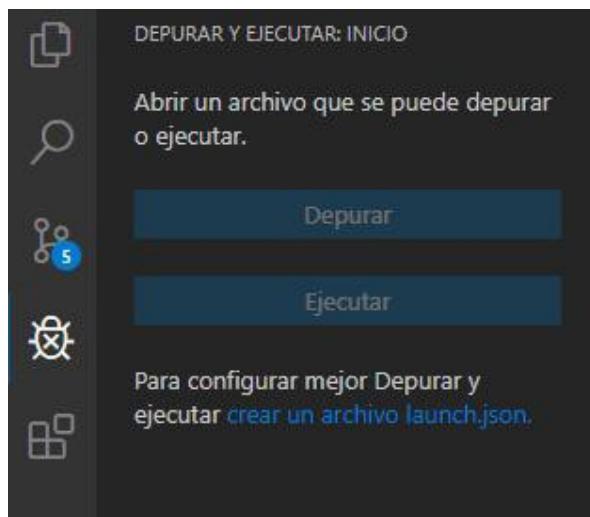
```
const heroes = [
  new Hero(1, 'Windstorm'),
  new Hero(13, 'Bombasto'),
  new Hero(15, 'Magneta'),
  new Hero(20, 'Tornado')
];
const myHero = this.heroes[0];
```

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() {  
15         • this.title = 'Hello world!';  
16     }  
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 2

Diseño de páginas interactivas frontend (Angular)

Data binding (fundamentos)



Data binding (fundamentos)

- ◆ Introducción
- ◆ Tipos de data binding
- ◆ Interpolación
- ◆ Expresiones de plantilla
- ◆ Operadores importantes en las expresiones de plantilla
 - Operador pipe (|)
 - Operador de navegación segura (?)
- ◆ Variables de entrada de plantilla
- ◆ Variables de referencia de plantilla
 - Contexto de las expresiones de plantilla

Introducción

- ◆ La dualidad componente/plantilla en Angular se asimila a la presenteada por la arquitectura modelo-vista-controlador (MVC) del backend. En Angular, el componente reproduce la parte del controlador o modelo, y la plantilla representa la vista.
- ◆ HTML es el lenguaje de las plantillas en Angular.
- ◆ Casi toda la sintaxis de HTML está permitida. El elemento script es una excepción y está prohibido, eliminando el riesgo de ataques de inyección de código. En la práctica, la etiqueta script se ignora, apareciendo una advertencia en la consola del navegador.
- ◆ Otro tipo de etiquetas como html, body o head no tienen ninguna función útil en las plantillas y deberían evitarse.
- ◆ El vocabulario HTML de las plantillas se puede ampliar con componentes y directivas nuevas que aparecen como nuevos elementos y atributos.
- ◆ La instancia de un componente es creada cuando se referencia a su selector asociado desde la plantilla HTML de otro componente (o del archivo principal src/index.html).

```
<body>
  <app-root></app-root>
</body>
```

- ◆ De forma predeterminada, Angular genera componentes con un archivo de plantilla HTML y CSS independientes. Para habilitar el estilo inline puede utilizarse la partícula -t y -s:

```
ng generate component hero -t # para HTML inline
ng generate component hero -s # para CSS inline
ng generate component hero -ts # para ambos
```

Tipos de data binding

- ◆ El data binding es la comunicación o enlace de datos entre el archivo Typescript del componente y el archivo HTML de la plantilla.
- ◆ 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.

```
// sintaxis de la interpolación (utilizando llaves dobles)
{{expression}}
```

```
// sintaxis del property binding, attribute binding, class binding y style
binding (utilizando corchetes)
[target]="expression"
```

```
// similar al anterior, pero utilizando la forma canónica utilizando mediante el
prefijo bind
bind-target="expression"
```

- Desde la plantilla al código: event binding.

```
// sintaxis del event binding (utilizando paréntesis)
(target)="statement"
```

```
// similar al anterior (forma canónica utilizando el prefijo on)
on-target="statement"
```

- Two-way binding: en ambas direcciones.

```
// sintaxis del two-way binding (utilizando corchetes y paréntesis)
[(target)]="expression"
```

```
// similar al anterior (forma canónica utilizando el prefijo bindon)
bindon-target="expression"
```

- ◆ Los binding tienen un nombre de destino a la izquierda del signo igual, rodeado por la notación [], () o [()]. También es posible utilizar un prefijo (bind-, on-, bindon-). La notación es preferible al prefijo.

- Los datos en plantilla cambiarán a partir de algún evento asíncrono, como una pulsación de tecla, un click de ratón, una finalización de un temporizador o una respuesta a una solicitud HTTP.

Interpolación

- La forma más sencilla de mostrar el valor de una propiedad del componente en la plantilla es enlazándola mediante interpolación utilizando las llaves dobles {{}}.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
  `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

Ejercicio: cambiar el ejemplo anterior y ubicar el código de la plantilla en un archivo separado.

- Dentro de las llaves es posible utilizar multitud de expresiones, pero el resultado siempre es transformado en una cadena de texto, que es sustituida de forma literal.

```
<h5>Nombre: {{ alumno1.nombre }}</h5>
<h5>Nota: {{ obtenerNota() }}</h5>
<h5>Nota: {{ 5+2 }}</h5>
<p>La suma de 1 + 1 + x es {{1 + 1 + getX()}}.</p>
<div>
```

```
<!-- directiva *ngIf, donde allowed es una expresión de plantilla que hace referencia a esa propiedad del componente. El bloque div únicamente se muestra cuando la propiedad allowed existe (no es undefined o null) y además su valor debe ser true, un string no vacío, o un número distinto de 0 -->
```

```
<div *ngIf="allowed">Permitido</div>
```

```
<!-- directiva *ngIf, donde 1-1 sería una expresión de plantilla. El bloque div no se mostraría -->
```

```
<div *ngIf="1-1">Permitido</div>
```

```
<!-- error porque la sintaxis de la directiva está evaluando la expresión de plantilla {{1-1}}, que es sintácticamente incorrecta -->
```

```
<div *ngIf="{{1-1}}>Permitido</div>
```

- ◆ Están prohibidos los siguientes operadores de JavaScript en las expresiones de plantilla:

- Asignaciones (=, +=, -=, ...).
- Operadores como new, typeof, instanceof, etc.
- Punto y coma y comas.
- Operadores de incremento o decremento (++ y --).
- Operadores bit a bit (| y &).
- Algunos operadores específicos de ES2015+.

- ◆ Las expresiones de plantilla no pueden referirse a los objetos window o document del navegador. Tampoco pueden invocar a console.log.

- ◆ Otras cuestiones importantes sobre las expresiones de plantilla:

- No deben cambiar el valor de ninguna propiedad del componente. Esto solamente debería realizarse desde el propio código del componente.

- Deben ser ejecutadas de forma rápida. Si es computacionalmente costoso podría lastrar la experiencia del usuario.
 - Deben ser simples.
 - Deben ser idempotentes. Una expresión idempotente siempre devuelve exactamente la misma salida para una determinada entrada, independientemente de las veces que se ejecute.
 - En la mayoría de los casos las expresiones de plantilla son simplemente propiedades del componente y, ocasionalmente, expresiones booleanas.
- ♦ Operadores que sí son permitidos:
- Operadores de comparación y aritméticos.
 - Operador pipe (|): permite una transformación de los datos.
 - Operador (?): es una forma conveniente de protegerse contra los valores nulos e indefinidos cuando se utilizan objetos.

```
<!-- muestra true en la plantilla si se cumple la condición o false en caso contrario o si no existe la propiedad en el componente -->
```

```
<h5>{{allowed === -3}}</h5>
```

```
<!-- uso de pipe para convertir a minúscula -->
```

```
<h5>{{texto | lowercase}}</h5>
```

```
<!-- no se produce error si la propiedad objeto no existe en el componente -->
```

```
<h5>{{objeto?.time}}</h5>
```

```
<!-- error siempre, incluso cuando exista la propiedad objeto en el componente (el operador ? se utiliza para acceder de forma segura a las propiedades de un objeto) -->
```

```
<h1>{{objeto?}}</h1>
```

```
<!-- muestra true o false en función de si existe o no la propiedad en el componente -->
```

```
>
```

```
<h5>{{!texto}}</h5>
```

Ejercicio: probar todas las expresiones plantillas presentadas con anterioridad.

- Debido a su importancia, algunos de estos operadores utilizados en las expresiones de plantilla son vistos con más profundidad a continuación.

Operadores importantes en las expresiones de plantilla

Operador pipe (|)

- El resultado de una expresión puede requerir de alguna transformación antes de ser mostrado en la plantilla. Por ejemplo, mostrar un valor entero como moneda, forzar un texto a ser convertido en mayúsculas o filtrar una lista y ordenarla.

- Los pipes son una buena opción para pequeñas transformaciones. Se trata de funciones simples que aceptan un valor de entrada y devuelven un valor transformado. Son fáciles de aplicar dentro de las expresiones de plantilla utilizando el operador de tubería (|).
- Angular proporciona un conjunto de pipes que pueden utilizarse en cualquier plantilla. También es posible crear pipes personalizados.

```
@Component({
  selector: 'app-root',
  template: `
    {{title | uppercase}}
  `
})
export class AppComponent {
  title = 'hola';
}
```

- Los pipes pueden encadenarse.

```
@Component({
  selector: 'app-root',
  template: `
    {{title | uppercase | lowercase}}
  `
})
export class AppComponent {
  title = 'hola';
}
```

- Y también es posible suministrar parámetros al pipe.

```
@Component({
  selector: 'app-root',
  template: `
    <div>Birthdate: {{currentHero.birthdate | date:'longDate'}}</div>
  `
})
export class AppComponent {
  currentHero = { birthdate: new Date() };
}
```

- Para depuración, es interesante el uso del pipe json.

```
@Component({
  selector: 'app-root',
  template: `
    <p>{{hero}}</p>
    <p>{{hero | json}}</p>
  `
})
export class AppComponent {
  hero = { name: 'Batman' };
}
```

- El operador pipe tiene prioridad sobre el operador ternario de JavaScript (?:), aunque pueden utilizarse ambos con precaución.
 - Si se combina el operador pipe con el operador ternario, deben utilizarse paréntesis en el primer y segundo operando de la sintaxis del operador ternario (en el tercer operando es opcional, aunque recomendable).

```
@Component({
  selector: 'app-root',
  template: `
    <!-- error porque el pipe d no existe -->
    <!-- <p>{{ a ? b : c | d }}</p> -->

    <!-- correcto porque el pipe json sí existe y no necesita paréntesis porque se
        encuentra en el tercer operando -->
    <p>{{ a ? b : c | json }}</p>

    <!-- error porque el pipe se encuentra en el segundo operando y no hay paréntesis --
    >
    <!-- <p>{{ a ? b | c : d }}</p> -->

    <!-- correcto porque el pipe se encuentra en el segundo operando y hay paréntesis --
    >
    <p>{{a ? (b | json) : d }}</p>
  `
})
export class AppComponent {
  a = false;
  b = 2;
  c = {};
  d = 4;
}
```

Operador de navegación segura (?)

- El operador de navegación segura (?) es una forma conveniente de protegerse contra los valores nulos y/o indefinidos de las propiedades de un objeto.

// cuando se intenta acceder desde la plantilla a una variable que es undefined o null, no se visualiza nada y la aplicación no se cuelga

```
@Component({
  selector: 'app-root',
  template: `
    <div> {{currentHero}} </div>
  `
})
export class AppComponent {
```

// cuando se intenta acceder desde la plantilla a una propiedad de una variable que es undefined o null, Angular lanza una excepción y la aplicación se cuelga

```
@Component({
  selector: 'app-root',
  template: `
    <div> {{currentHero.a}}</div>

    <!-- no es mostrado porque la aplicación crashea antes -->
    {{ a }}
  `
})
export class AppComponent {
  a = 5;
}
/*  
ERROR TypeError: Cannot read property 'currentHero' of undefined  
*/
```

- Para solucionar el problema anterior, puede utilizarse la directiva *ngIf.

```
@Component({
  selector: 'app-root',
  template: `
    <div *ngIf="currentHero"> {{currentHero.a}}</div>
    {{a}}
  `
})
export class AppComponent {
  a = 4;
}
```

- Sin embargo, será necesario comprobar todas las propiedades intermedias para acceder a una propiedad más profunda.

```
<div *ngIf="nullHero && nullHero.name && nullHero.name.first"></div>
```

- Una forma más simple y limpia es utilizar el operador de navegación segura (?).

```
@Component({
  selector: 'app-root',
  template: `
    <div>{{hero?.name?.first}}</div>

    <!-- no produce una excepción -->
    <!-- al contrario que con la directiva *ngIf, el elemento div existe, pero se encuentra vacío -->
    <div>{{hero?.name?.first?.second?.third}}</div>
  `
})
export class AppComponent {
  hero = { name: { first: 'Batman' } };
}
```

Variables de entrada de plantilla

- La palabra clave let en una plantilla HTML crea una variable de entrada de plantilla.
- Esta variable es accesible dentro del elemento del bloque y dentro de sus descendientes. Su ámbito está restringido exclusivamente a ese contexto.

```
@Component({
  selector: 'app-root',
  template: `
    <!-- la directiva *ngFor crea la variable de entrada de plantilla hero -->
    <div *ngFor="let hero of heroes">

      <!-- la variable de entrada de plantilla hero es accesible aquí porque se encuentra dentro del bloque del for -->
      <p>{{hero.name}}</p>
    </div>
  `
})
export class AppComponent {
  heroes = [
    { name: 'Batman' },
    { name: 'Superman' },
  ]
}
```

Variables de referencia de plantilla

- Una variable de referencia de plantilla es una referencia a un elemento DOM dentro de una plantilla. También puede ser una referencia a un componente o directiva de Angular.
- Se declaran mediante un hash (#).
- Esta variable es accesible desde cualquier parte de la plantilla. Si se definen varias variables de referencia de plantilla con el mismo nombre, entonces sus valores pueden impredecibles en tiempo de ejecución.
- Puede utilizarse un event binding para pasar el valor de la variable al componente.,

```
@Component({
  selector: 'app-root',
  template: `
    <!-- ejemplo de event binding utilizando el evento click -->
    <input #phone value="666222333"/>
    <button (click)="print(phone)">Imprimir</button>
  `

})
export class AppComponent {
  print(phone: HTMLInputElement) {
    console.log(phone);
    console.log(phone.value);
  }
}
```

- Puede utilizarse el prefijo ref como alternativa a la notación del hash.

```
<input ref-phone value="666222333"/>
```

Contexto de las expresiones de plantilla

- Una expresión de plantilla puede referirse a propiedades de la clase del componente, a variables de entrada de plantilla (declaradas con let) o a variables de referencia de plantilla (declaradas con #).
- Si se hace referencia a un nombre que pertenece a más de uno de estos espacios de nombres, la variable de entrada de plantilla tiene prioridad, seguido de la variable de referencia de plantilla y, por último, la propiedad del componente.

```
@Component({
  selector: 'app-root',
  template: `
```

<!-- colisión de nombre porque el componente tiene una propiedad hero que coincide con una variable de entrada de plantilla. Tiene prioridad la variable de entrada de plantilla -->

```
<div *ngFor="let hero of heroes">{{hero.name}}</div>
```
})
export class AppComponent {
 hero = {name: 'Superman'};
 heroes = [
 { name: 'Batman' }
];
}
```

@Component({  
 selector: 'app-root',  
 template: `<!-- colisión de nombre. Las variables de entrada de plantilla tiene prioridad respecto a las variables de referencia de plantilla -->  
 <input #hero value="Hola">  
 <div \*ngFor="let hero of heroes">

<!-- variable de entrada de plantilla tiene más prioridad que la variable de referencia de plantilla -->  
 {{ hero.name }}  
 </div>  
 <!-- se imprime la variable de referencia de plantilla -->  
 {{ hero.value }}

`)
export class AppComponent {  
 hero = {name: 'Superman'};  
 heroes = [  
 { name: 'Batman' }  
 ];
}

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 un lenguaje de programación para desarrollar programas muy pequeños, veloces, confiables y transportables.



## Características de Java

# Orientado a Objetos

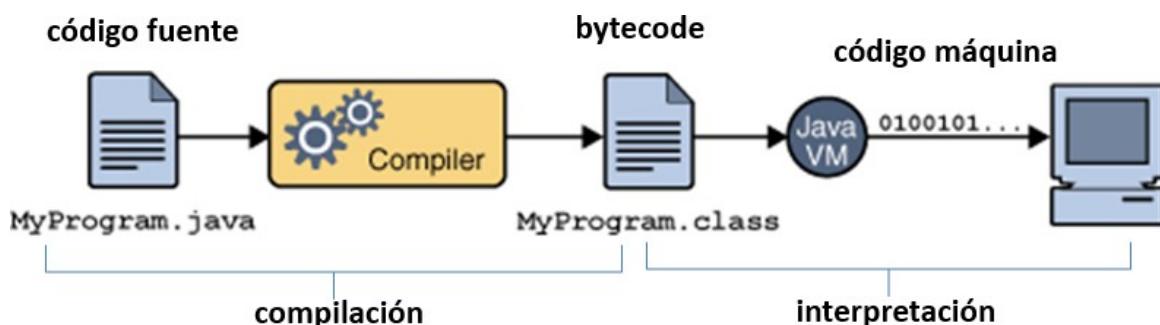
Compilado

Interpretado

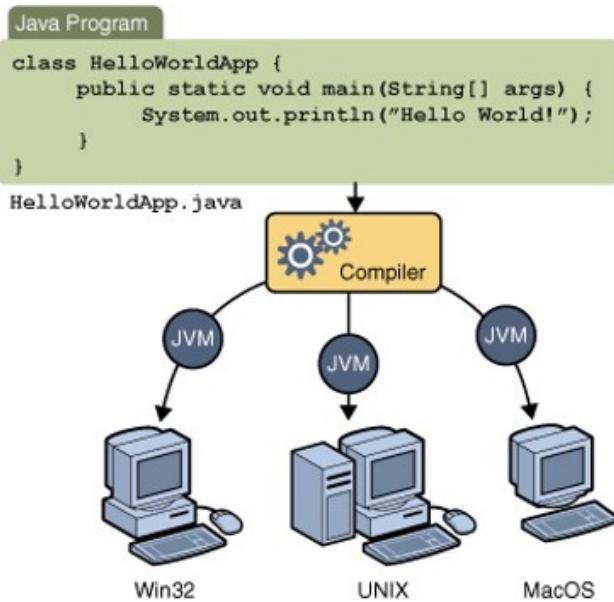
Independiente de la plataforma

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).

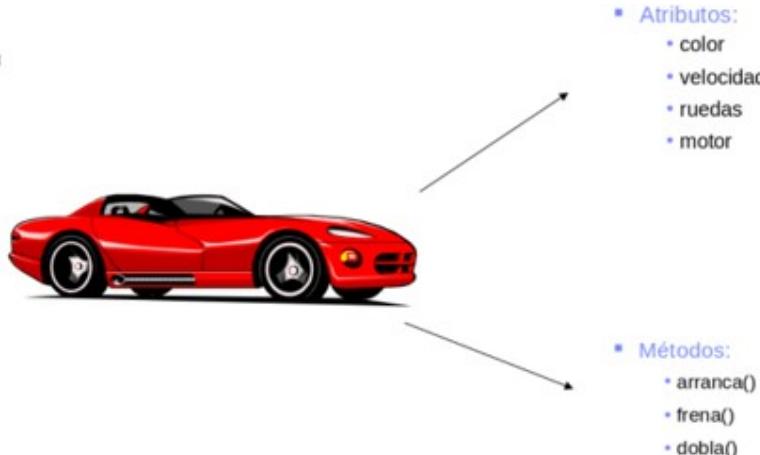


- Independiente de la plataforma: el código bytecode generado por el compilador puede ser interpretado en casi cualquier sistema operativo, dispositivo o arquitectura de computadores(JVM).



*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

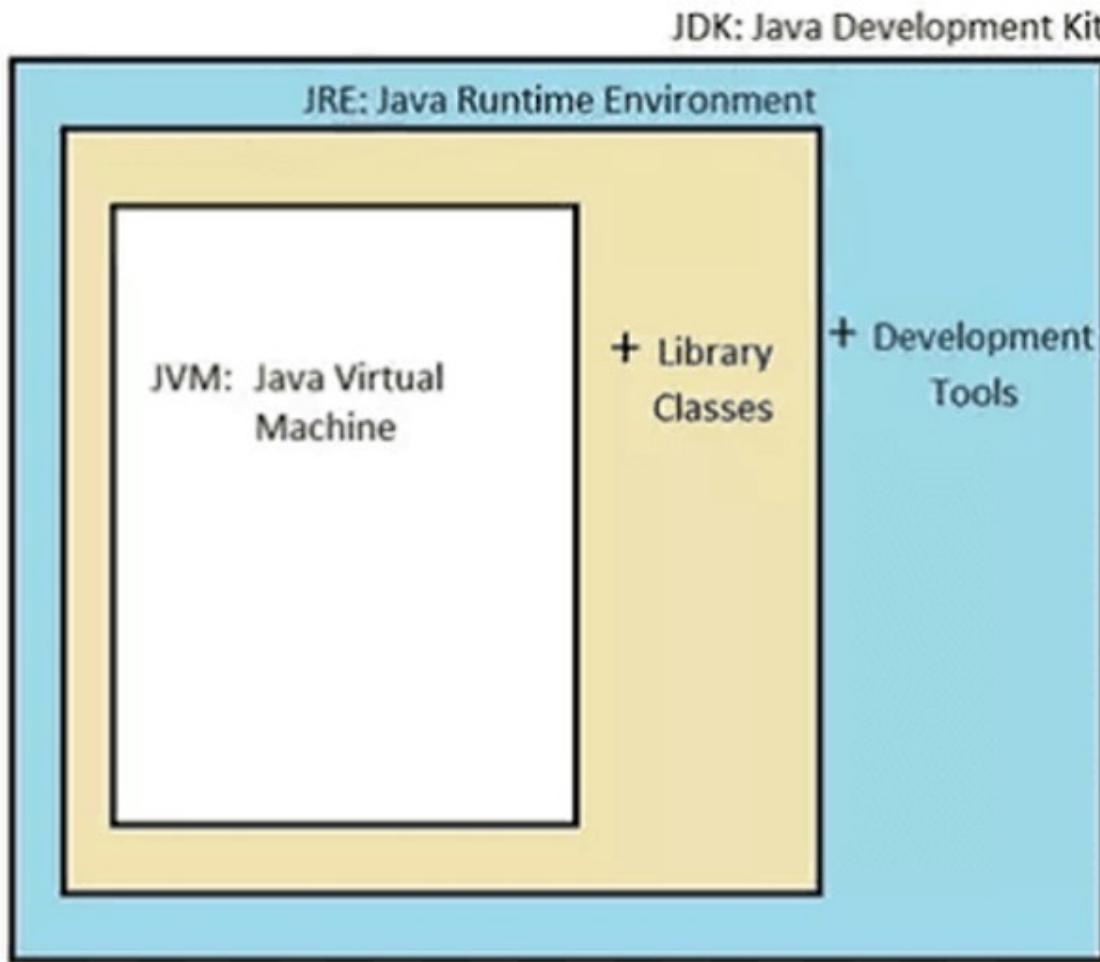
|Versión de Java EE / Jakarta EE|Año| |J2EE 1.2|1999| |J2EE 1.3|2001| |J2EE 1.4|2003| |Java EE 5|2006| |Java EE 6|2009| |Java EE 7|2013| |Java EE 8|2017| |Jakarta EE 8|2019| |Jakarta EE 9|2020| |Jakarta EE 9.1|2021|

- 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).

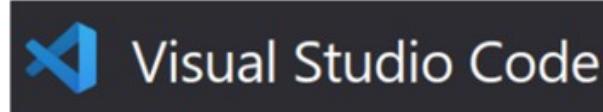
## Software para Java

- 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](#).



- ◆ 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).

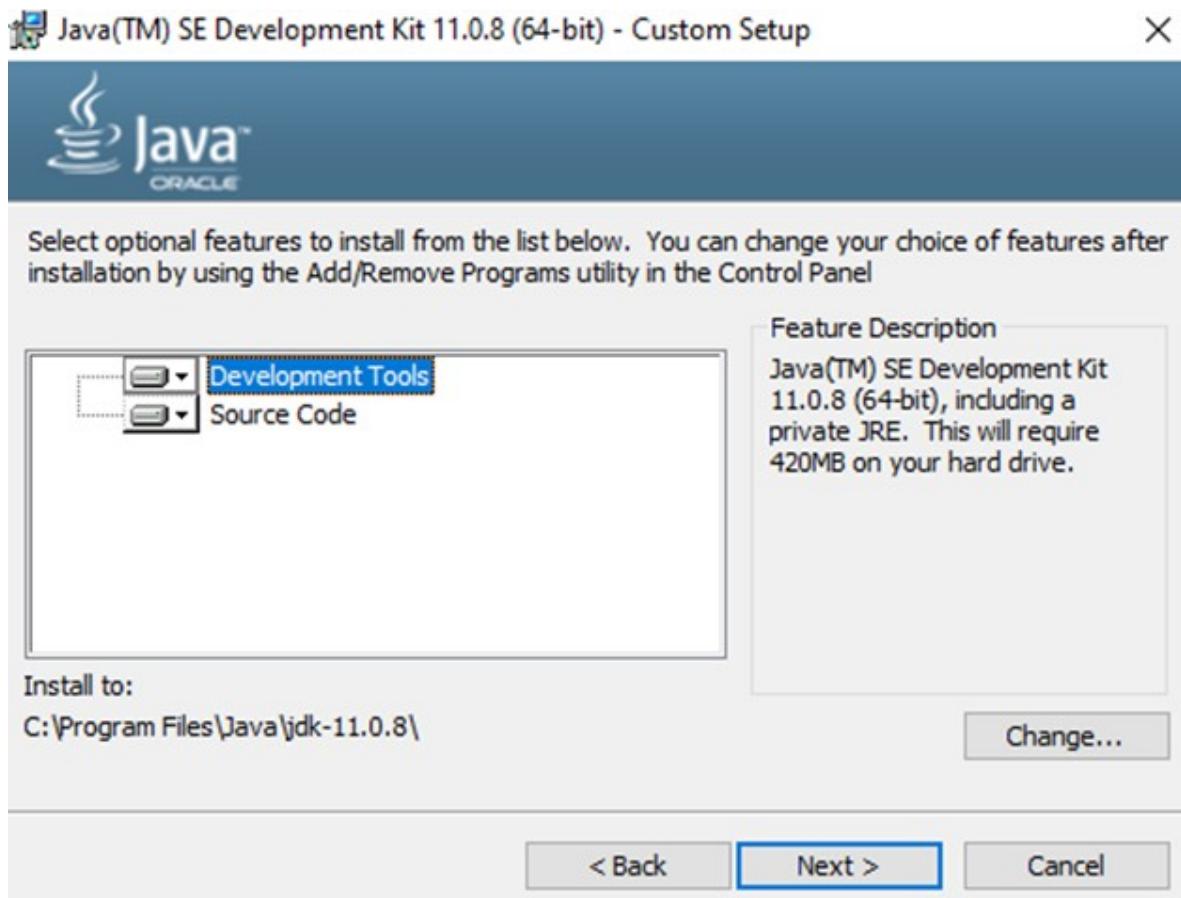


## Instalación del JDK y Eclipse

- 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.



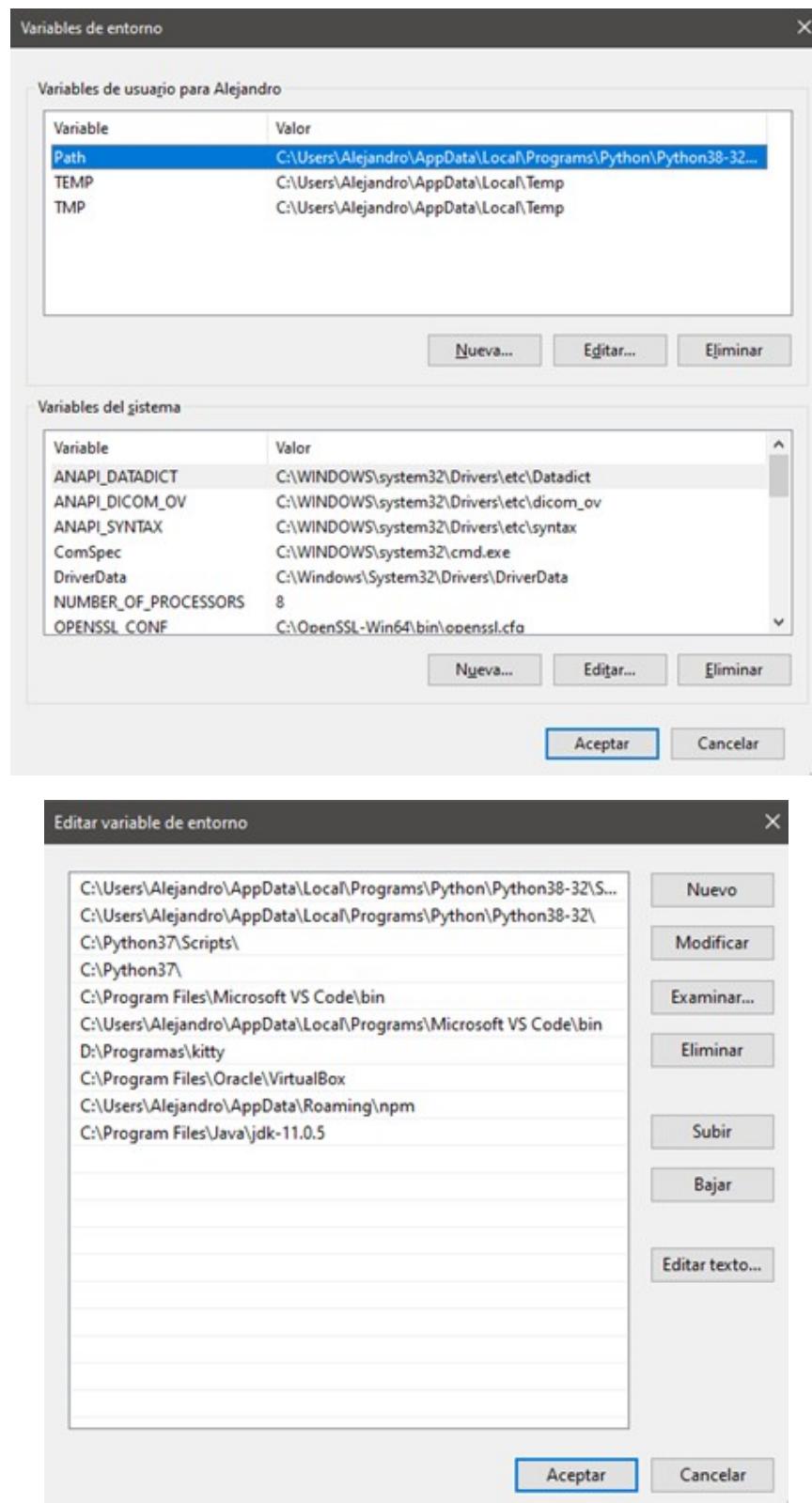
- A continuación comenzará el proceso de instalación.



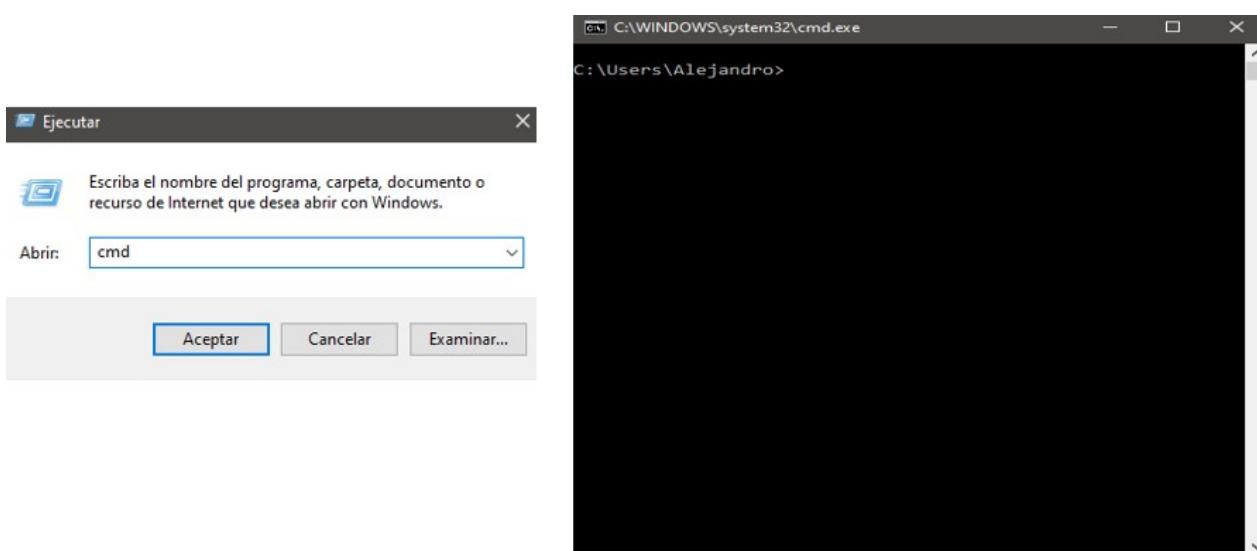
- Al finalizar, cerrar el asistente de instalación.



- 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).

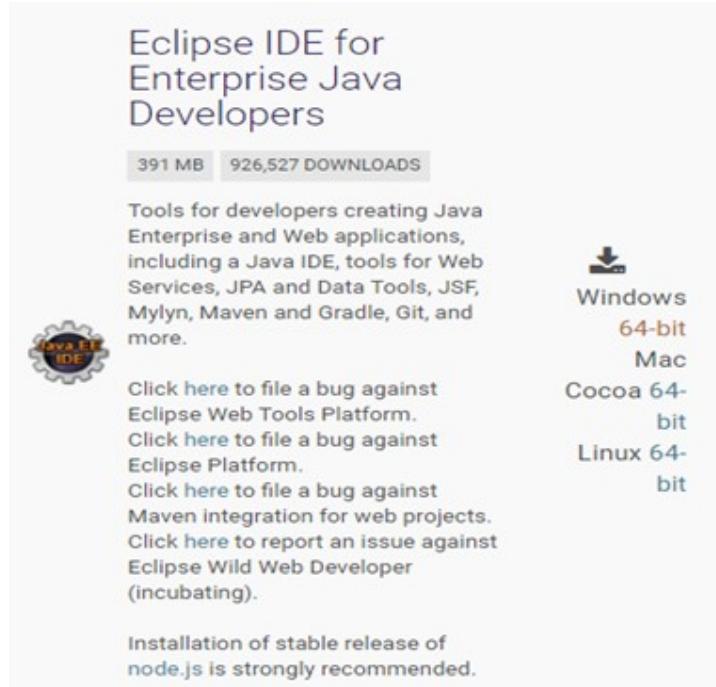
```
C:\Users\master>jshell
| Welcome to JShell -- Version 11.0.8
| For an introduction type: /help intro

jshell> 1+2
$1 ==> 3

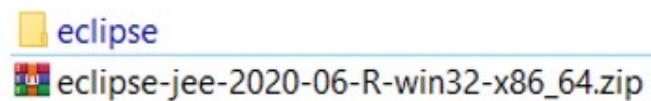
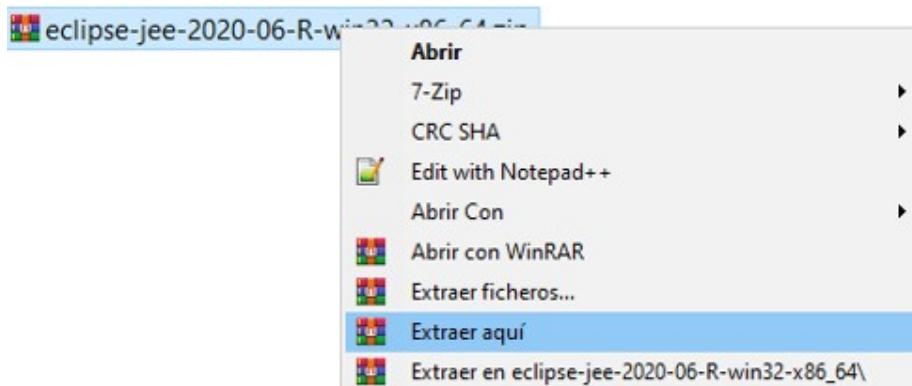
jshell> 25*3
$2 ==> 75

jshell> /exit
| Goodbye
```

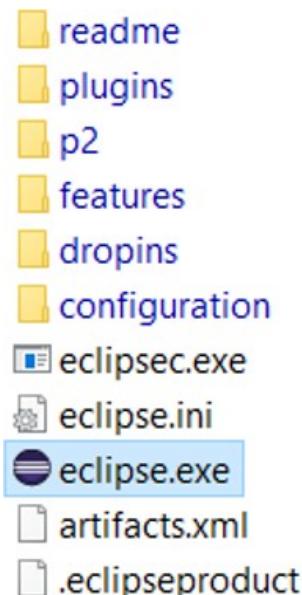
- Eclipse puede descargarse desde su [página web oficial]  
[\(https://www.eclipse.org/downloads/packages/\)](https://www.eclipse.org/downloads/packages/)
- Concretamente es necesario descargar Eclipse IDE for Enterprise Java Developers.



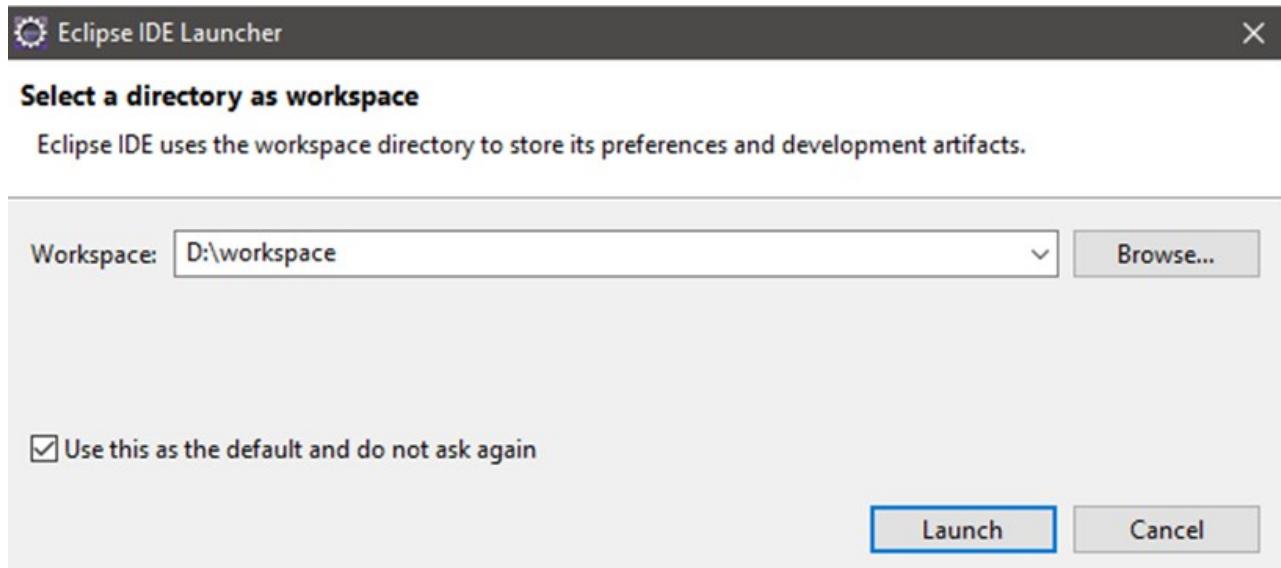
- A continuación se descomprime el archivo descargado.



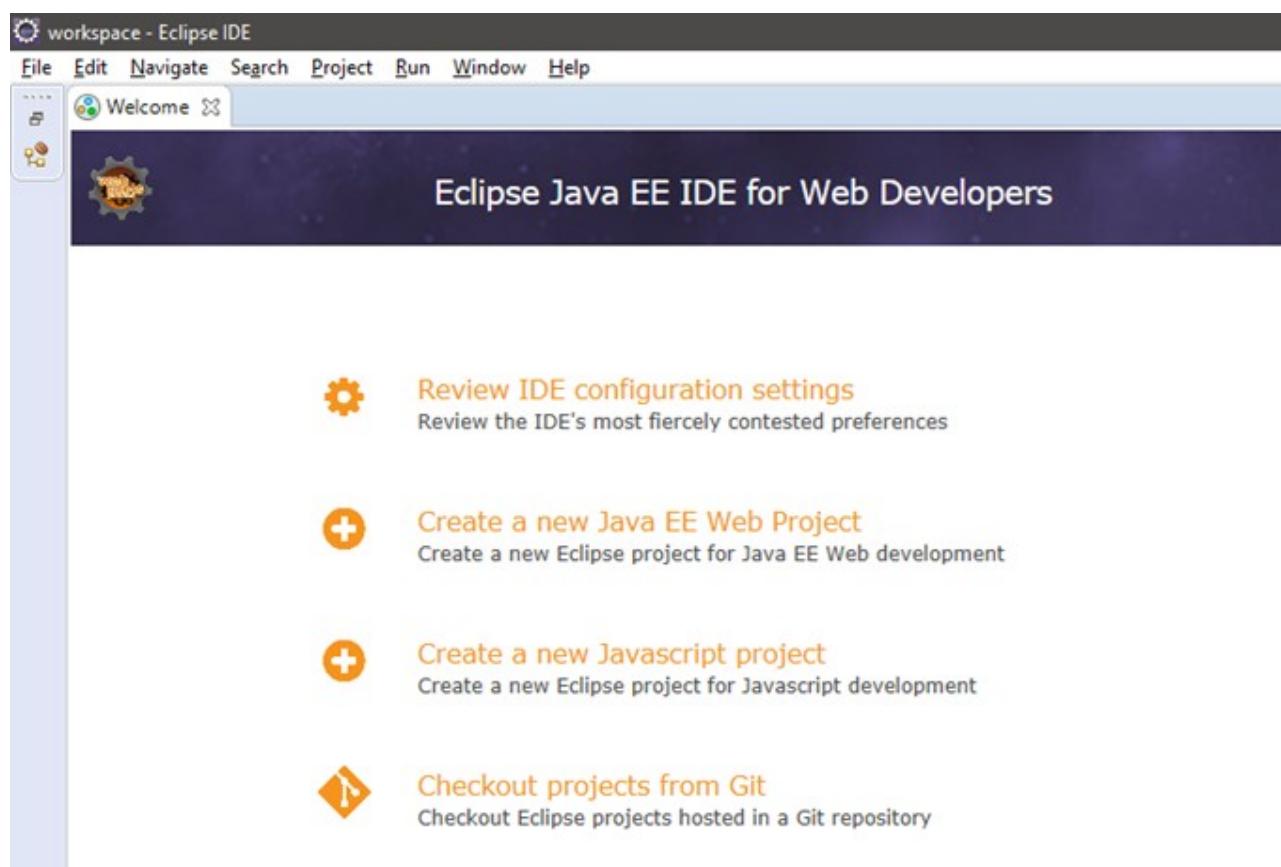
- Acceder al directorio eclipse y ejecutar el archivo eclipse.exe.



- Seleccionar el directorio de trabajo (donde se almacenará todo el código fuente de los programas). Debería de ser un directorio vacío.

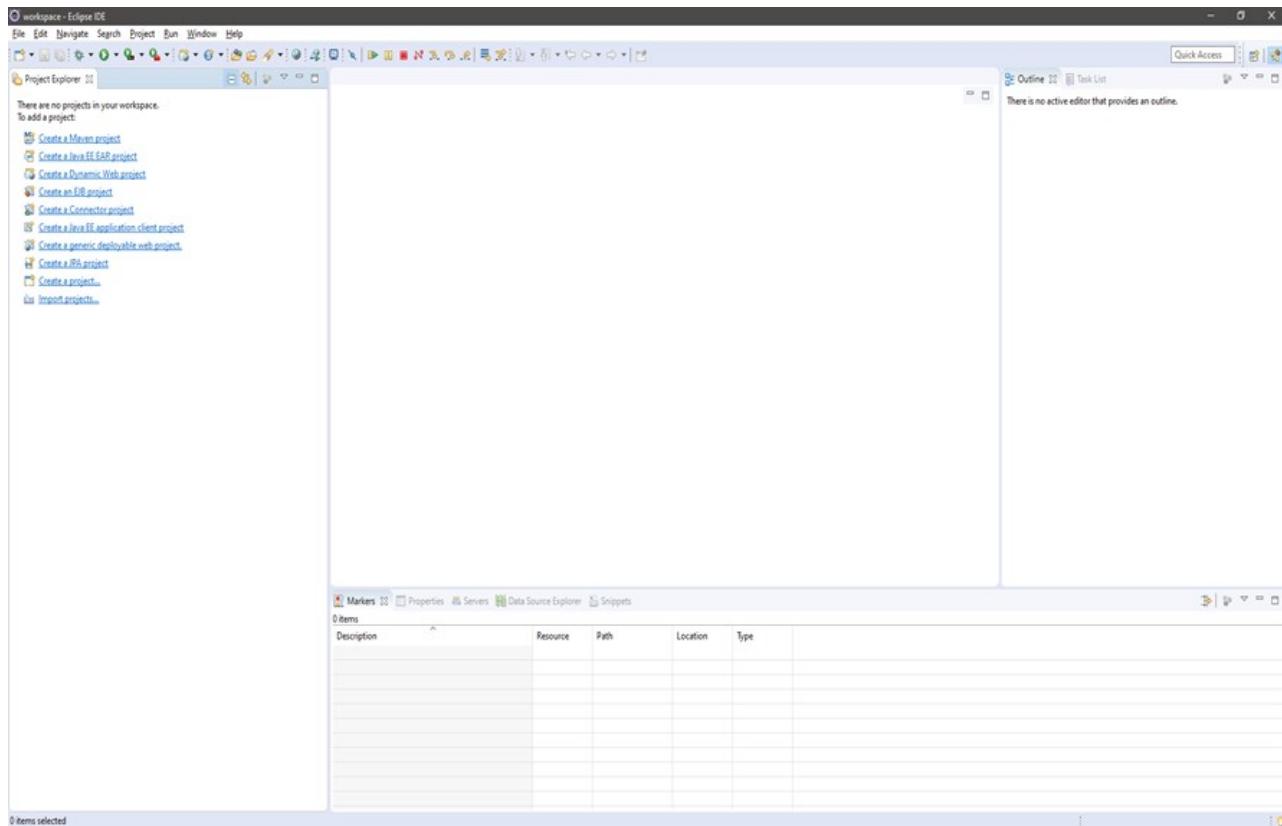
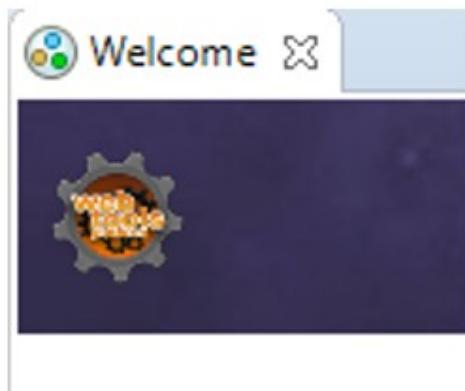


- Eclipse EE IDE debe abrirse y presentar una interfaz similar a la siguiente:



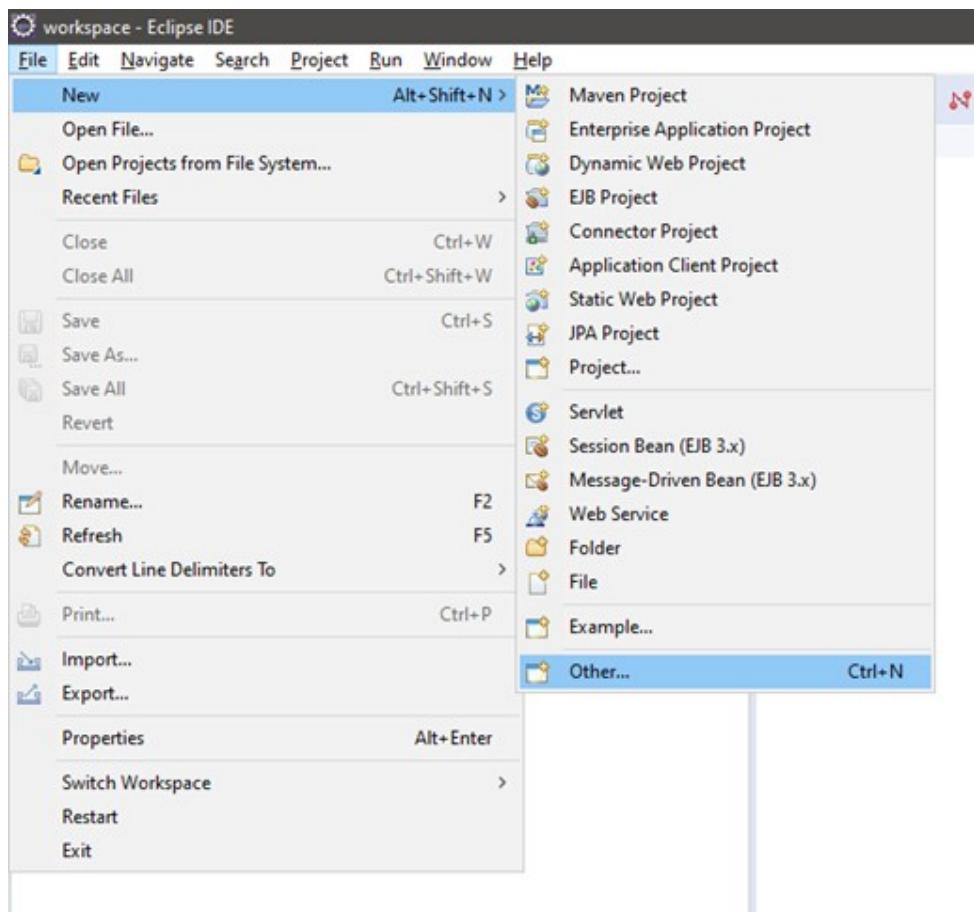
- Desactivar la casilla "Always show Welcome at start up" (en la parte inferior derecha de la pantalla) y pulsar la cruz de Welcome para cerrar la pestaña de bienvenida de Eclipse.

## Always show Welcome at start up

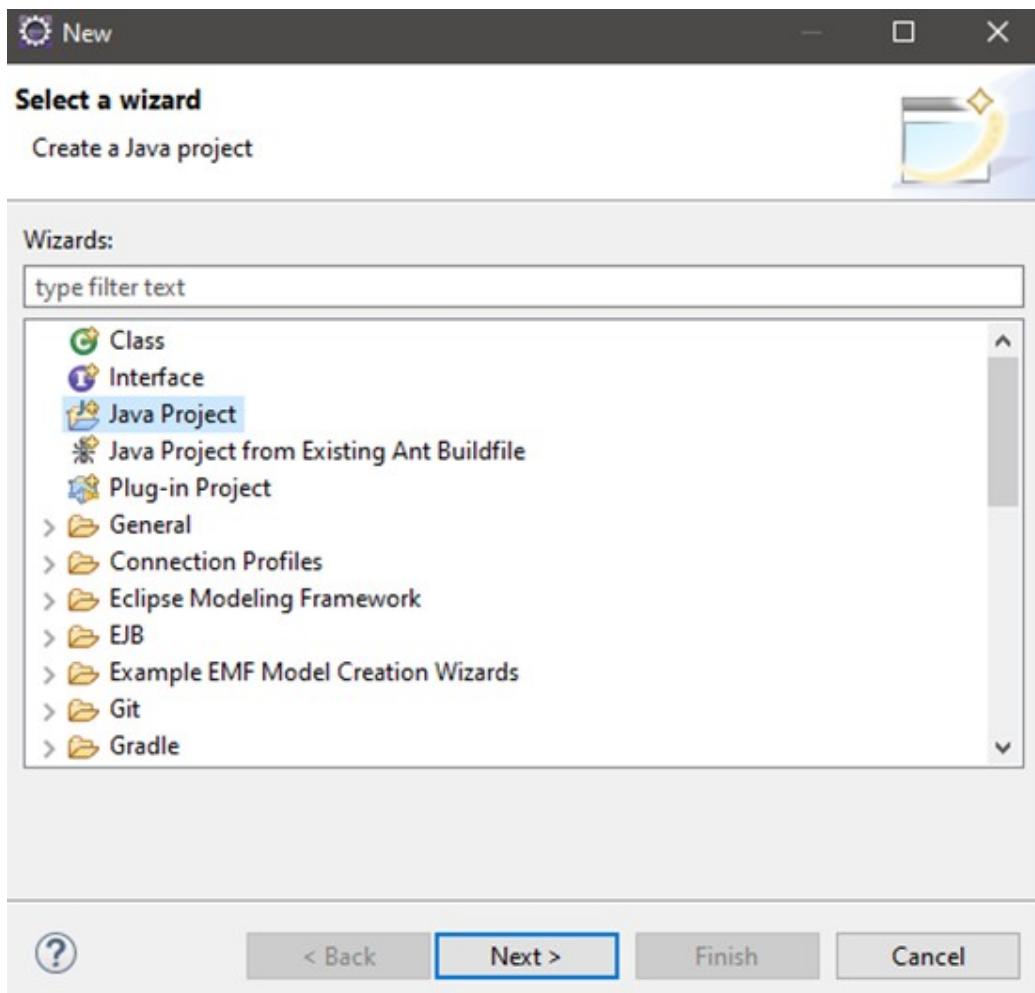


## Primer programa en Java

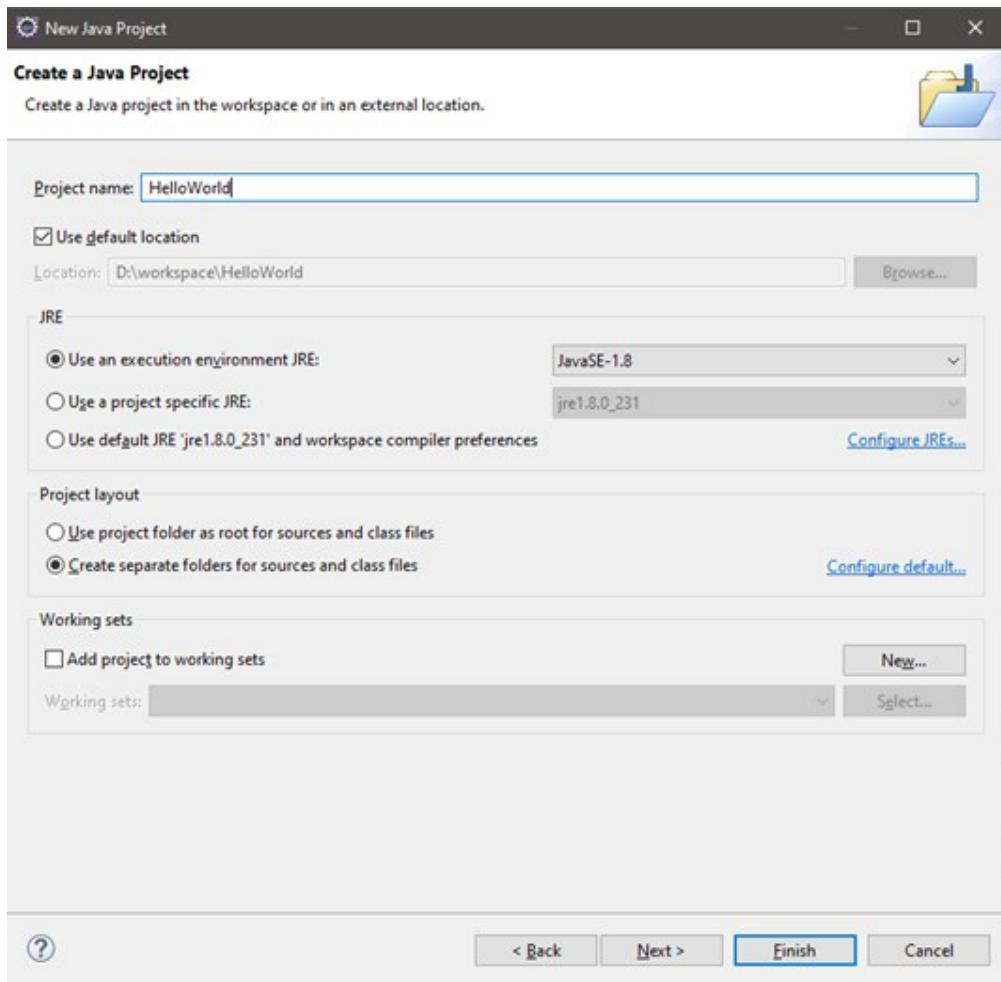
- Crear un nuevo Proyecto básico de Java.
  - *File -> New -> Others*



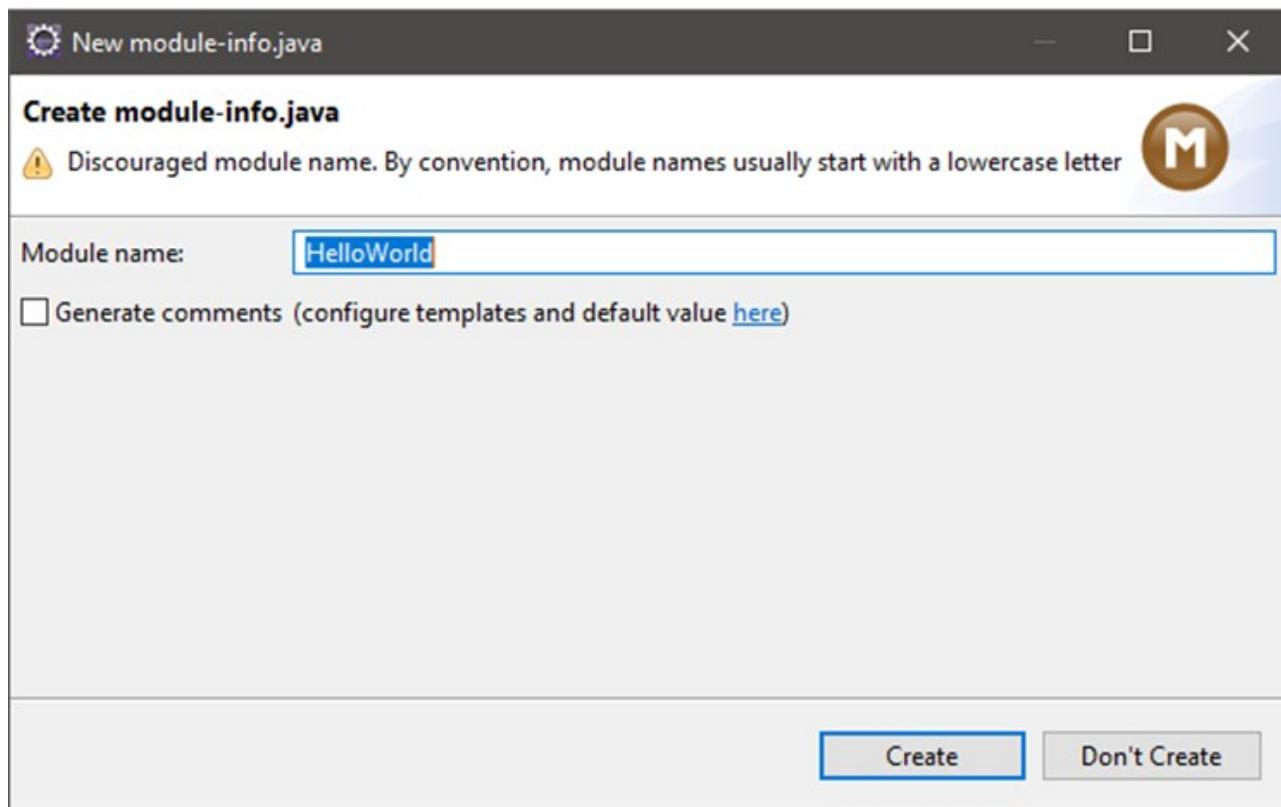
- Seleccionar Java Project.



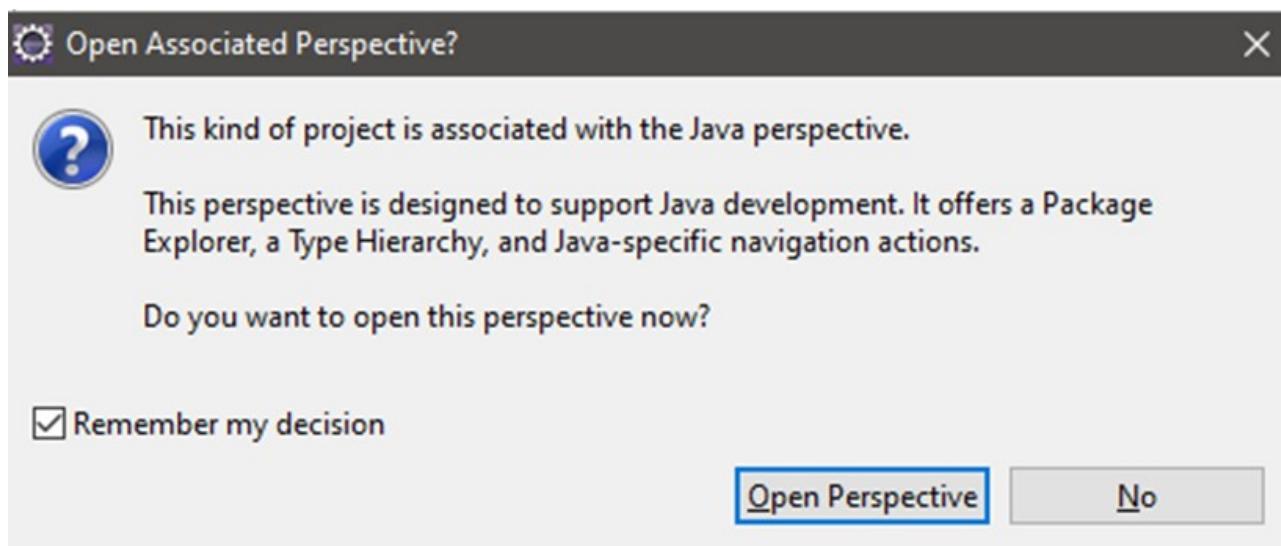
- Escribir el nombre del proyecto y finalizar.



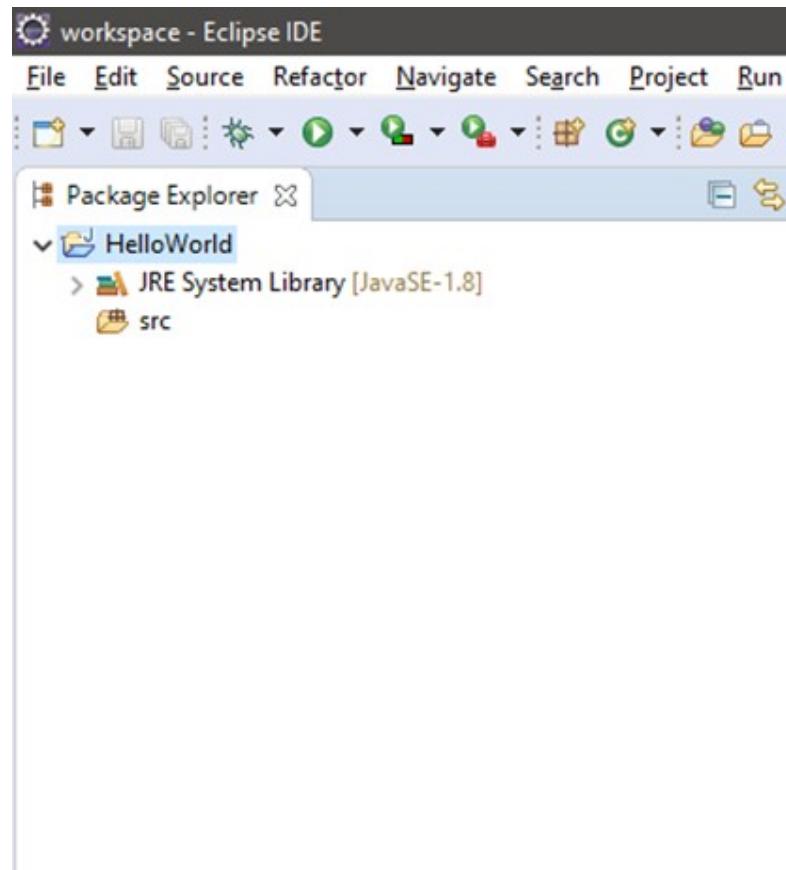
- ♦ Aparecerá una ventana para preguntar acerca de la creación de un módulo. Pulsar en *Don't Create*



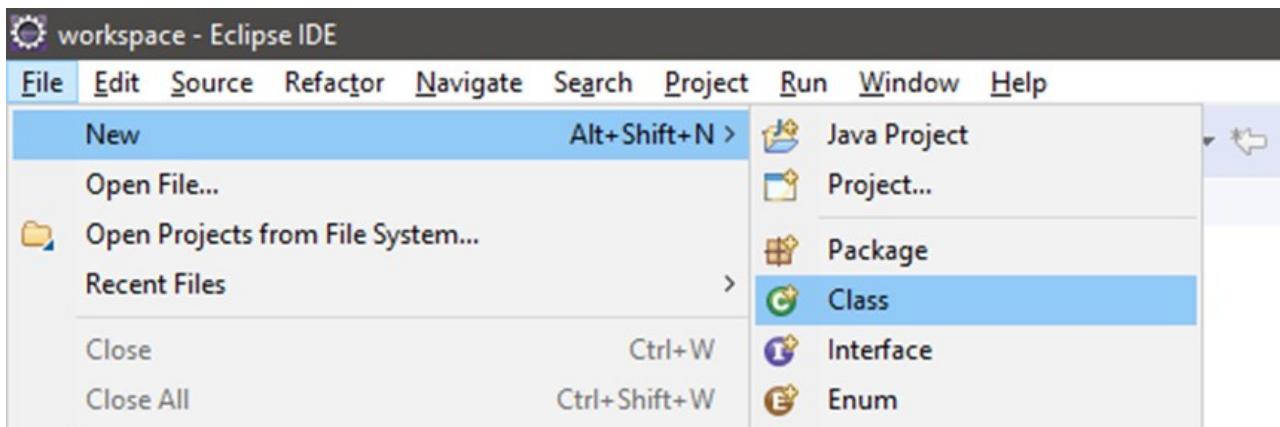
- Una nueva ventana aparecerá de nuevo. Hay que pulsar en “Open Perspective” (por defecto la perspectiva es la de Java EE).



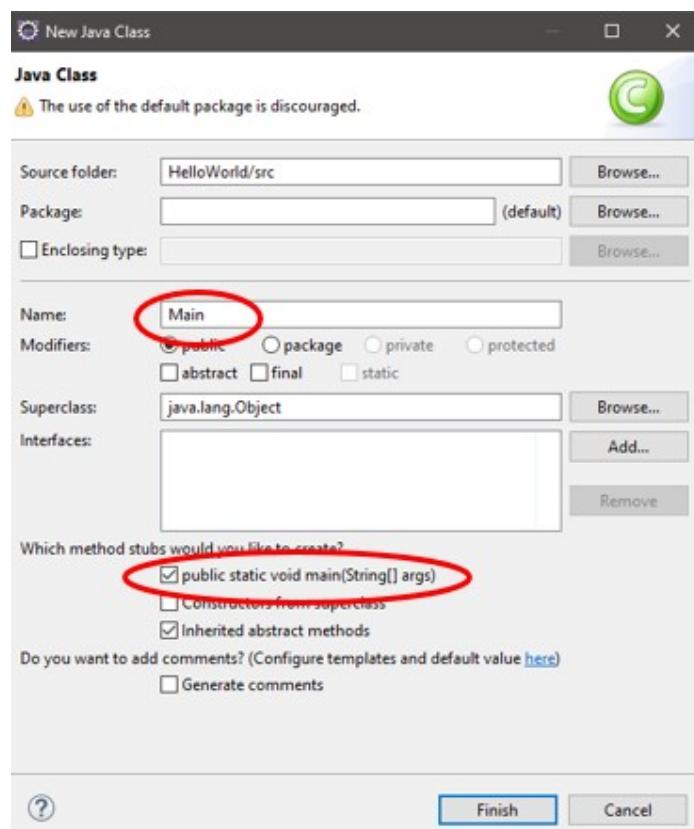
- En el explorador de paquetes aparece el nuevo proyecto. El directorio src contendrá todo los archivos de código fuente Java.



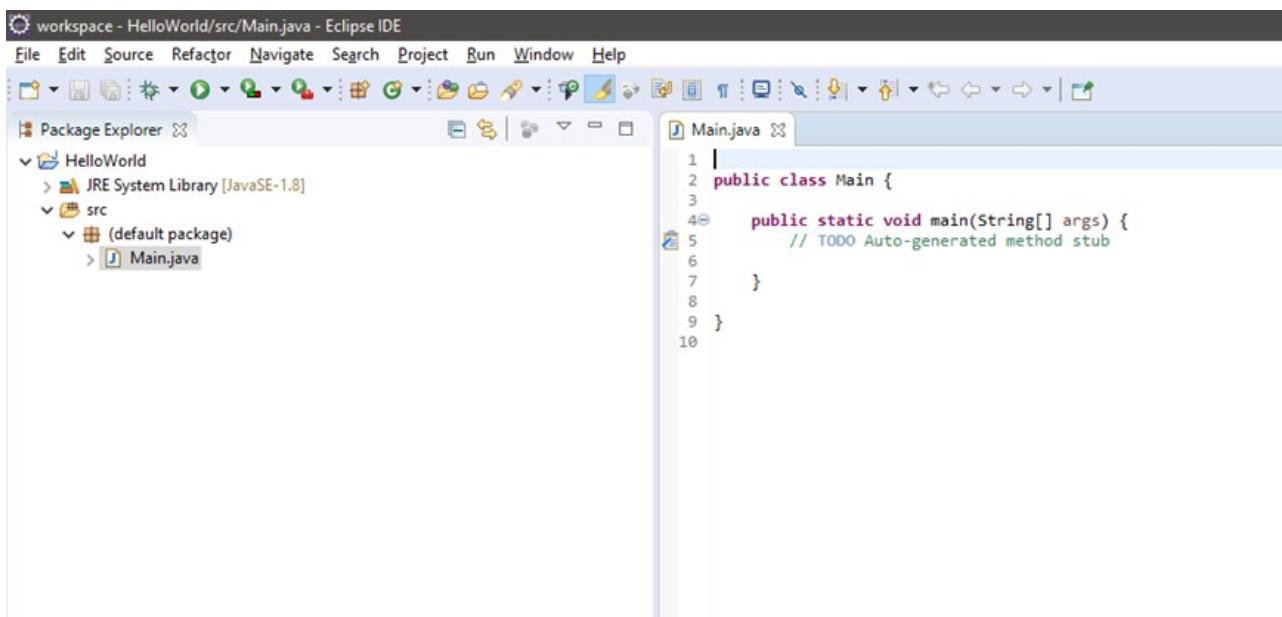
- Crear un nuevo archivo Java.
  - New -> Class



- 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.



- Puede ampliarse el tamaño de la letra desde:

- 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 output window displays the text: '<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0\_231\bin\javaw.exe (14 nov. 2019 19:29:00)' followed by the printed message '¡Hola mundo!'

- 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>
```

"El FSE invierte en tu futuro"  
Fondo Social Europeo

Tu carrera digital ~

# Módulo 3

## Java básico

### Fundamentos



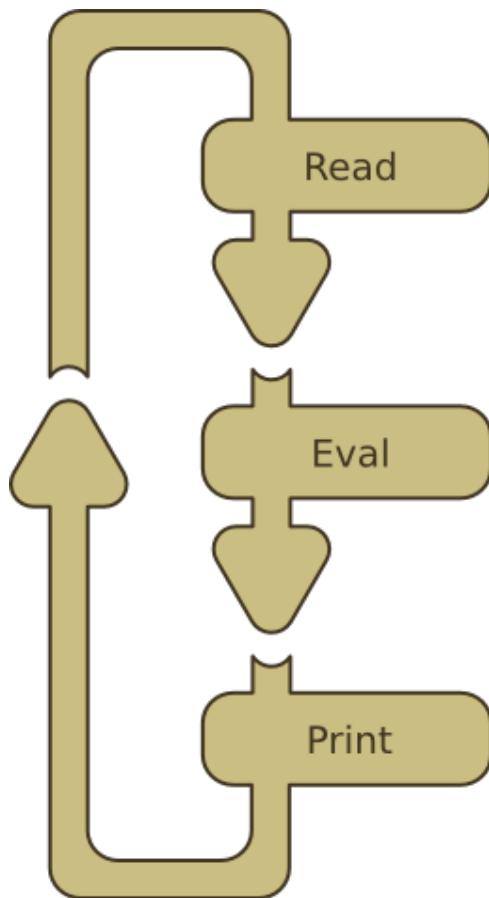
# Fundamentos

---

- ◆ 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>jshell
| 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 \$.

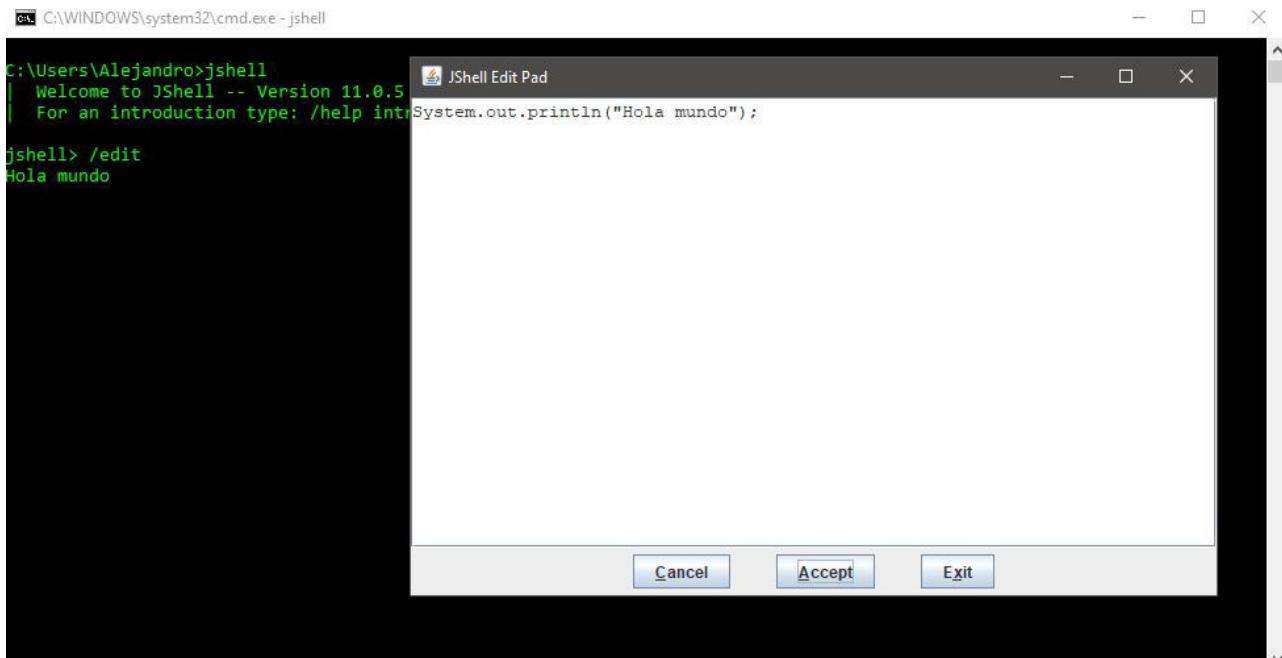
```
jshell> $7
$7 ==> 6

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.

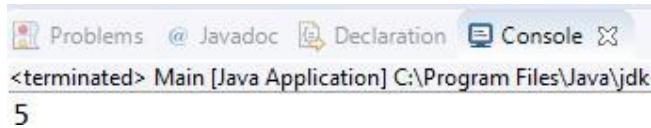
## System.out.println

- 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");
 }

}
```

Problems | @ Javadoc Declaration Console  
<terminated> Main [Java Application] C:\Program Files\Java\jdk-  
Hola mundo!  
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:
ingFormatException: 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.

| Tipo primitivo | Tipo de valores          | Tamaño (bits) | Rango de valores                                       | Ejemplo                |
|----------------|--------------------------|---------------|--------------------------------------------------------|------------------------|
| 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 */* 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 = 2222222222L;
```

- 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 = 222222222L;

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       |
|-------------------------|-------------------|
| <code>==</code>         | Igual que         |
| <code>&lt;</code>       | Menor que         |
| <code>&lt;=</code>      | Menor o igual que |
| <code>&gt;</code>       | Mayor             |
| <code>&gt;=</code>      | Mayor o igual que |
| <code>!=</code>         | Distinto que      |

| Operador de comparación | Significado       |
|-------------------------|-------------------|
| <code>==</code>         | Igual que         |
| <code>&lt;</code>       | Menor que         |
| <code>&lt;=</code>      | Menor o igual que |
| <code>&gt;</code>       | Mayor             |
| <code>&gt;=</code>      | Mayor o igual que |
| <code>!=</code>         | 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.

$4.3 >= 4$

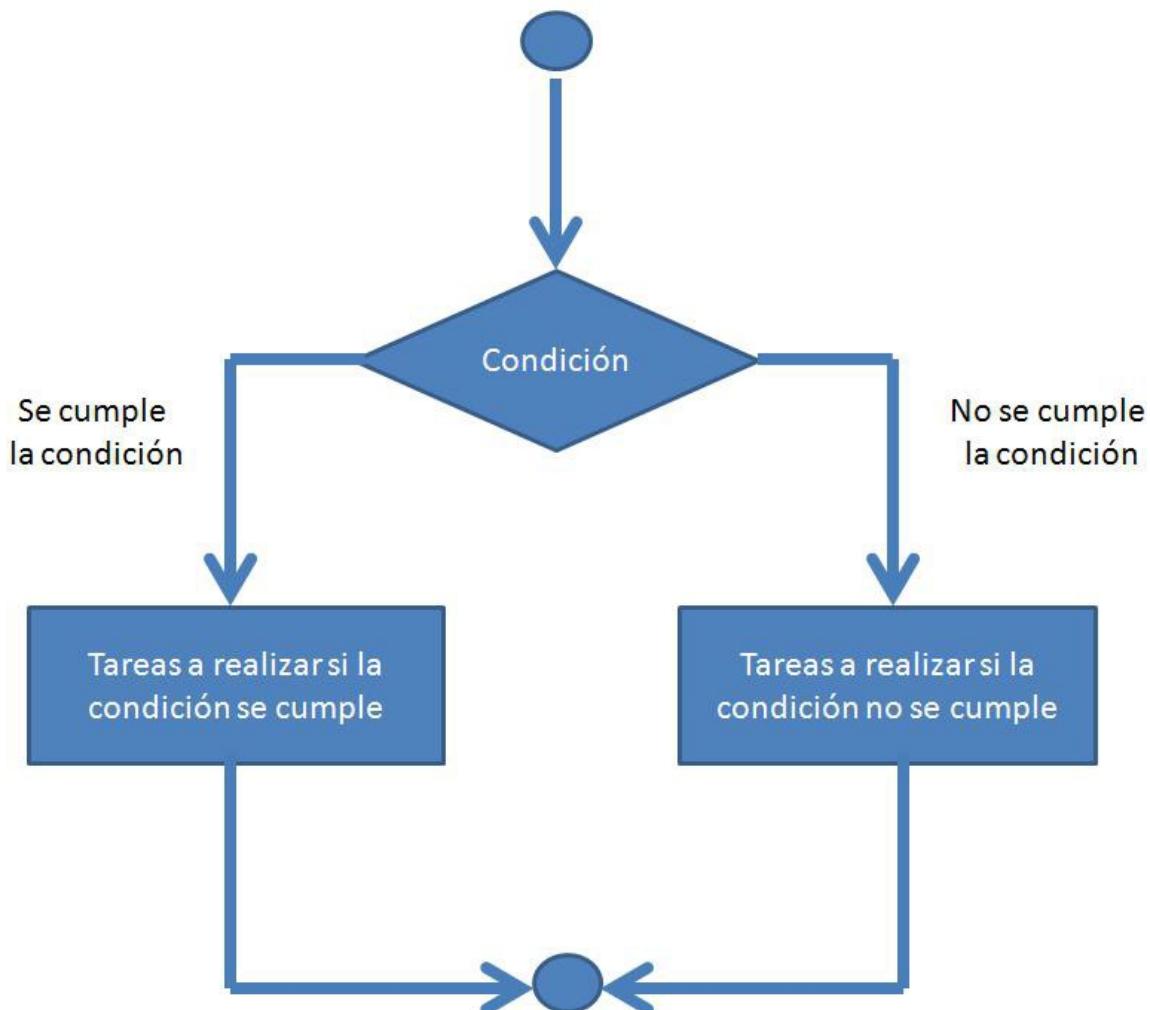
$1 == 2$

$4 < 4$

$2 != 5$

## 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.

| Operador lógico | Descripción | Ejemplo                  | Descripción del ejemplo                  |
|-----------------|-------------|--------------------------|------------------------------------------|
| &&              | 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 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

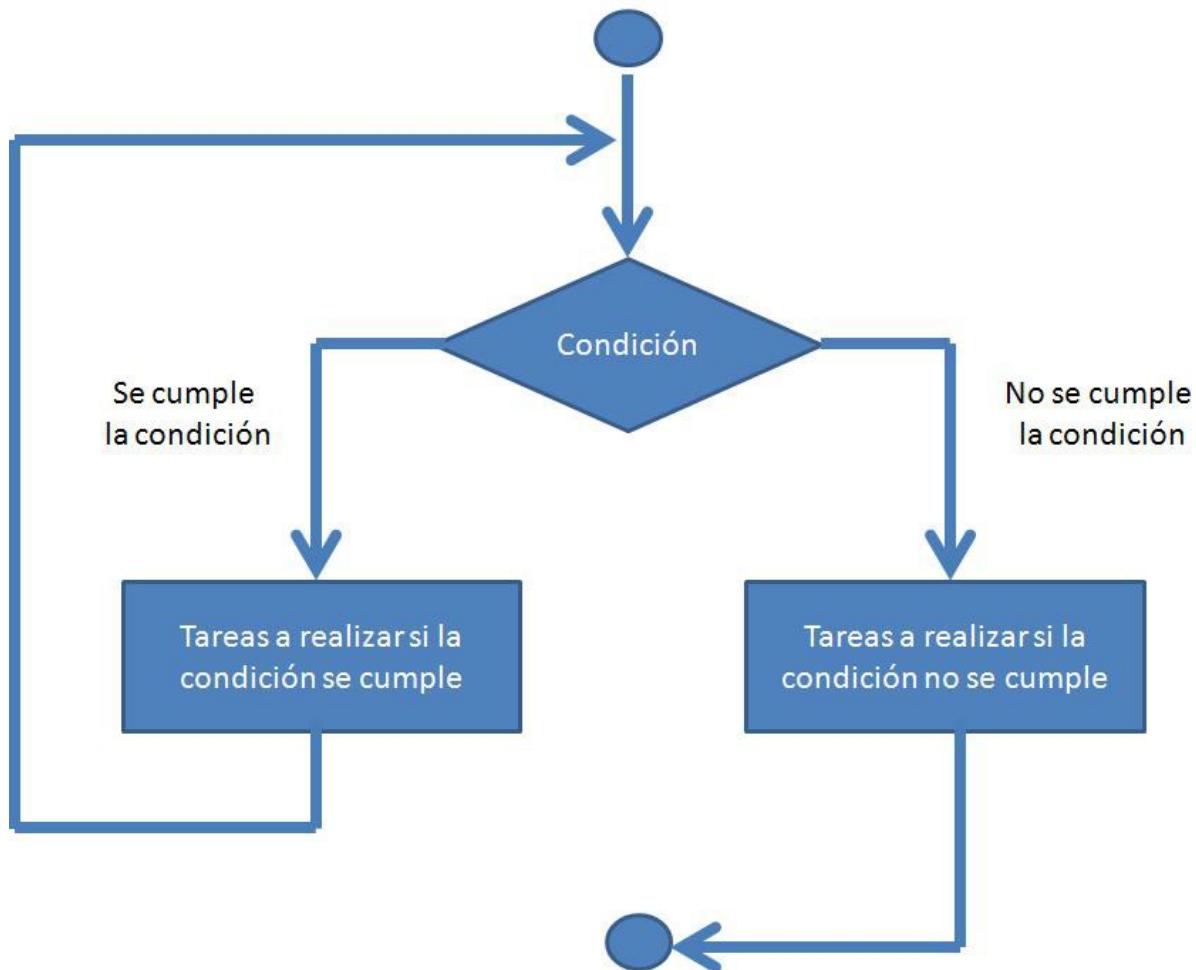
5 --> Préstamos

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 genera 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.ArithmetiException: / 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 (ArithmeticException 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.

| Abstracción de datos      | Abstracción funcional |
|---------------------------|-----------------------|
| 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



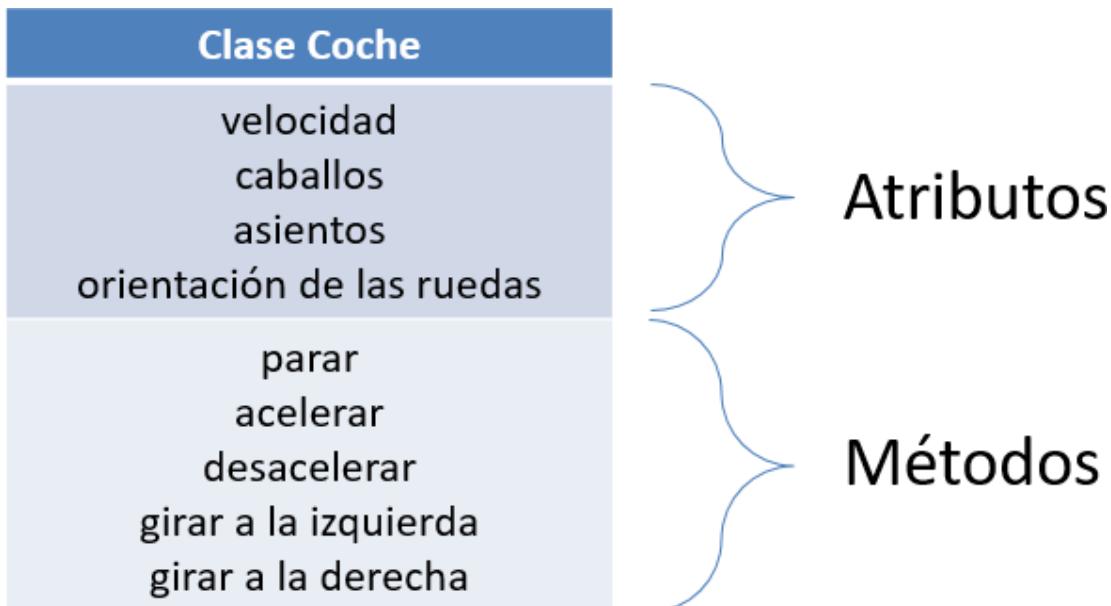
Atributos



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.



- 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;

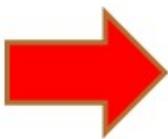
 // métodos
 void parar() { }
 void acelerar() { }
 void desacelerar() { }
 void girarIzquierda() { }
 void girarDerecha() { }
}
```

- Por tanto, una clase es una plantilla para definir objetos.



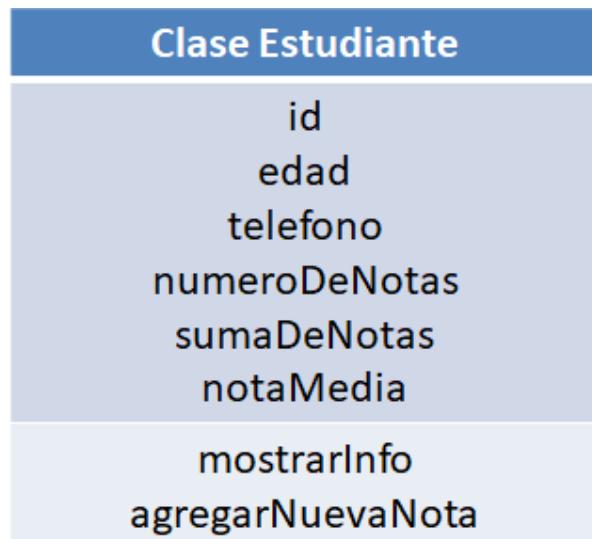
- A partir de la clase pueden crearse objetos como si fueran variables mediante la partícula new seguido del nombre de la clase y paréntesis de apertura y cierre.

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



## 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<br>caballos = 0.0<br>orientación de las ruedas = 0<br>marcha = 0 | velocidad = 0.0<br>caballos = 0.0<br>orientación de las ruedas = 0<br>marcha = 0 | velocidad = 0.0<br>caballos = 0.0<br>orientación de las ruedas = 0<br>marcha = 0 |
| parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>girar a la derecha   | parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>girar a la derecha   | parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>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<br>caballos = 350.0 CV<br>asientos = 2<br>orientación de las ruedas = 0º | velocidad = 0.0 km/h<br>caballos = 90.0 CV<br>asientos = 4<br>orientación de las ruedas = 0º | velocidad = 12.0 km/h<br>caballos = 75.0 CV<br>asientos = 4<br>orientación de las ruedas = 9º |
| parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>girar a la derecha                | parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>girar a la derecha               | parar<br>acelerar<br>desacelerar<br>girar a la izquierda<br>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());

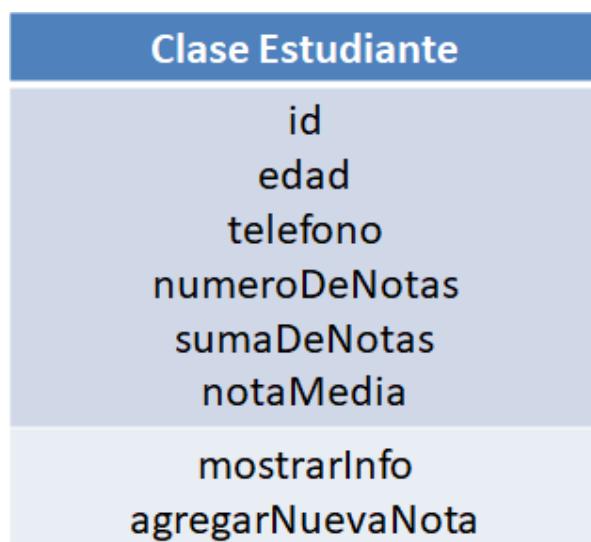
// se modifica el valor actual del atributo caballos a 250
audi.setCaballos(250.0f);

// 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.



## 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

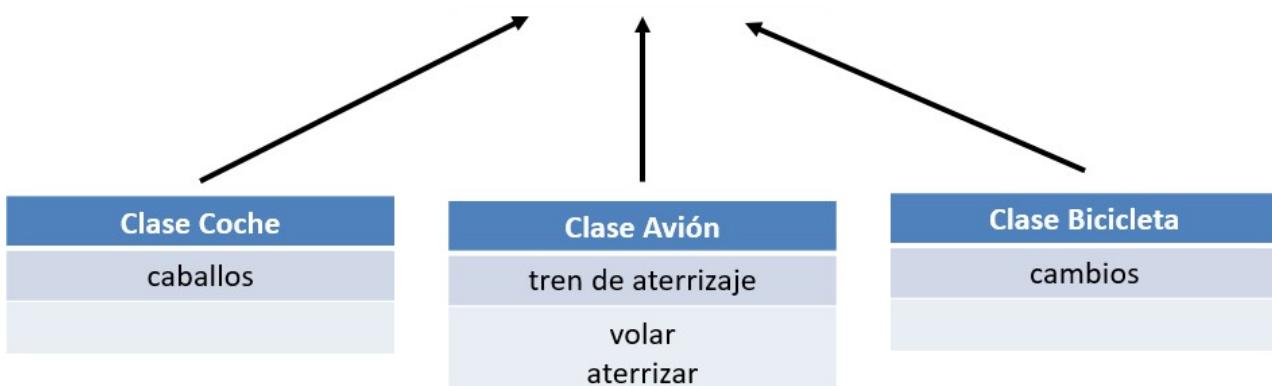
- A veces pueden existir clases que comparten características comunes.
- La herencia permite definir una clase a partir de otra relacionada.
- Es un mecanismo para la reutilización de software.

| Clase Coche                                                                                  | Clase Avión                                                                                               | Clase Bicicleta                                                                              |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| velocidad<br>caballos<br>asientos<br><b>orientación de las ruedas</b>                        | velocidad<br>tren de aterrizaje<br>asientos<br><b>orientación de las ruedas</b>                           | velocidad<br>cambios<br>asientos<br><b>orientación de las ruedas</b>                         |
| parar<br>acelerar<br>desacelerar<br><b>girar a la izquierda</b><br><b>girar a la derecha</b> | volar<br>aterrizar<br>acelerar<br>desacelerar<br><b>girar a la izquierda</b><br><b>girar a la derecha</b> | parar<br>acelerar<br>desacelerar<br><b>girar a la izquierda</b><br><b>girar a la derecha</b> |

Las clases Coche, Avión y Bicicleta son especializaciones de la clase Vehículo

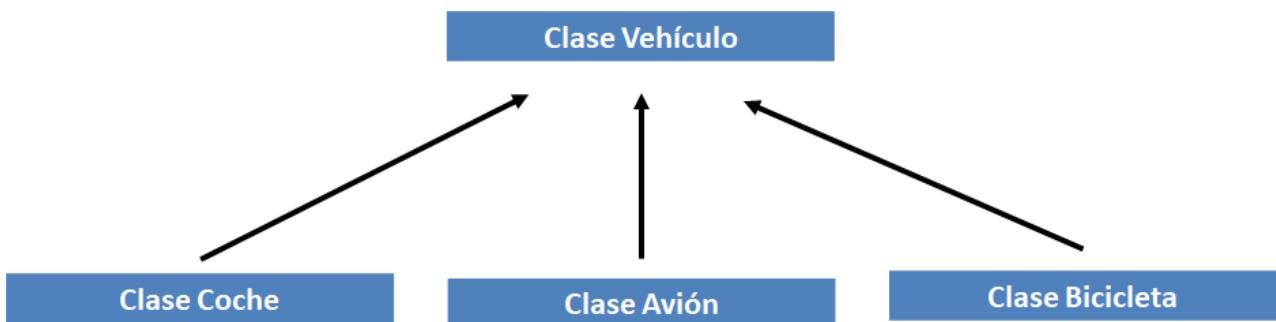
| Clase Vehículo              |
|-----------------------------|
| velocidad                   |
| asientos                    |
| orientación de las ruedas   |
| parar                       |
| acelerar                    |
| desacelerar                 |
| <b>girar a la izquierda</b> |
| <b>girar a la derecha</b>   |

Las clase Vehículo es una generalización de las clases Coche, Avión y Bicicleta.

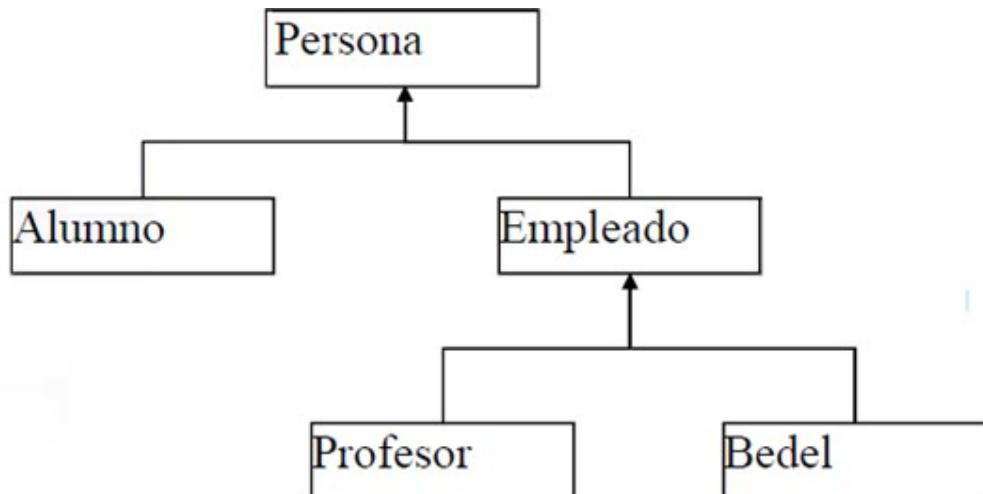


- La terminología utilizada en la herencia es la siguiente:
  - Coche, Avión y Bicicleta heredan los atributos y métodos de la clase Vehículo.
  - Coche, Avión y Bicicleta son subclases, clases derivadas o clases hijas de la clase Vehículo.

- 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        |

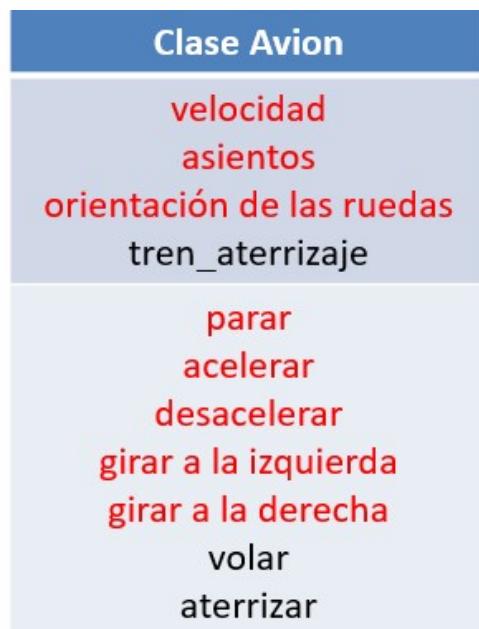
- Una vez declarada una clase hija, ésta hereda todos los atributos y métodos de la superclase.
- Cualquier cambio en el código en la superclase, afectará a la clase hija.

## 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;

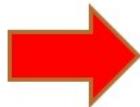
 public Coche(float velocidad, float caballos, int asientos, int orientacionRuedas) {

 // 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");
 }
}
```

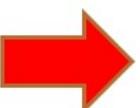
Fichero Main.java



```
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");
 }
}
```

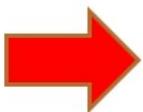
Fichero Main.java



```
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



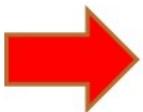
```
public class Vehiculo {
 public Vehiculo(float variable) {
 System.out.println("Constructor de la clase padre");
 }
}

public class Bicicleta extends Vehiculo {
 public Bicicleta() {
 super();
 System.out.println("Constructor de la clase hija");
 }
}

public class Main {
 public static void main(String[] args) {
 Bicicleta bicicleta = new Bicicleta();
 }
}
```

Error

Fichero Bicicleta.java



Fichero Main.java



## 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;

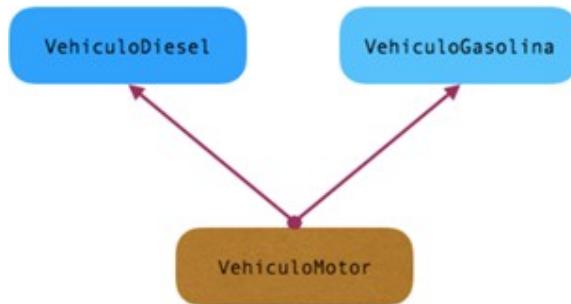
 // reescritura del método
 public void parar() {

 // invocación al método de la superclase
 super.parar();
 }
}

```

## Herencia múltiple

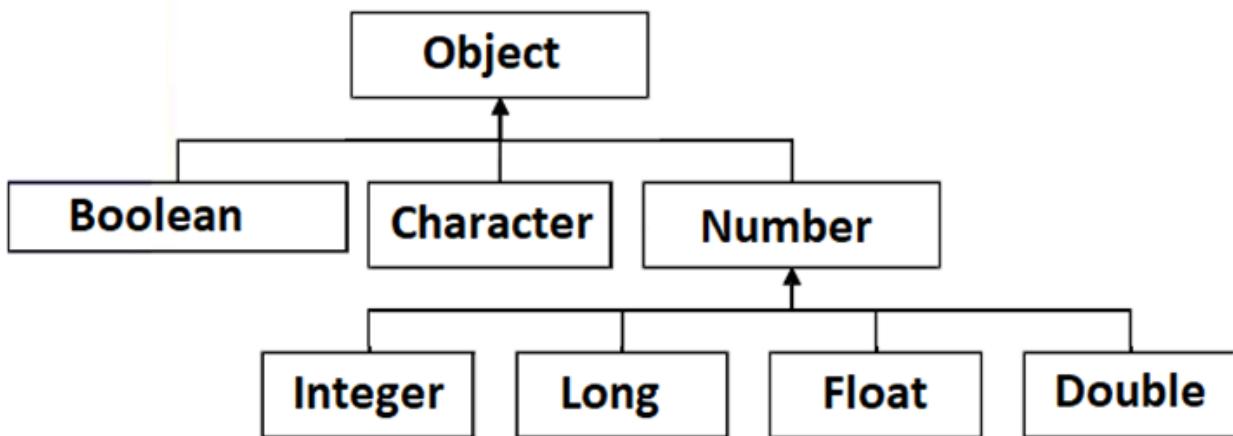
- La herencia múltiple (una clase hereda de varias superclases) no es permitida en Java por tres razones:
  - Ambigüedad en el código.
  - Mayor complejidad.
  - Problemas en la compilación de código.



- 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 las 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<br>edad<br>telefono<br><br>mostrarInfo | numeroDeNotas<br>sumaDeNotas<br>notaMedia<br><br>agregarNuevaNota*3<br>mostrarInfo+<br><i>crearEstudiante</i> | numeroDeEdadesEstudiantes<br>sumaDeEdadesEstudiantes<br>estudiantesEdadMedia<br><br>agregarEdadEstudiante<br>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();
}

```

Problems | @ Javadoc Declaration Console X  
<terminated> Main [Java Application] C:\Program Files\Java\j  
Método parar del vehículo  
Método parar del coche  
Método parar del avión

## 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.

```
1 public class Bicicleta extends Vehiculo {
2
3 }
4
5
```

Add unimplemented methods  
Create new JUnit test case for 'Bicicleta.java'  
Make type 'Bicicleta' abstract  
Rename in file (Ctrl+2, R)  
Rename in workspace (Alt+Shift+R)

```
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.

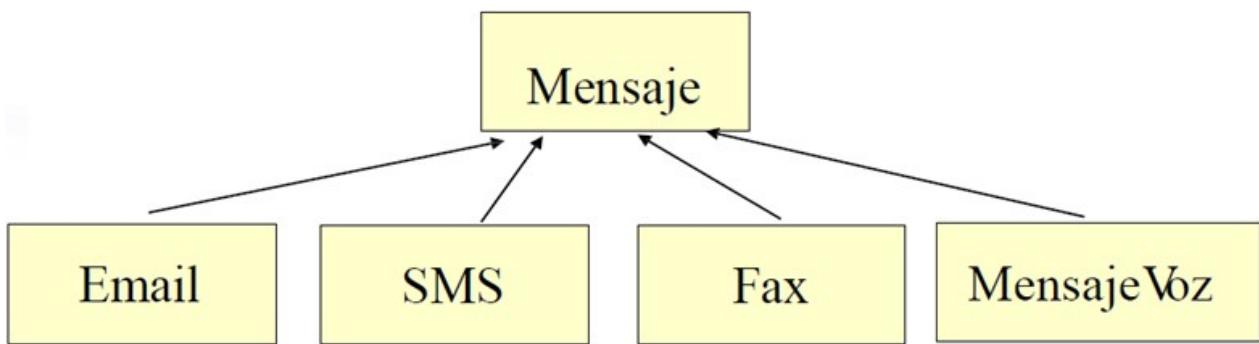
```
public abstract class Coche {

 // 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

 }

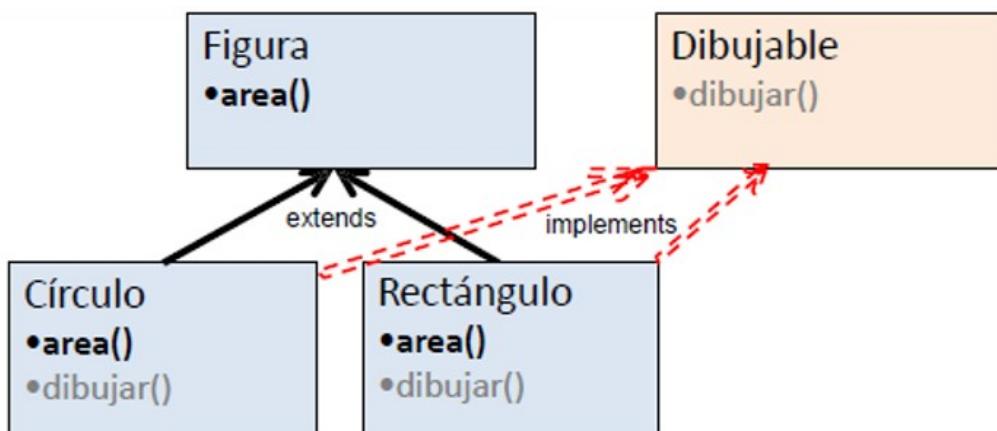
}
```

- 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.



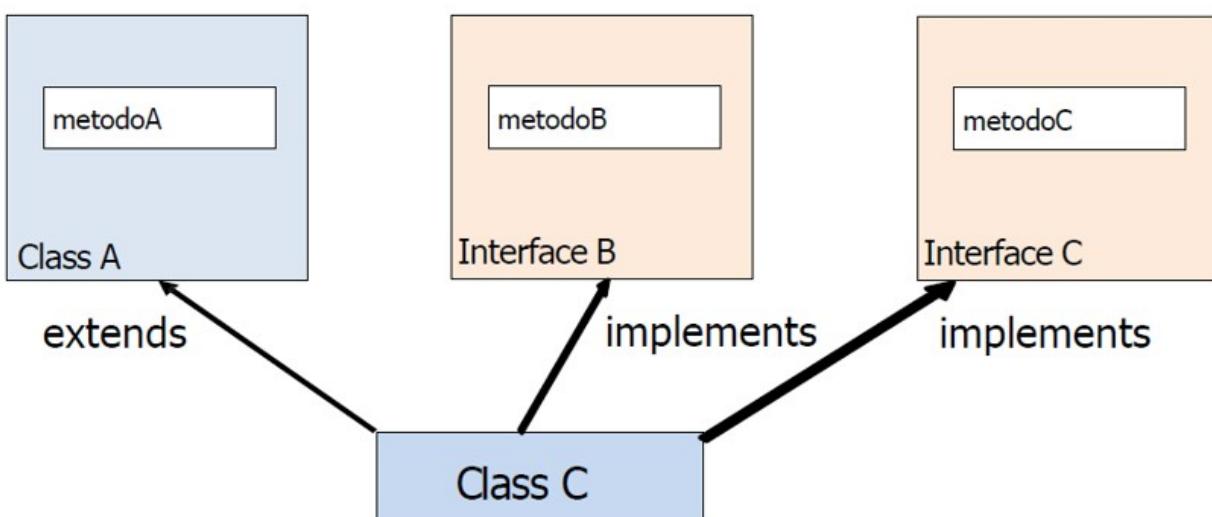
```
public abstract class Figura {...}
```

```
public interface Dibujable {...}
```

```
public class Circulo extends Figura implements Dibujable
```

```
public class Rectangulo 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 {

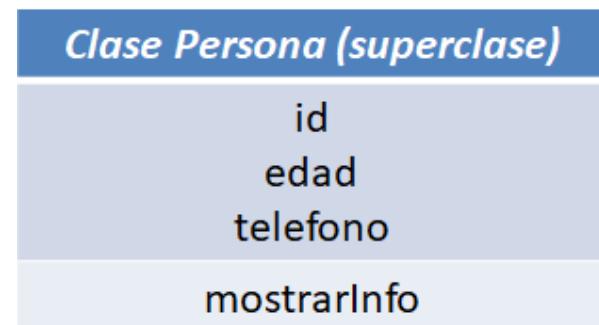
 @Override
 public void metodoC() {
 // TODO Auto-generated method stub
 }

 @Override
 public void metodoB() {
 // TODO Auto-generated method stub
 }
}
```

- ◆ En conclusión:

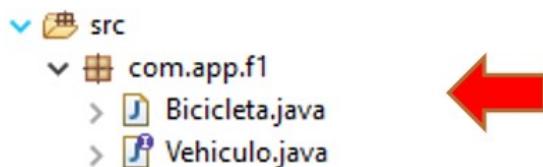
- Clase: todos los métodos se encuentran implementados.
- Clase abstracta: declarada con el modificador abstract y generalmente contiene al menos un método no implementado (abstracto)
- Interfaz: declarada con el modificador interface y no debe contener ningún método implementado (todos son abstractos).

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.

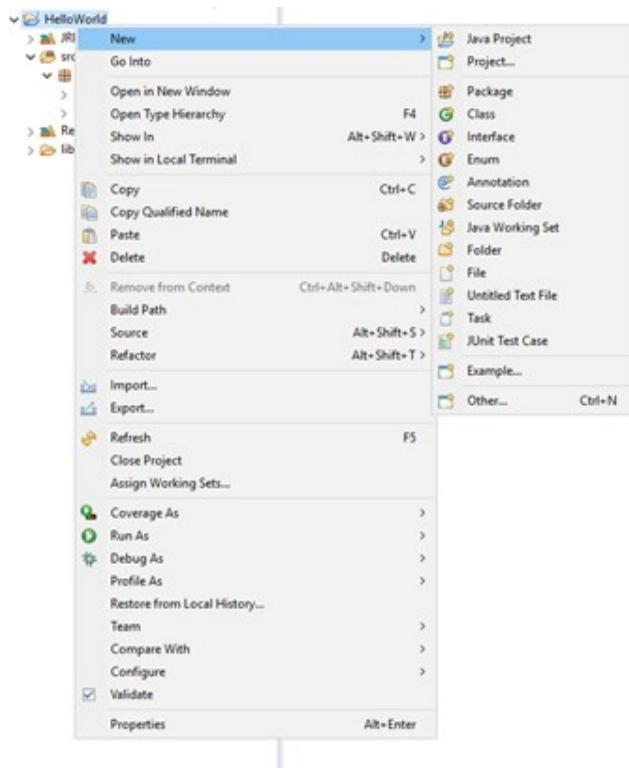


Ejemplo de un paquete llamado com.app.f1 que engloba a dos clases (Bicicleta y Vehiculo)

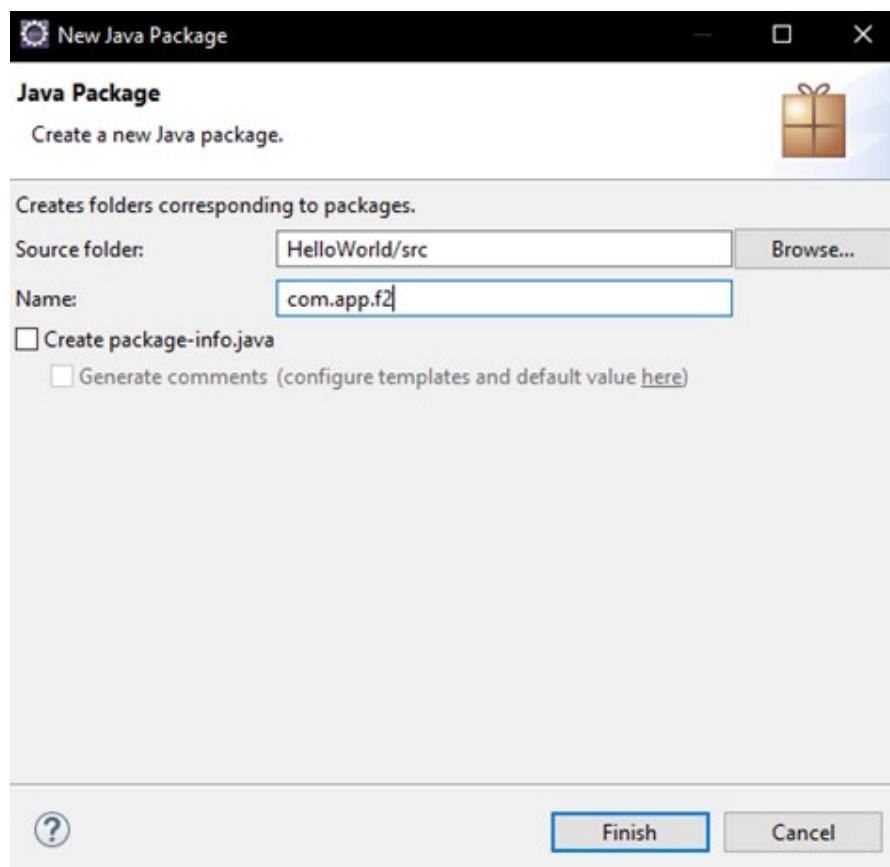
Los archivos del paquete se encuentran contenidos dentro del directorio src, donde la jerarquía del paquete corresponde con la estructura de subdirectorios



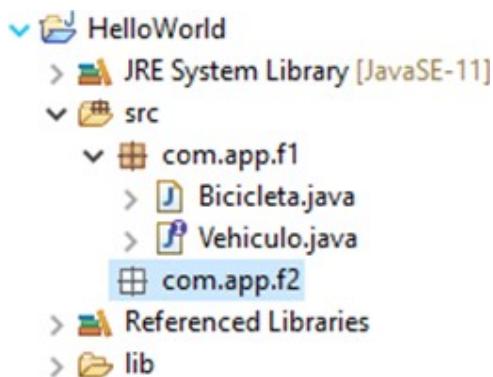
- Puede crearse un paquete para un proyecto en Eclipse pulsando el botón derecho del ratón sobre el nombre del Proyecto y, a continuación, accediendo a New -> Package.
- Eclipse creará automáticamente toda la estructura de directorios asociada al paquete.



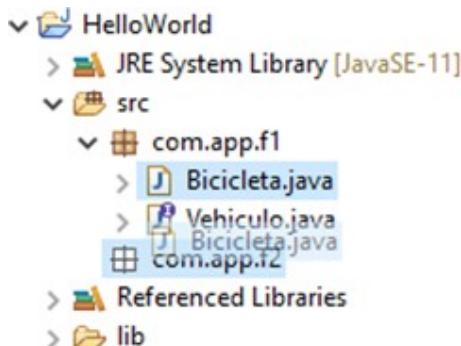
- Posteriormente se especifica el nombre del paquete, utilizando la notación punto.



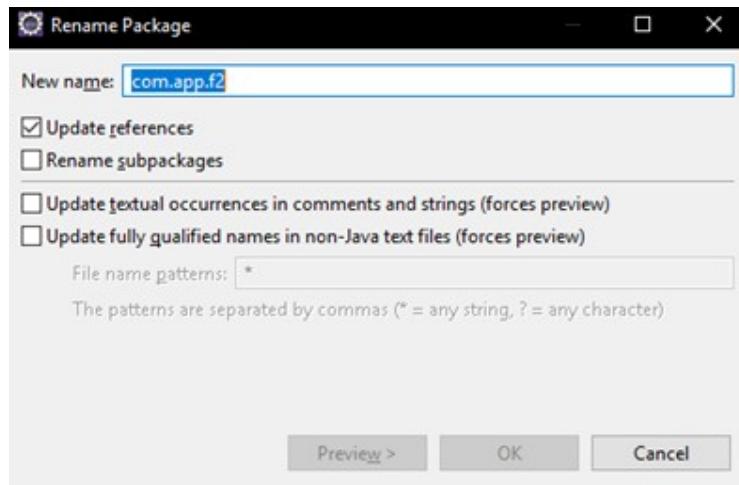
- El nuevo paquete se encuentra vacío tras su creación.



- 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);
 }
}

```

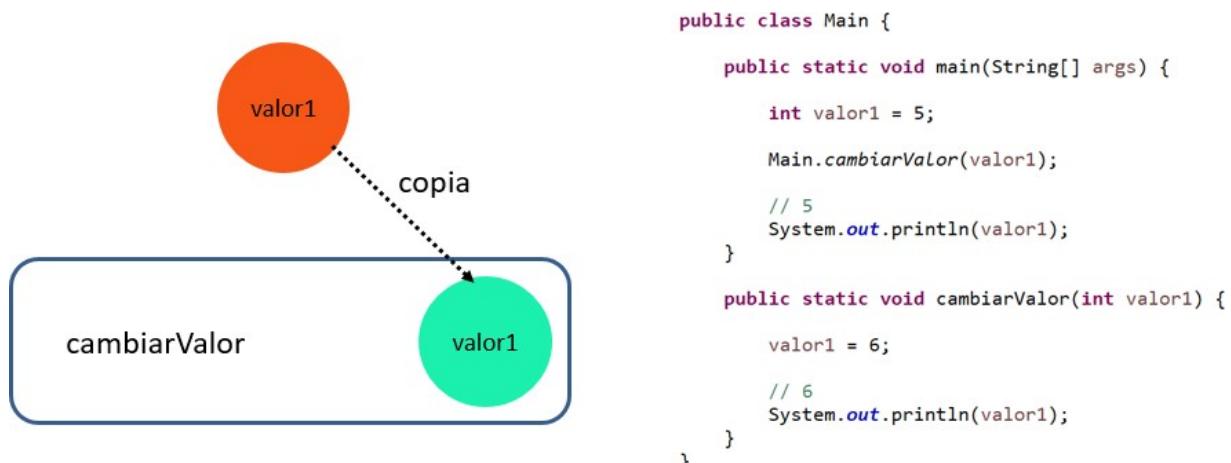
- El uso de paquetes en Java proporciona las siguientes ventajas:
  - Agrupamiento de clases con características comunes.
  - Reutilización de código al promover principios de programación orientada a objetos como la encapsulación y la modularidad.
  - Mayor seguridad al existir niveles de acceso.
  - 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.

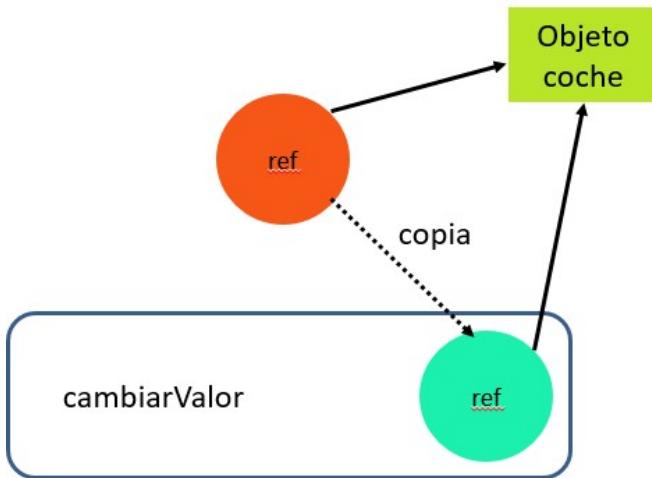
Ejercicio: Crea un nuevo paquete llamado com.clase.info y agrega todos los archivos con las clases e interfaces creadas hasta el momento: PersonaInterfaz.java, Persona.java, Estudiante.java y Profesor.java

## 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.



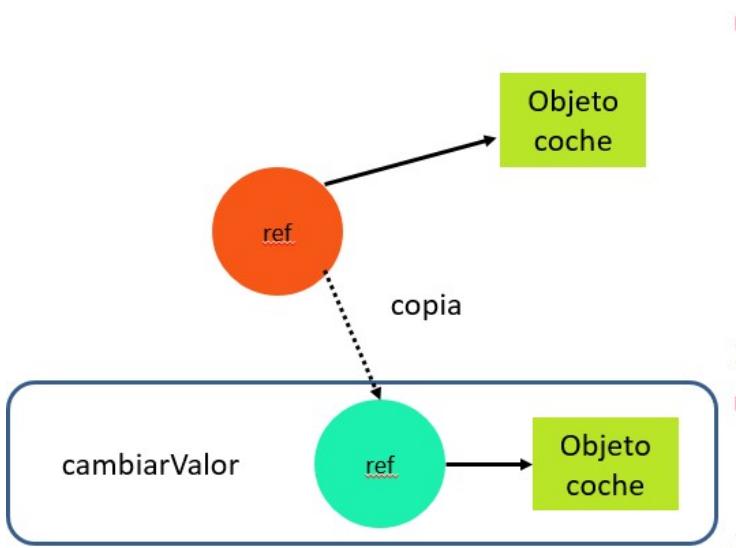
- Las objetos se almacenan en un espacio de la memoria como una dirección de memoria que apunta a los datos del objeto.
- Cuando un objeto se pasa por argumentos a un método, se hace una copia de su referencia (dirección de memoria).



```
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 3

## Java básico

### Strings



# Strings

- ◆ Introducción
- ◆ Métodos de String
- ◆ Concatenación
- ◆ Inmutabilidad
- ◆ StringBuffer y StringBuilder
- ◆ Métodos de StringBuilder y StringBuffer

## Introducción

- ◆ El tipo primitivo char permite almacenar un carácter en una variable.

Ejercicio: escribe un programa que muestre por pantalla los caracteres individuales de una determinada palabra utilizando variables de tipo char.

- ◆ Se utiliza El tipo String se utiliza para almacenar cadenas de caracteres o texto.
- ◆ La clase String es parte del paquete java.lang de Java. Provee un conjunto de métodos para procesamiento de texto.
- ◆ Existen dos formas de inicializar una variable de tipo String: mediante el operador de asignación (=) y creando una instancia de la clase String.

```
String nombre1 = "Alejandro";
System.out.println(nombre1);

// otra forma de declarar un String
String nombre2 = new String("Marcos");
System.out.println(nombre2);
```

Ejercicio proyecto (Main5): crea la clase Gestor a partir del modelo de datos suministrado y crea varios objetos desde la clase principal Main5. Mueve los archivos de las dos clases al paquete com.banco

Ejercicio proyecto (Main6): reutiliza el programa de la clase Main4 para añadir el siguiente menú cuando se seleccione la opción de Gestores.

- ```
---
1. Añadir gestor
2. Modificar gestor
3. Eliminar gestor
4. Ver gestor
5. Ver gestores
```

6. Atrás

Introduzca un número:

Métodos de String

- `length()`: retorna la longitud del String.

```
String texto = "Esto es un texto";
System.out.println(texto.length());
```

- `charAt(int index)`: devuelve el carácter de una determinada posición o índice del String pasado por argumentos (`index`), donde el índice 0 es el primer carácter del String.

```
String texto = "Esto es un texto";

// devuelve el primer carácter del String
System.out.println(texto.charAt(0));

// devuelve el tercer carácter del String
System.out.println(texto.charAt(2));

// devuelve el último carácter del String
System.out.println(texto.charAt(texto.length() - 1));

// error porque el String texto posee una longitud de 16 caracteres. Se genera un
// excepción de tipo StringIndexOutOfBoundsException
// System.out.println(texto.charAt(222));
```

- `indexOf`: tiene dos posibles formas de uso:
 - `indexOf(char ch, int index)`: busca un determinado carácter en el String y retorna el índice de la primera coincidencia o -1 si no existe coincidencia. Un segundo parámetro opcional permite buscar a partir de un determinado índice del String.
 - `indexOf(String str, int index)`: busca una determinada cadena de texto en el String y retorna el índice inicial de la primera coincidencia o -1 si no existe coincidencia. Un segundo parámetro opcional permite buscar a partir de un determinado índice del String.

```
String texto = "Esto es un texto";

// 3
System.out.println(texto.indexOf('o'));

// -1
System.out.println(texto.indexOf('z'));
```

```
// 5
System.out.println(texto.indexOf("es"));

// -1
System.out.println(texto.indexOf("a"));

// 15
System.out.println(texto.indexOf('o', 6));

// -1
System.out.println(texto.indexOf("es", 6));
```

- `lastIndexOf`: tiene dos posibles formas de uso:
 - `lastIndexOf(char ch)`: busca un determinado carácter en el String y retorna el índice de la última coincidencia. Retorna -1 si no existe coincidencia.
 - `lastIndexOf(String str)`: busca una determinada cadena de texto en el String y retorna el índice inicial de la última coincidencia. Retorna -1 si no existe coincidencia.

```
String texto = "Esto es un texto";

// 15
System.out.println(texto.lastIndexOf('o'));

// -1
System.out.println(texto.lastIndexOf('z'));

// 5
System.out.println(texto.lastIndexOf("es"));

// -1
System.out.println(texto.lastIndexOf("a"));
```

- `startsWith(String str)`: retorna true si el String comienza con una cadena de texto pasada por argumentos, o false en caso contrario.

```
String texto = "Esto es un texto";

// true
System.out.println(texto.startsWith("Esto"));

// false
System.out.println(texto.startsWith("a"));
```

- `endsWith(String str)`: retorna true si el String finaliza con una cadena de texto pasada por argumentos, o false en caso contrario.

```
String texto = "Esto es un texto";  
  
// true  
System.out.println(texto.endsWith("to"));  
  
// false  
System.out.println(texto.endsWith("a"));
```

- `isEmpty()`: retorna true si el String está vacío, o false en caso contrario.

```
String texto = "";  
  
// true  
System.out.println(texto.isEmpty());
```

- `equals(String str)`: retorna true si el String es idéntico a una cadena de texto pasada por argumentos. En caso de no coincidencia, devuelve false. La comparación entre dos variables de tipo String no debe realizarse nunca con el operador de comparación ==

```
String texto1 = "hola";  
String texto2 = "adiós";  
  
// true  
System.out.println(texto1.equals("hola"));  
  
// false  
System.out.println(texto2.equals("hola"));  
  
// false porque el primer carácter se encuentra en mayúsculas  
System.out.println(texto1.equals("Hola"));
```

- `equalsIgnoreCase(String str)`: retorna true si el String es idéntico a una cadena de texto pasada por argumentos, pero ignorando las mayúsculas y minúsculas. En caso de no coincidencia, devuelve false.

```
String texto1 = "hola";  
String texto2 = "AdIóS";  
  
// true  
System.out.println(texto1.equalsIgnoreCase("hOlA"));
```

```
// true
System.out.println(texto2.equalsIgnoreCase("adiós"));
```

- `contains(String str)`: retorna true si el String contiene una cadena de texto pasada por argumentos, o false en caso contrario.

```
String texto = "hola";
// true
System.out.println(texto.contains("ol"));
// false
System.out.println(texto.equals("al"));
```

- `substring`: retorna un substring del String. Tiene dos posibles formas de uso:
 - `substring(int beginIndex)`: el substring comienza en el índice beginIndex del String y se extiende hasta el final del mismo.
 - `substring(int beginIndex, int endIndex)`: el substring comienza en el índice beginIndex del String y se extiende hasta el índice endIndex - 1 del mismo.

```
String texto = "Hola, cómo estás";
// cómo estás
System.out.println(texto.substring(6));
// cómo
System.out.println(texto.substring(6, 10));
```

- `toUpperCase()`: retorna el String con todas las letras en mayúsculas.

```
String texto = "hello";
// HELLO
System.out.println(texto.toUpperCase());
```

- `toLowerCase()`: retorna el String con todas las letras en minúsculas.

```
String texto = "HELLO";
// hello
System.out.println(texto.toLowerCase());
```

- concat(String str): retorna la unión entre el String y una cadena de texto.

```
String texto = "Esto es un texto";  
  
// Esto es un texto más grande  
System.out.println(texto.concat(" más grande"));
```

- replace(String str1, String str2): retorna el resultado de reemplazar todas las ocurrencias de una cadena de texto en el String, con otra cadena de texto distinta.

```
String texto = "En un lugar de la Mancha";  
  
// En un lugur de lu Munchu  
System.out.println(texto.replace("a", "u"));
```

- replaceFirst(String str1, String str2): retorna el resultado de reemplazar la primera ocurrencia de una cadena de texto en el String, con otra cadena de texto distinta.

```
String texto = "En un lugar de la Mancha";  
  
// En un lugur de la Mancha  
System.out.println(texto.replaceFirst("a", "u"));
```

- trim(): retorna el String eliminando espacios en blanco en ambos extremos. No afecta los espacios en blanco restantes.

```
String texto = " En un lugar de la Mancha ";  
  
// "En un lugar de la Mancha"  
System.out.println(texto.trim());
```

- String.valueOf(): es un método estático de String que permite convertir una variable de un tipo concreto, a String.

```
// convierte de booleano a String  
String str1 = String.valueOf(true);  
System.out.println(str1);  
  
// convierte de int a String  
String str2 = String.valueOf(1);
```

```
System.out.println(str2);

// convierte de char a String
String str3 = String.valueOf('a');
System.out.println(str3);
```

Ejercicio: escribe un programa que dado dos String compruebe si los dos primeros caracteres son iguales.

Ejercicio: escribe un programa que dado dos String compruebe si los dos primeros caracteres y los dos últimos son iguales.

Ejercicio: escribe un programa que compruebe que el substring "abc" se encuentra en un String dado, pero no puede encontrarse ni al comienzo, ni al final.

Ejercicio: escribe un programa que dado un String y un determinado índice, compruebe que el carácter anterior y el posterior son iguales o no.

Ejercicio: escribe un programa que muestra por pantalla el índice de la segunda y tercera ocurrencia del carácter 'a' en un String dado.

Ejercicio: escribe un programa que devuelva si un String es palíndromo (se escribe igual hacia delante y hacia detrás). Ejemplo: "sometemos".

Concatenación

- El símbolo + es utilizado con variables de tipo numéricas (enteros y flotantes) para realizar sumas.
- Sin embargo, cuando al menos uno de los operandos es una variable de tipo String, el símbolo + actúa como una concatenación.
- Por tanto, la forma de funcionar del símbolo + depende de los operandos:
 - Si los dos operandos son números, entonces + actúa como una suma.
 - Si al menos uno de los dos operandos es un String, entonces + actúa como un operador de concatenación.

```
// 3
System.out.println(1+2);

// 12
System.out.println("1" + 2);

// 11
System.out.println("1" + "1");
```

- Utilizando paréntesis se puede dar prioridad a la suma o a la concatenación.

```
int i = 20;  
  
// El valor es 2020  
System.out.println("El valor es " + i + 20);  
  
// El valor es 40  
System.out.println("El valor es " + (i + 20));
```

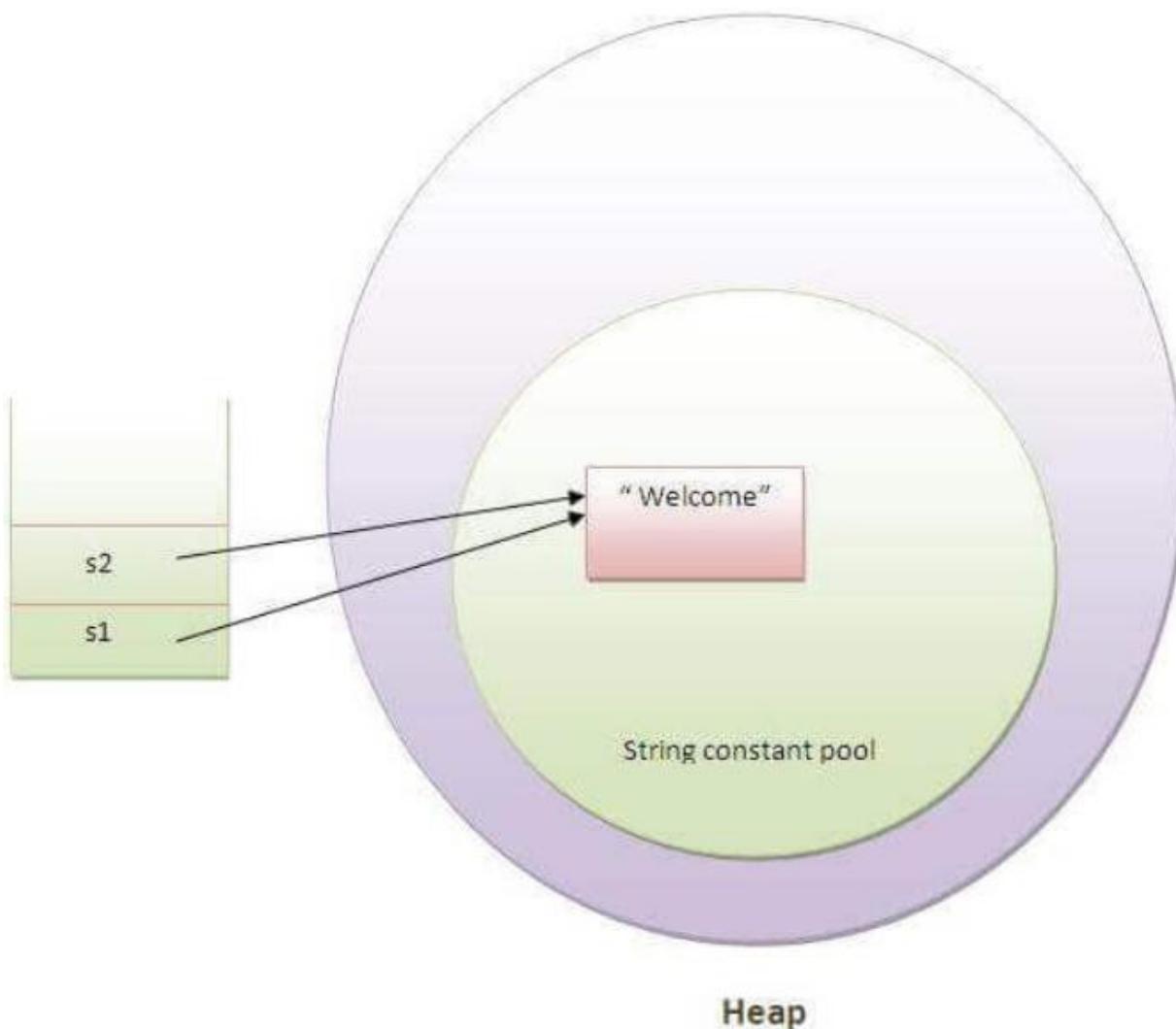
Inmutabilidad

- Una variable de tipo String es un objeto inmutable. Esto quiere decir que su valor no puede ser cambiado.
- Los métodos de String vistos con anterioridad no modifican el String original, sino que devuelven uno nuevo.

```
String texto1 = "Esto es un texto";  
String texto2 = texto1.concat(" mayor");  
  
// Esto es un texto  
System.out.println(texto1);  
  
// Esto es un texto mayor  
System.out.println(texto2);
```

- Cada vez que crea un String se comprueba primero si existe uno idéntico en el "String constant pool" (grupo constante de cadenas). Si el String ya existe, se retorna una referencia a la misma. Si el String no existe, se crea una nueva instancia y se coloca en el grupo.

```
String s1 = "Welcome";  
  
// no se crea una nueva instancia, sino que se aprovecha de la anterior  
String s2 = "Welcome";
```



- Por esta razón no es conveniente utilizar el operador de comparación == para comparar Strings, porque realmente la comparación se realiza sobre las referencias, en lugar de sobre el contenido de los Strings. Para comparar Strings hay que utilizar el método equals.

StringBuffer y StringBuilder

- Las clases StringBuffer y StringBuilder permiten crear variables de tipo String que posteriormente serán modificadas.
 - Es recomendable utilizar StringBuffer en programas con múltiples subprocesos manipulando la misma variable.
 - De lo contrario, StringBuilder es más apropiada porque posee mejor rendimiento, tanto en ejecución, como en uso de memoria.

```
/*
Concatenación utilizando String
*/
long startTime = System.currentTimeMillis();
```

```
String str1 = "";
for(int i=0; i<100000; i++) {
    str1 += i;
}

long endTime = System.currentTimeMillis() - startTime;

System.out.printf("El tiempo que ha tardado con String es de %d milisegundos",
endTime).println();

/*
Concatenación utilizando StringBuilder
*/

startTime = System.currentTimeMillis();

StringBuilder str2 = new StringBuilder();
for(int i=0; i<100000; i++) {
    str2.append(i);
}

endTime = System.currentTimeMillis() - startTime;
System.out.printf("El tiempo que ha tardado con StringBuilder es de %d milisegundos",
endTime).println();

/*
Concatenación utilizando StringBuffer
*/

startTime = System.currentTimeMillis();

StringBuffer str3 = new StringBuffer();
for(int i=0; i<100000; i++) {
    str3.append(i);
}

endTime = System.currentTimeMillis() - startTime;
System.out.printf("El tiempo que ha tardado con StringBuffer es de %d milisegundos",
endTime).println();
```

- En conclusión:

- En general, para declarar una variable de tipo texto se empleará String.
 - En situaciones especiales donde se requiera de muchas modificaciones de un String, la opción recomendada es StringBuilder.
 - StringBuffer es la clase más adecuada para trabajar con masivas modificaciones de String en un entorno multihilo.
- StringBuilder y StringBuffer poseen las mismas formas de inicialización:

- (): crea un StringBuilder o un StringBuffer vacío con una capacidad inicial de 16 caracteres por defecto.
- (String str): crea un StringBuilder o un StringBuffer vacío inicializado con una cadena de texto pasada por argumentos, donde la capacidad es mínimo de 16 caracteres.
- (int capacity): crea un StringBuilder o un StringBuffer vacío con una capacidad inicial definida por capacity.

```
// tres formas posibles de inicializar un StringBuilder
StringBuilder strb1 = new StringBuilder();
StringBuilder strb2 = new StringBuilder("hola");
StringBuilder strb3 = new StringBuilder(100);

// tres formas posibles de inicializar un StringBuffer
StringBuffer strb1 = new StringBuffer();
StringBuffer strb2 = new StringBuffer("hola");
StringBuffer strb3 = new StringBuffer(100);
```

- StringBuffer y StringBuilder poseen los mismos métodos.

Métodos de StringBuilder y StringBuffer

- `toString()`: convierte el StringBuilder o el StringBuffer en un String.

```
StringBuilder strb = new StringBuilder("hola");
String str = strb.toString();

// hola
System.out.println(strb);

// hola
System.out.println(str);
```

- `capacity()`: retorna la capacidad actual del StringBuilder o del StringBuffer.
 - La capacidad se refiere a la cantidad total para almacenar caracteres.
 - Esta capacidad puede aumentar con el uso de métodos como `append`.
 - Por defecto, la capacidad es de 16 elementos vacíos.
 - Cuando a StringBuilder se le pasa un String, la capacidad es de 16 más la longitud del String pasado.

```
// variable de tipo StringBuilder con una capacidad de 16 elementos vacíos
StringBuilder strb1 = new StringBuilder();

// variable de tipo StringBuilder con la capacidad del String pasado por parámetro (4
```

caracteres), más 16 elementos vacíos adicionales
StringBuilder strb2 = new StringBuilder("holo");

```
// variable de tipo StringBuilder con una capacidad de 1  
StringBuilder strb3 = new StringBuilder(1);
```

```
// 16  
System.out.println(strb1.capacity());
```

```
// 20  
System.out.println(strb2.capacity());
```

```
// 1  
System.out.println(strb3.capacity());
```

- `length()`: retorna la longitud del `StringBuilder` o del `StringBuffer`.

```
StringBuilder strb1 = new StringBuilder();  
StringBuilder strb2 = new StringBuilder("holo");  
StringBuilder strb3 = new StringBuilder(1);
```

```
// 0  
System.out.println(strb1.length());
```

```
// 4  
System.out.println(strb2.length());
```

```
// 0  
System.out.println(strb3.length());
```

- `append(String str)`: añade una cadena de texto al final del `StringBuilder` o del `StringBuffer`.

```
StringBuilder strb = new StringBuilder("holo");
```

```
// agrega el carácter a al final del StringBuilder  
strb.append("a");
```

```
// holaa  
System.out.println(strb);
```

- `insert(int offset, String str)`: añade una cadena de texto al `StringBuilder` o del `StringBuffer` a partir del índice indicado por el offset.

```
StringBuilder strb = new StringBuilder("la");
// agrega la cadena de texto ho en el índice 0
strb.insert(0, "ho");

// hola
System.out.println(strb);

// agrega la cadena de texto ca en el índice 2
strb.insert(2, "ca");

// hocala
System.out.println(strb);

// error porque el StringBuilder posee una longitud menor que 22
// strb.insert(22, "ho");
```

- `replace(int start, int end, String str)`: reemplaza con una cadena de texto desde el índice start hasta el índice end - 1, o hasta el final si end es mayor que la longitud del StringBuilder o del StringBuffer.

```
StringBuilder strb1 = new StringBuilder("hola a todos");
StringBuilder strb2 = new StringBuilder("hola a todos");

// reemplaza desde el índice 5 al índice 6 (1 carácter) por la cadena de texto dd
strb1.replace(5, 6, "dd");

// hola dd todos
System.out.println(strb1);

// reemplaza desde el índice 5 al índice 22 (hasta el final del StringBuilder) por la cadena de texto dd
strb2.replace(5, 22, "dd");

// hola dd
System.out.println(strb2);
```

- `setCharAt(int index, char ch)`: reemplaza el carácter de un índice pasado como primer argumento, por el carácter que se le pasa en el segundo.

```
StringBuilder strb = new StringBuilder("hola a todos");

// reemplaza el carácter del índice 2 del StringBuilder con el carácter c
strb.setCharAt(2, 'c');
```

```
// hoca a todos
System.out.println(strb);
```

- reverse(): invierte el orden de los caracteres.

```
StringBuilder strb = new StringBuilder("hola a todos");
strb.reverse();
// sodot a aloh
System.out.println(strb);
```

- delete(int start, int end): borra los caracteres entre dos índices pasados por argumentos.

```
StringBuilder strb = new StringBuilder("hola a todos");
// elimina los caracteres del índice 1 al 3
strb.delete(1, 3);
// ha a todos
System.out.println(strb);
```

- deleteCharAt(int index): borra el carácter del índice pasado por argumento.

```
StringBuilder strb = new StringBuilder("hola a todos");
// elimina el carácter del índice 1
strb.deleteCharAt(1);
// hla a todos
System.out.println(strb);
```

- Otros métodos importantes comentados anteriormente para String y presentes también en StringBuilder y StringBuffer son: charAt, subString y indexOf.
- El operador concatenación no funciona para los tipos StringBuilder y StringBuffer. Es necesario convertirlos a String previamente.

```
StringBuilder strb1 = new StringBuilder("hola a ");
StringBuilder strb2 = new StringBuilder("todos");
String str = strb1.toString() + strb2.toString();
```

```
// hola a todos  
System.out.println(str);
```

Ejercicio: escribe un programa que concatene dos StringBuilder, utilizando append y también el operador concatenación.

Ejercicio: escribe un programa que elimine el último carácter de un StringBuilder.

Ejercicio: escribe un programa que inserte el carácter 'b' cada tres posiciones en un StringBuilder.

Ejercicio: escribe un programa que convierta en mayúsculas la primera letra de cada palabra de un StringBuilder.

Ejercicio: escribe un programa para invertir cada palabra de un párrafo de un StringBuilder.

Tu carrera digital ~

Módulo 3

Java básico

Arrays



Array

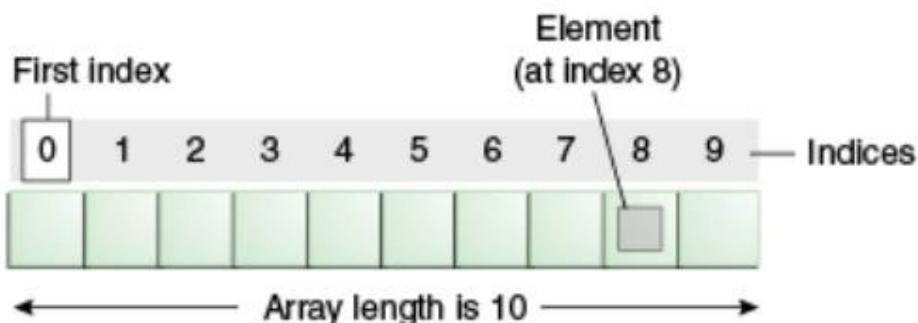
- Arrays
- Métodos estáticos de la clase Arrays
- Entendiendo el método main

Arrays

- En ocasiones puede interesar agrupar un conjunto de valores del mismo tipo, en lugar de utilizar múltiples variables.

```
int manzanasVendidasDia1 = 24;
int manzanasVendidasDia2 = 4;
int manzanasVendidasDia3 = 23;
int manzanasVendidasDia4 = 55;
int manzanasVendidasDia5 = 34;
int manzanasVendidasDia6 = 33;
```

- Los arrays permiten agrupar un conjunto ordenado de valores del mismo tipo.



```
int manzanasVendidas[] = { 24, 4, 23, 55, 34, 33 };
```

- Una vez declarados, los arrays ocupan un espacio fijo en la memoria, es decir, no pueden crecer en tamaño. Sin embargo, poseen un rendimiento muy rápido respecto a otro tipo de estructuras de datos.
- Para crear un array es necesario declararlo, instanciarlo y, opcionalmente, inicializarlo.

```
// declaración (no se puede utilizar el array todavía si no se instancia)
int a[];
```

```
// declaración e instanciación de un array de 31 elementos (todos los valores son
```

inicializados a cero por defecto en caso de arrays de tipo int, 0.0 en arrays de tipo float y false en arrays de tipo boolean)

```
int b[] = new int[31];
```

// declaración, instanciaión de un array de 6 elementos e inicialización con valores

```
int c[] = { 24, 4, 23, 55, 34, 33 };
```

- Para acceder a los elementos del array se utiliza un índice y el operador de indexación []. La expresión [0] representa el primer elemento del array, [1] el segundo y así sucesivamente.

```
int numeros[] = { 24, 4, 23, 55, 34, 33 };
```

// 24

```
System.out.println(numeros[0]);
```

// 4

```
System.out.println(numeros[1]);
```

- Al igual que los String, length permite obtener el número de elementos de un array, aunque en este caso es un atributo en lugar de un método.

```
int numeros[] = { 24, 4, 23, 55, 34, 33 };
```

// 6

```
System.out.println(numeros.length);
```

// 33

```
System.out.println(numeros[numeros.length - 1]);
```

- Para recorrer un array por completo puede utilizarse un bucle for junto con el atributo length.

```
int numeros[] = { 24, 4, 23, 55, 34, 33 };
```

```
for(int i=0; i<numeros.length; i++) {
    System.out.println(numeros[i]);
}
```

- Sin embargo, existe otra sintaxis más simple para recorrer los elementos de un array. Esta sintaxis es conocida como forEach.

```
int numeros[] = { 24, 4, 23, 55, 34, 33 };

/*
numeros es el array
numero es la variable de tipo int que irá cambiando en cada iteración, comenzando por
el primer elemento del array y terminando por el último
*/
for(int numero : numeros) {
    System.out.println(numero);
}
```

- Los valores de un array pueden ser modificados utilizando la expresión [index], siendo index la posición del elemento a modificar.

```
int numeros[] = { 24, 4, 23, 55, 34, 33 };

// 24
System.out.println(numeros[0]);

// modifica el valor del primer elemento (24) a 0
numeros[0] = 0;

// 0
System.out.println(numeros[0]);
```

Ejercicio: escribe un programa que genere un array de 6 valores de tipo int y sean mostrados en pantalla utilizando un bucle for.

- Los arrays pueden almacenar valores primitivos u objetos, pero todos deben ser del mismo tipo.

```
// correcto porque todos los objetos heredan de Object
Object datos[] = { "a", 'a', true, 55, 2.3, 33 };

// true porque el cuarto elemento es un objeto de tipo Integer y no una variable
// primitiva de tipo int (instanceof solamente trabaja con objetos, no funciona con variables
// primitivas)
System.out.println(datos[3] instanceof Integer);
```

- Un método puede recibir un número indefinido de parámetros si en la declaración del mismo se utiliza la expresión type...values como argumento a recibir, donde type es el tipo de datos del array y values el nombre del array.

```

public class Main {

    public static void main(String[] args) {

        // 24
        System.out.println(Main.suma(1,3,4,5,4,3,4));

        // 1
        System.out.println(Main.suma(1));

        // 11
        System.out.println(Main.suma(1,3,3,4));
    }

    // método estático que puede recibir un número indefinido de parámetros de tipo int
    static int suma(int... numeros) {

        int value = 0;

        for (int numero : numeros) {
            value += numero;
        }

        return value;
    }
}

```

Métodos estáticos de la clase Arrays

- Arrays.toString([]): imprime todos los valores del array directamente.

```

int numeros[] = { 24, 4, 23, 55, 34, 33 };

// [24, 4, 23, 55, 34, 33]
System.out.println(Arrays.toString(numeros));

```

- Arrays.fill([]): cambia todos los valores de un array por un valor concreto.

```

// declaración e instanciación de un array de 5 elementos (todos con el valor 0 por
// defecto)
int[] numeros = new int[5];

// cambia todos los valores del array a 100
Arrays.fill(numeros, 100);

```

- `Arrays.equals([], [])`: compara dos arrays y retorna true solo si:
 - Ambos arrays poseen la misma longitud.
 - Los elementos para todos los índices son iguales en los dos arrays.

```
int[] numeros1 = { 1,2,3,4,5 };
int[] numeros2 = { 1,2,3,4,5 };

// true
System.out.println(Arrays.equals(numeros1, numeros2));
```

- `Arrays.sort()`: ordena un array de menor a mayor (en caso de arrays de tipo char realiza una ordenación ASCII). Es un método que no retorna nada (void).

```
int[] numeros = { 11,22,3,4,5,0 };
Arrays.sort(numeros);

// [0, 3, 4, 5, 11, 22]
System.out.println(Arrays.toString(numeros));

char[] caracteres = { 'c', 'b', 'a', 'A', 'C' };
Arrays.sort(caracteres);

// [A, C, a, b, c]
System.out.println(Arrays.toString(caracteres));
```

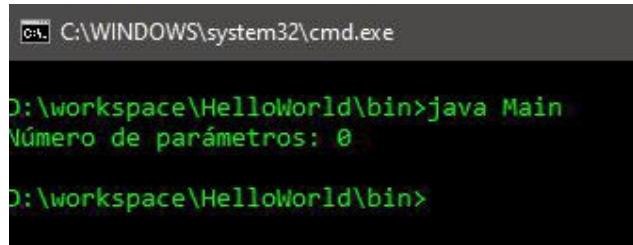
Entendiendo el método main

- `public static void main (String [] args)` es el método Java más importante porque es el punto de entrada de cualquier aplicación.
- Su sintaxis es siempre la misma y solamente puede cambiarse:
 - El nombre del argumento del array de String (`args`).
 - La forma de declarar el argumento del array de String con las dos siguientes opciones:
 - `String ... args`
 - `String args []`
- Los modificadores `public` y `static` son obligatorios para que la máquina virtual de Java pueda ejecutar este método al inicio.
- No tiene sentido que el método `main` retorne nada porque el programa finaliza cuando el método acaba. Por esa razón, el método `main` se declara como `void`.
- Aparecerá un error al ejecutar el programa en caso de no declarar correctamente el método `main`.

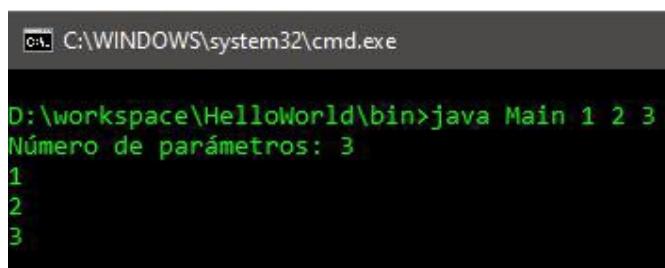
Error: Main method not found in class Main, please define the main method as:
 public static void main(String[] args) or a JavaFX application class must extend javafx.application.Application

- El método main acepta un único argumento de tipo array de String. El valor de los elementos de este array de String se corresponderá con los valores o argumentos suministrados por línea de comandos.
- Si no se suministra argumentos por línea de comandos, la longitud del array de String args es cero.

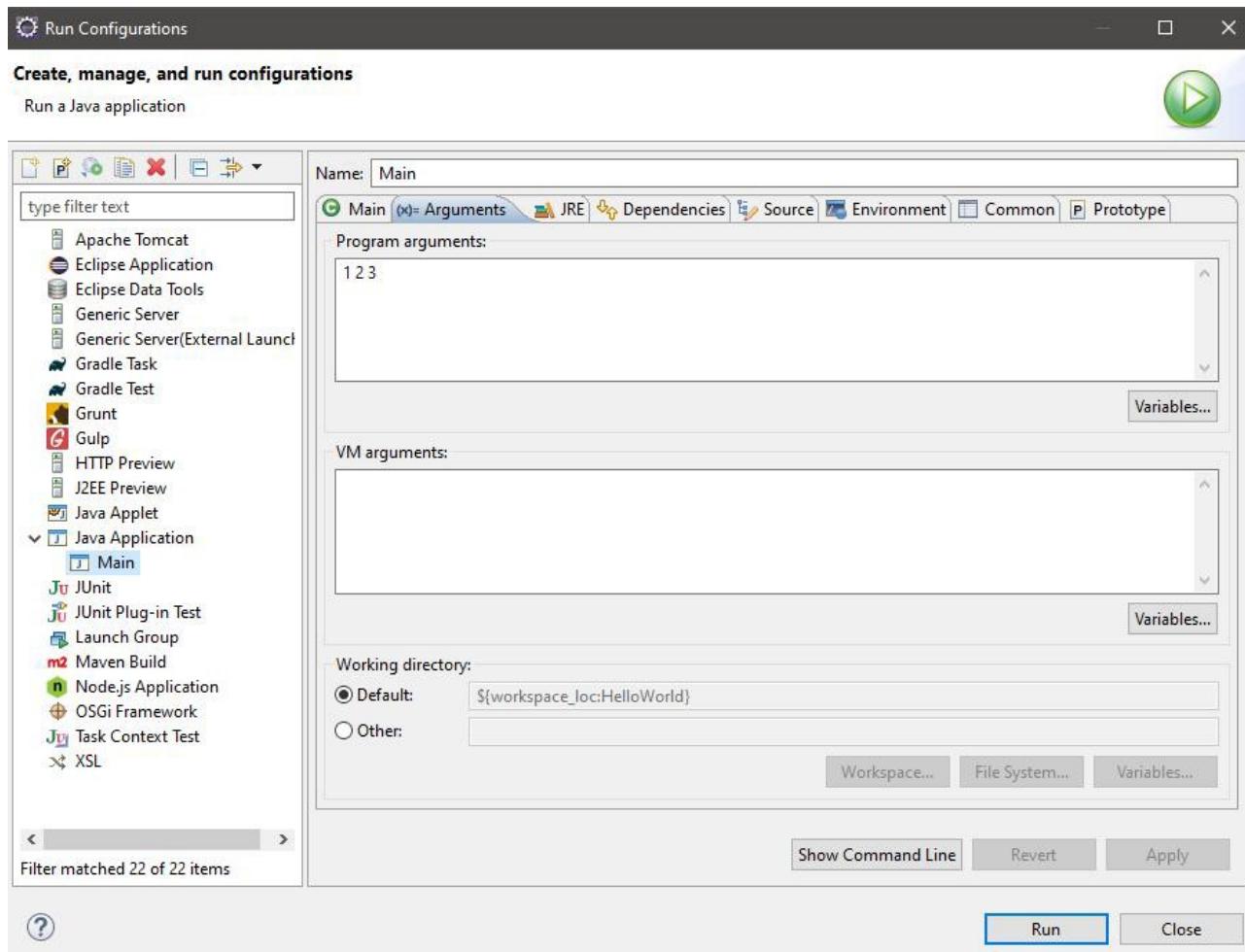
```
public class Main {  
  
    public static void main(String... args) {  
        System.out.printf("Número de parámetros: %d", args.length).println();  
  
        for(String arg : args) {  
            System.out.println(arg);  
        }  
    }  
}
```



- Los parámetros suministrados por línea de comandos pueden ser accedidos recorriendo el array.



- En Eclipse también se pueden definir argumentos desde Run - Run Configurations..



```
Problems Javadoc Declaration Console Coverage
<terminated> Main [Java Application] C:\Program Files\Java\jdk-11.0.5\bin\java
Número de parámetros: 3
1
2
3
```

Tu carrera digital ~

Módulo 3

Java básico

JSON



JSON

- ◆ [Introducción](#)
- ◆ [Sintaxis](#)
- ◆ [Tipos de datos](#)
- ◆ [JSON vs XML](#)
- ◆ [Librería JSON in Java](#)
- ◆ [Trabajando con la librería JSON in Java](#)

Introducción

- ◆ JSON (JavaScript Object Notation) es un formato de texto ligero para el almacenamiento e intercambio de datos.
- ◆ JSON es texto plano y puede ser convertido fácilmente en un objeto JavaScript.
- ◆ JSON es independiente de lenguaje de programación, aunque la manipulación de objetos JSON es más sencilla con JavaScript.

Sintaxis

- ◆ La sintaxis JSON se deriva de la sintaxis de notación de objetos JavaScript:
 - Un objeto JSON siempre comienza con una llave de apertura { y finaliza con una llave de cierre }.
 - Pueden existir objetos JSON vacíos (solamente tienen llave de apertura y de cierre)
 - Los datos se estructuran en pares clave/valor.
 - Cada par de datos se separa por una coma, excepto el último elemento.
 - Los nombres de las claves son siempre cadenas de texto (string) y requieren de comillas dobles.
 - Los valores pueden ser de tipo string, number, object, array, boolean y null.

```
{  
}
```

```
{  
  "nombre": "John"  
}
```

```
{  
  "nombre": "John",
```

```
"edad": 22,
"aficiones": ["baloncesto", "tenis", "lectura"]
}
```

- El tipo de archivo para archivos JSON es ".json"

Tipos de datos

- Los tipos de datos permitidos en JSON son: string, number, object, array, boolean y null. Otras expresiones de JavaScript, como las funciones, las variables de tipo date el valor undefined, no son permitidos.

```
{
  "a": "Hola",
  "b": 2,
  "c": {
    "c1": "Hola",
    "c2": 2
  },
  "d": true,
  "e": null,
  "f": [1, true, "Hola"],
  "g": [
    {
      "g1a": "a",
      "g1b": 1
    },
    {
      "g2a": "b",
      "g2b": 2
    },
    {
      "g3a": "c",
      "g3b": 3
    }
  ]
}
```

JSON vs XML

- Similitudes:
 - Con estructura jerárquica.
 - Con librerías de parseo para cualquier lenguaje de programación.
- Diferencias:
 - JSON no utiliza etiquetas.

- JSON es más sencillo de entender.
- Un objeto JSON ocupa menos espacio que un objeto XML para contener la misma información.
- JSON puede utilizar arrays.
- XML requiere de parseadores explícitos. JSON puede ser interpretado simplemente como un objeto.
- JSON puede ser manejado y recorrido de forma más simple.
- JSON no proporciona metadatos como los atributos de XML.

Librería JSON in Java

- Java incorpora una librería nativa para manejar JSON, pero no es muy utilizada debido su complejidad y falta de funcionalidades.
- Otras librerías de terceros como Gson, Jackson o JSON In Java son actualmente más utilizadas.
- La librería [JSON in Java](#) puede descargarse en el [repositorio oficial en Maven](#).
- La librería JSON in Java es un simple archivo JAR de unos pocos KB.
- Para trabajar con esta librería en un proyecto en Eclipse es necesario añadirla al build path.

Trabajando con la librería JSON in Java

- La clase principal de la librería JSON in Java es JSONObject.
- JSONObject permite crear objetos para manejar estructuras de datos de tipo JSON.

```
// creación de una objeto de tipo JSONObject
JSONObject objeto = new JSONObject();
```

- Un objeto JSONObject es similar a un objeto de tipo HashMap:
 - Las claves son únicas, no nulas y siempre de tipo String.
 - Los valores pueden ser de tipo: Boolean, numéricas (Short, Integer, Float, Long, Double), String, JSONArray o JSONObject.
 - JSONObject posee varios tipos de constructores parametrizados.
 - Un objeto JSONObject provee una serie de métodos para manejar los datos, entre ellos:
 - get(String key): obtiene el valor asociado a una clave.
 - getX(String key): obtiene el valor (del tipo X) asociado a una clave.
 - put(String key, X value): guarda un valor (del tipo X) a partir de una clave.
 - has(String key): devuelve true o false dependiendo de si la clave pasada existe o no.
 - length(): obtiene el número de pares clave/valor del objeto.

- keySet(): obtiene un array con todas las claves del objeto.

```
JSONObject objeto = new JSONObject();

// guarda valores de tipo Double, String, Boolean, JSONArray, JSONObject y null
objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);

// obtiene los valores asociados a sus respectivas claves
System.out.println(objeto.get("numero"));
System.out.println(objeto.get("str"));
System.out.println(objeto.get("bool"));
System.out.println(objeto.get("array"));
System.out.println(objeto.get("objeto"));
System.out.println(objeto.get("vacio"));

// devuelve true o false dependiendo de si la clave pasada existe o no
System.out.println(objeto.has("numero"));
System.out.println(objeto.has("numero2"));

// obtiene el número de elementos clave/valor del objeto (6)
System.out.println(objeto.length());

// devuelve un array con todas las claves del objeto
System.out.println(objeto.keySet());

// {"str":"valor","vacio":null,"numero":12,"bool":true,"array":[],"objeto":{}}
System.out.println(objeto);
```

- El orden de los elementos de un objeto JSONObject no necesariamente es mantenido según el orden de inserción.

```
JSONObject objeto = new JSONObject();

objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);

// {"str":"valor","vacio":null,"numero":12,"bool":true,"array":[],"objeto":{}}
System.out.println(objeto);
```

- La forma de iterar un objeto JSONObject es aplicando un forEach al array de claves devuelto por el método keySet.

```
JSONObject objeto = new JSONObject();

objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);

for(String key : objeto.keySet()) {
    System.out.println(key + ":" + objeto.get(key));
}
```

- Los objetos de tipo JSONArray también poseen una serie de métodos para obtener y agregar valores (métodos de tipo get y put).
 - El orden de inserción en un objeto de tipo JSONArray se mantiene (al contrario que sucedía con los objetos de tipo JSONObject). Por ello, para obtener los valores con get es necesario suministrar el índice o posición del elemento (como en los ArrayList).

```
JSONArray array = new JSONArray();

// guarda valores de tipo Integer, JSONObject y String
array.put(1);
array.put(new JSONObject());
array.put("str");

// guarda un valor de tipo JSONObject
JSONObject objeto = new JSONObject();
objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);
array.put(objeto);

// [1,[],"str",{"str":"valor","vacio":null,"numero":12,"bool":true,"array":[],"objeto":{}}]
System.out.println(array);

// obtiene el valor del primer elemento (índice 0)
System.out.println(array.get(0));

// obtiene el valor del cuarto elemento (índice 3)
```

```
// {"str":"valor","vacio":null,"numero":12,"bool":true,"array":[],"objeto":{}}
System.out.println(array.get(3));
```

- Puede utilizarse el método `toString()` para convertir un objeto `JSONObject` a tipo `String` (para enviar, por ejemplo, los datos a un servidor).

```
JSONObject objeto = new JSONObject();
```

```
objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);
```

```
// {"str":"valor","vacio":null,"numero":12,"bool":true,"array":[],"objeto":{}}
String datos = objeto.toString();
```

- Pasando un número al método `toString()` puede establecerse el número de espacios a utilizar en el identado del `String` en formato JSON devuelto.

```
JSONObject objeto = new JSONObject();
```

```
objeto.put("numero", 12.0);
objeto.put("str", "valor");
objeto.put("bool", true);
objeto.put("array", new JSONArray());
objeto.put("objeto", new JSONObject());
objeto.put("vacio", JSONObject.NULL);
```

```
// se definen 3 espacios de identación
System.out.println(objeto.toString(3));
```

```
/*
{
  "str": "valor",
  "vacio": null,
  "numero": 12,
  "bool": true,
  "array": [],
  "objeto": {}
}
```

- Una tarea muy habitual en programación cuando se trabaja con JSON es convertir datos de tipo `String` en formato JSON a un objeto de tipo `JSONObject`.

- Debido a que el formato JSON utiliza comillas dobles obligatorias en las claves, es necesario escapar estos caracteres cuando se almacenan en una variable de tipo String en Java.

```
// variable de tipo String donde se almacena datos en formato JSON
String contenido = "
{\\"balance\\":1000.21,\\"is_vip\\":true,\\"num\\":100,\\"name\\":\\"foo\\"}"";

// conversión a JSONObject utilizando el constructor de la clase
JSONObject objeto = new JSONObject(contenido);

System.out.println(objeto.get("name"));
System.out.println(objeto.get("num"));
System.out.println(objeto.get("balance"));
System.out.println(objeto.get("is_vip"));
```

- También se puede realizar una conversión desde un objeto de tipo HashMap a un objeto de tipo JSONObject.

```
// objeto de tipo Map donde se almacena claves de tipo String y valores de tipo String
HashMap<String, String> map = new HashMap<>();

// inserción de algunos pares clave/valor para el objeto de tipo HashMap
map.put("name", "jon doe");
map.put("age", "22");
map.put("city", "chicago");

// conversión a JSONObject utilizando el constructor de la clase
JSONObject objeto = new JSONObject(map);

// {"name":"jon doe", "city":"chicago", "age":"22"}
System.out.println(objeto);
```

- La conversión desde una clase de tipo Java Bean (clase donde solamente existen atributos y métodos de tipo get y set) también es una tarea simple utilizando el constructor de JSONObject.

```
// Foo.java
public class Foo {

    private int a;
    private String b;
    private boolean c;

    public Foo(int a, String b, boolean c) {
        this.a = a;
```

```

this.b = b;
this.c = c;
}

public int getA() {
    return a;
}

public void setA(int a) {
    this.a = a;
}

public String getB() {
    return b;
}

public void setB(String b) {
    this.b = b;
}

public boolean isC() {
    return c;
}

public void setC(boolean c) {
    this.c = c;
}
}

```

```

// objeto de tipo Foo (clase de tipo Java Bean)
Foo foo = new Foo(2, "hola", true);

// conversión a JSONObject utilizando el constructor parametrizado de la clase
JSONObject
JSONObject fooJson = new JSONObject(foo);

// {"a":2,"b":"hola","c":true}
System.out.println(fooJson);

```

- No existe una forma flexible de realizar el proceso inverso (convertir un JSONObject a Java Bean) mediante JSON in Java. Sin embargo, existen otras librerías más avanzadas que sí permiten esta opción: ([Jackson](#) o [Gson](#)).

Ejercicio: escribir un programa que lea un archivo con formato JSON, cargarlo desde Java para leer sus propiedades y mapearlo a una clase Java Bean (una clase que solamente tiene atributos y métodos getters/setters). Puede tomarse como ejemplo los datos en formato JSON devueltos por la [API de Star Wars](#).

Ejercicio proyecto (Main16): implementa dos métodos nuevos (getGestoresJSON y getMensajesJSON) en la clase Datos para convertir los HashMap de los gestores y mensajes a objetos JSONObject.

Ejercicio proyecto (Main17): implementa dos métodos nuevos (setGestoresJSON y setMensajesJSON) en la clase Datos para convertir los objetos JSONObject a los HashMap de los gestores y mensajes.

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.

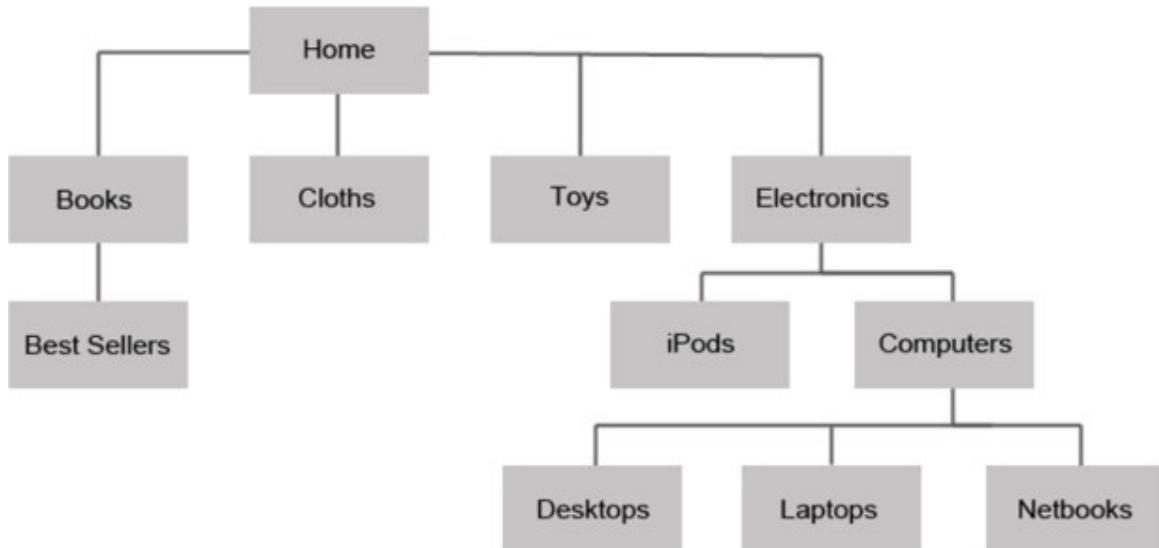


- Una base de datos es una colección organizada de datos estructurados que se almacenan en un sistema informático.
- Normalmente, una base de datos está controlada por un Sistema de gestión de bases de datos (SGBD).
- Un SGBD es un software que permite el almacenamiento, modificación y extracción de la información en una base de datos.
- Los datos y el SGBD constituyen el sistema de base de datos o simplemente base de datos.

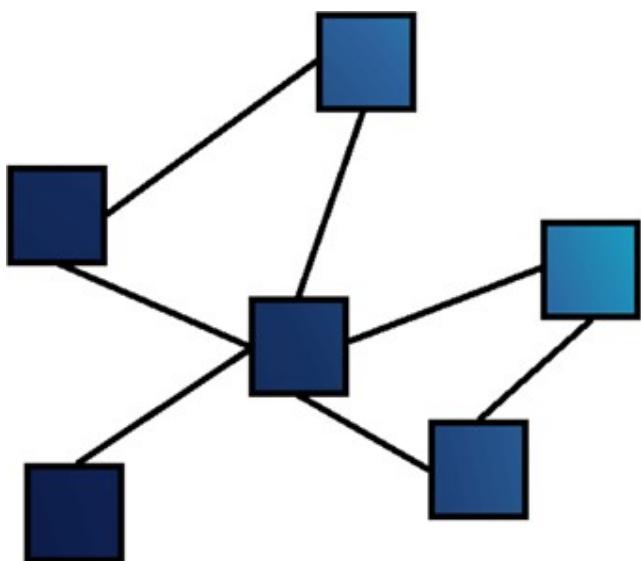


Tipos de bases de datos

- 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).



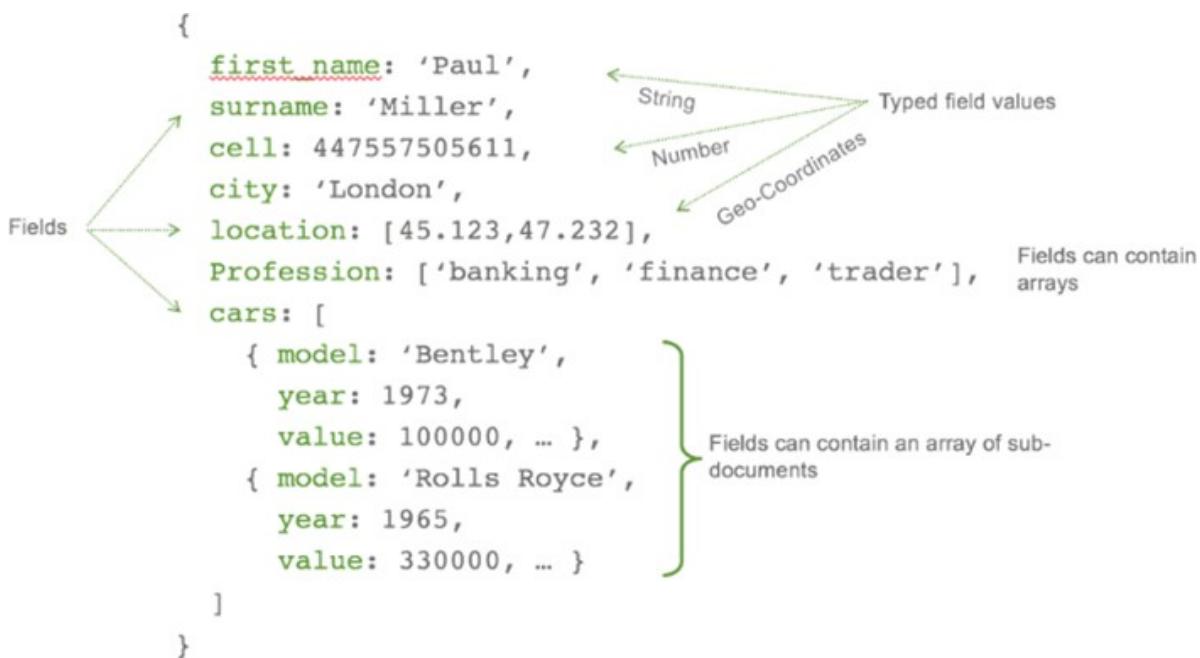
- Bases de datos con estructura en red: contiene relaciones más complejas que las jerárquicas, admitiendo que un dato puede estar relacionado con otro o con varios al mismo tiempo sin necesidad de jerarquía.



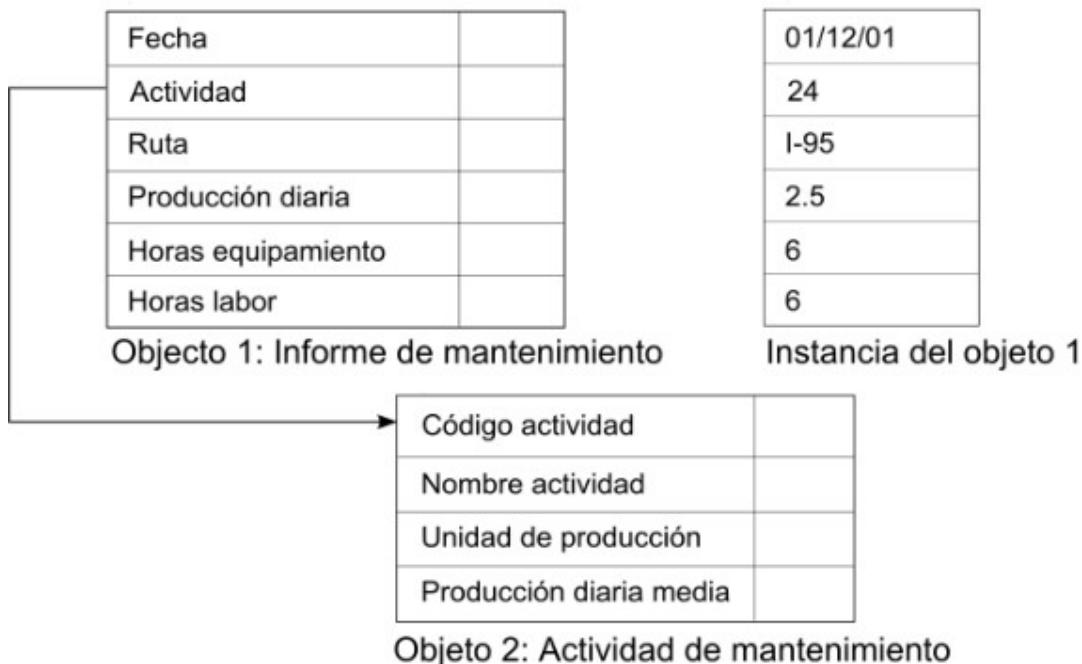
- Bases de datos clave/valor: presentan una estructura de datos que asocia claves con valores. La operación principal que soporta de manera más eficiente es la búsqueda por clave.

| 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).

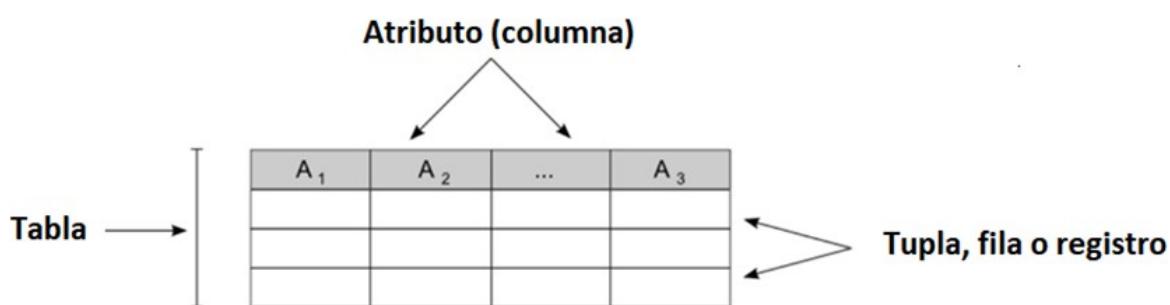


- Bases de datos con estructura orientada a objetos: los datos son almacenados de acuerdo a una estructura orientada a objetos, siguiendo el paradigma de los lenguajes orientados a objetos.



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.



| id | usuario | password | email |
|-----------|----------------|--|--------------------|
| 1 | pepito | 627a6ab55608711e421a1132be531a7a7fec1df7 b253dda73ec482824269e5d7 | pepito@correo.com |
| 2 | juanito | 44e235daceb8ee492f6bb5249ffb0aec993a330 d8c1fd9cf30447deae1ecda3 | juanito@correo.com |
| 3 | marcos | 2a5d486545bbd36a0c1a9786a68a1d578370c2e eb339699f853b031322d0eb5c | marcos@correo.com |
| 4 | roberto | daf5f918ab83da2bcb7bbe7de3b18d70f77e4041 a20626f40144c2d6adf62875 | roberto@correo.com |

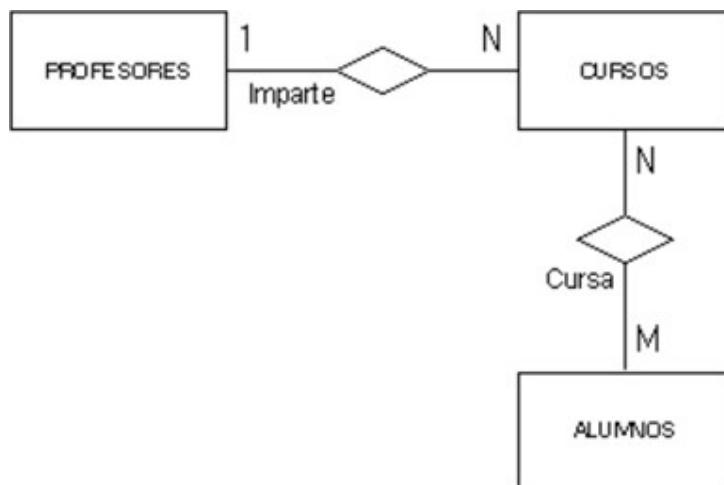
- Cada registro en la tabla es llamado tupla o fila.
- Los atributos o columnas son los campos que constituyen cada tupla.
- Cada columna está compuesto obligatoriamente por un nombre y un tipo de dato.
- Adicionalmente al nombre y al tipo de dato, cada columna de una tabla puede ser también clave primaria.
- 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.



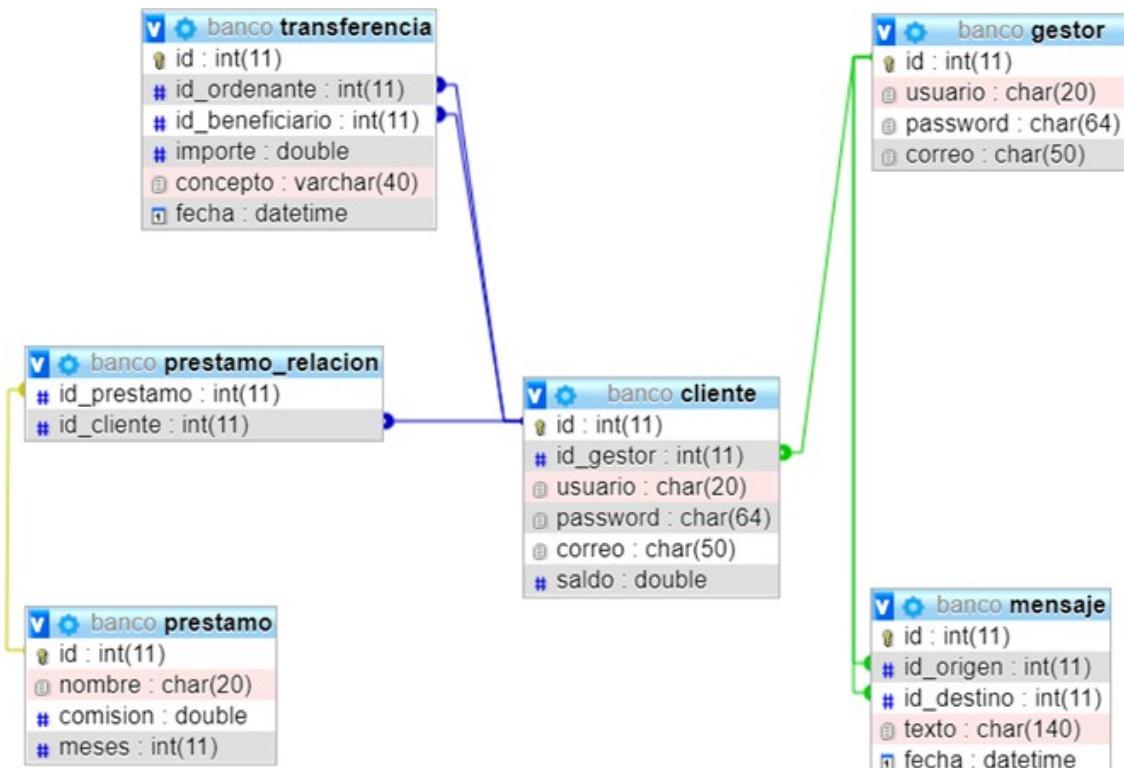
- A veces es útil definir también una columna que es clave primaria también como autoincremental.
- Una columna autoincremental aumentará su valor cada vez que se inserte una nueva tupla.

| id | usuario | password | email |
|-----------|----------------|--|--------------------|
| 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 software escrita en PHP para manejar MariaDB/MySQL mediante la Web.



- ◆ XAMP puedes descargar 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 ¿Qué está incluido?. | md5 sha1 | Descargar (64 bit) 145 Mb |
| 7.3.13 / PHP 7.3.13 ¿Qué está incluido?. | md5 sha1 | Descargar (64 bit) 146 Mb |
| 7.4.1 / PHP 7.4.1 ¿Qué está incluido?. | md5 sha1 | Descargar (64 bit) 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?

Yes No

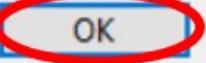
- Algunas funciones de XAMPP pueden estar limitadas si el User Account Control (UAC) de Windows se encuentra activo.

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



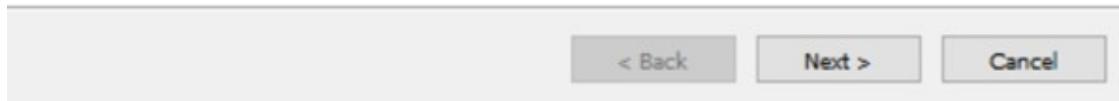
- El asistente de instalación de XAMPP es muy simple.

Setup

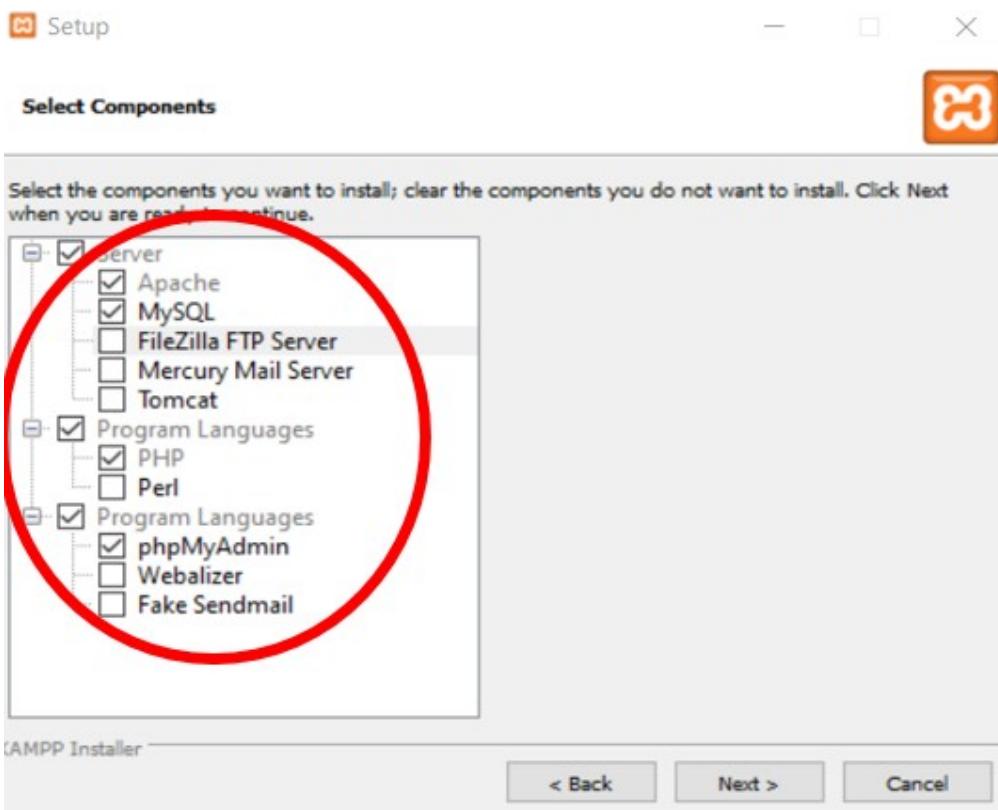


Setup - XAMPP

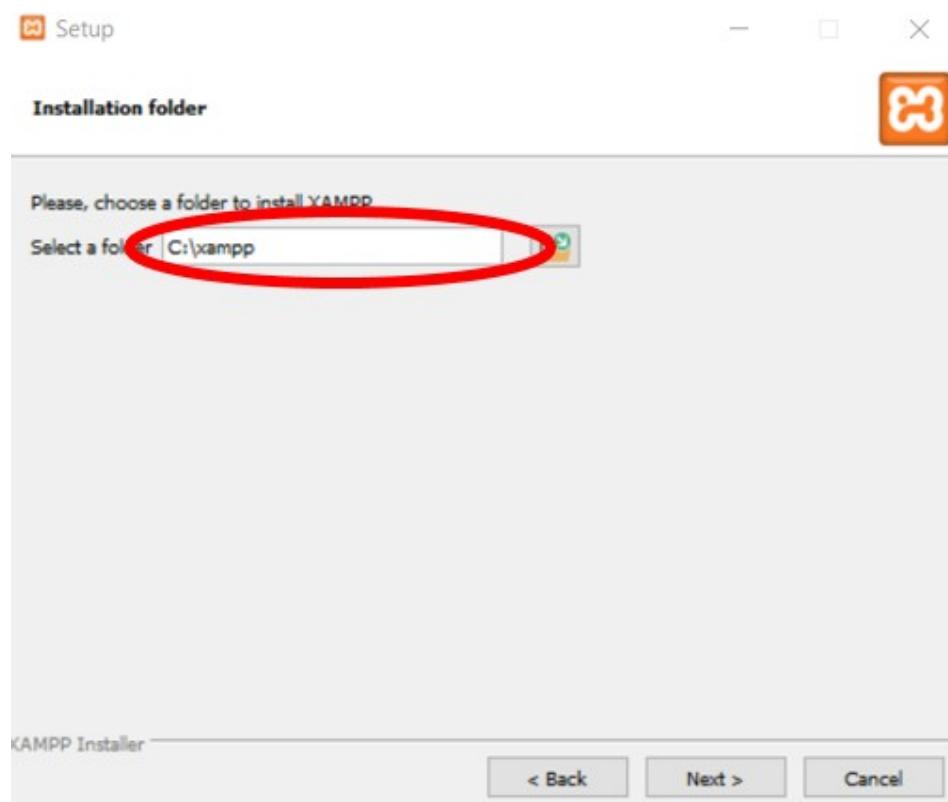
Welcome to the XAMPP Setup Wizard.



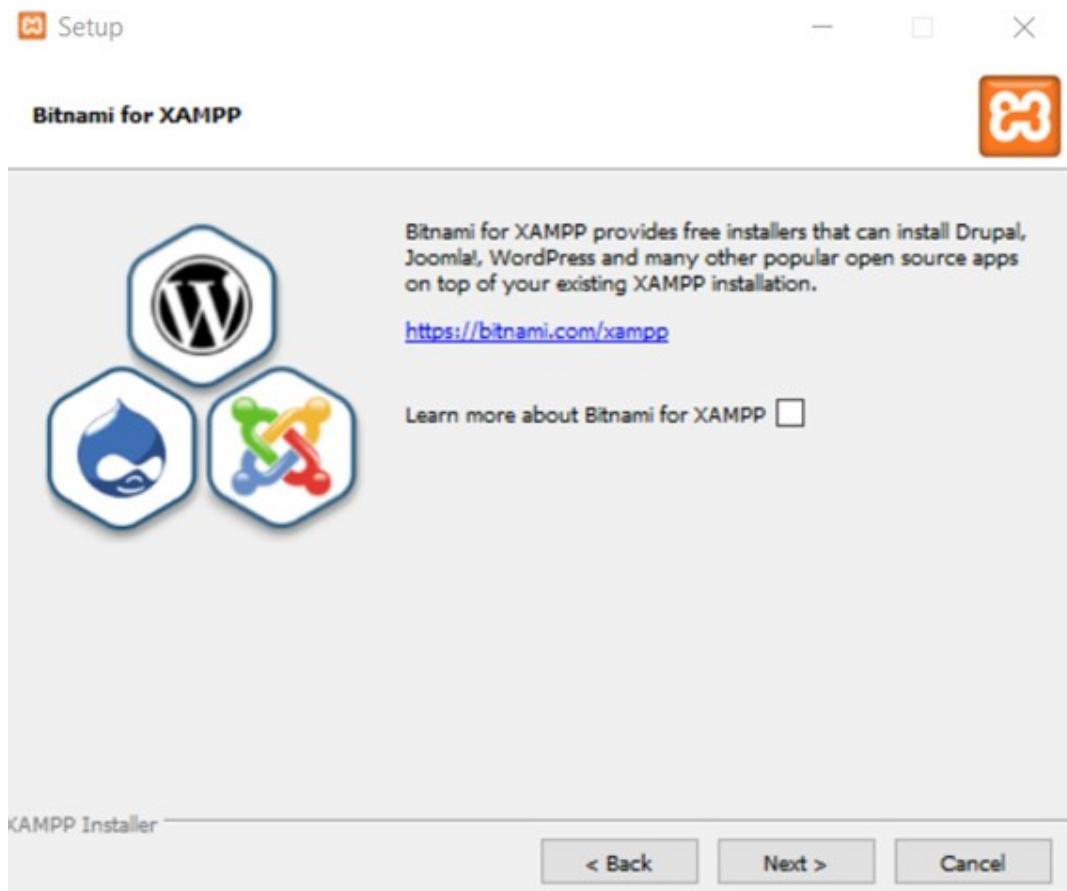
- Los únicos componentes de XAMPP a instalar son: Apache, MySQL, PHP y phpMyAdmin. El resto pueden deseleccionarse.



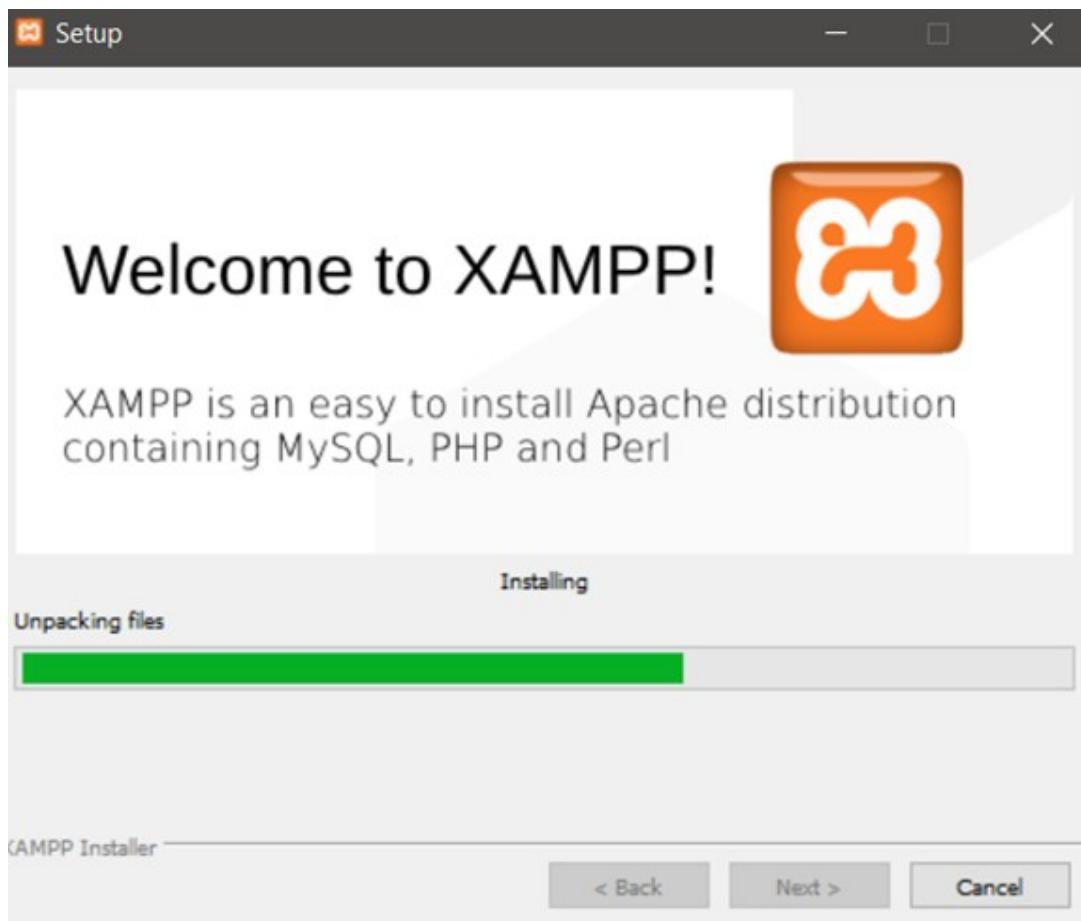
- A continuación aparece la ruta de destino donde se instalará XAMPP.



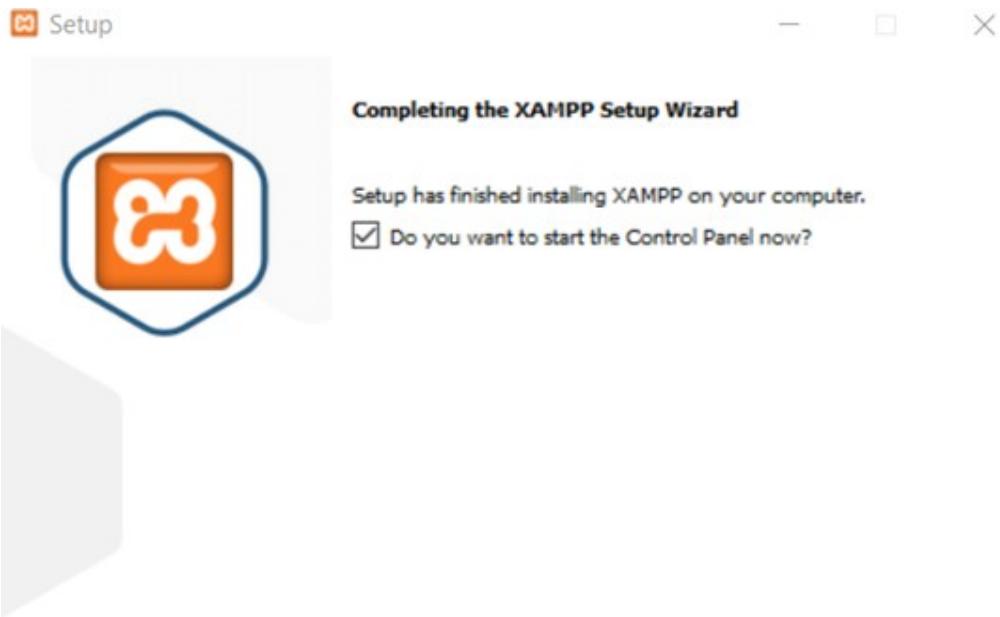
- En el siguiente paso se ofrecerá la posibilidad de obtener más información sobre la iniciativa Bitnami.



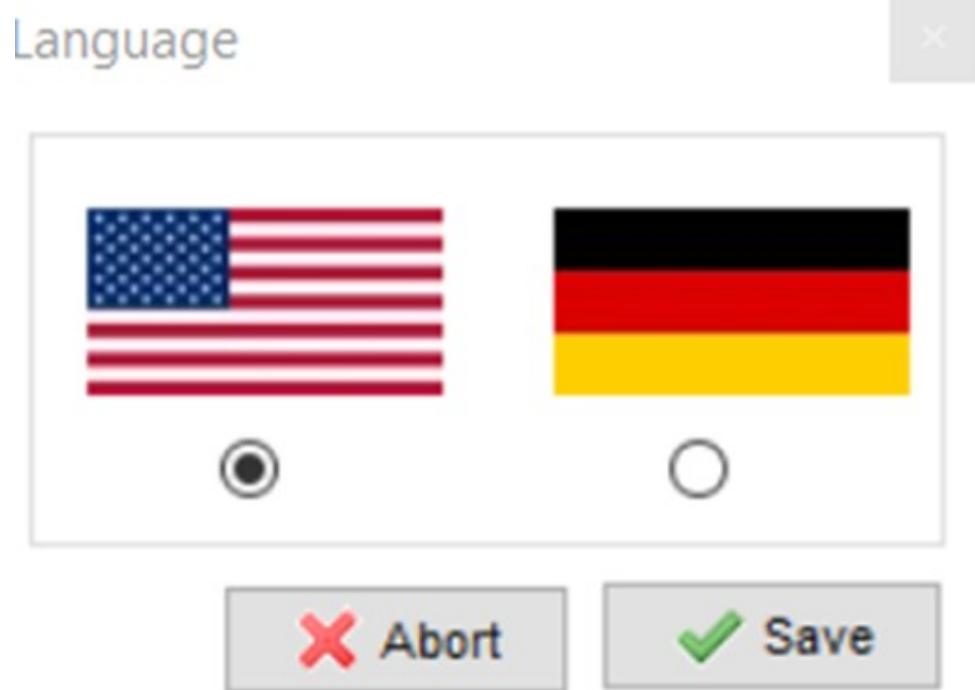
- Seguidamente el proceso de instalación comenzará.



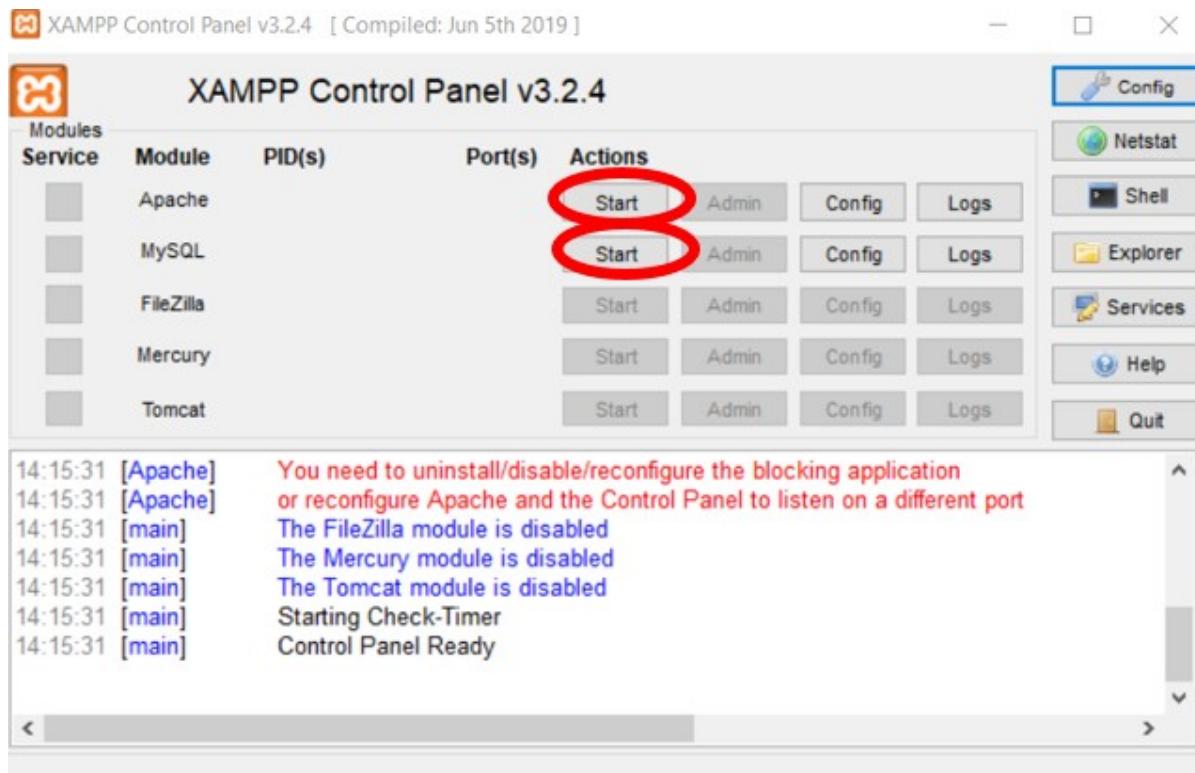
- Finalmente es posible abrir directamente el Panel de Control de XAMPP una vez terminada la instalación.



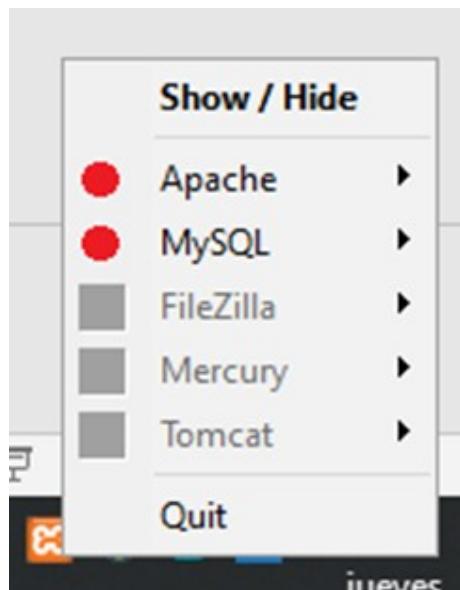
- En la primera ejecución del Panel de Control de XAMPP aparecerá la posibilidad de elegir entre el idioma inglés o el alemán.



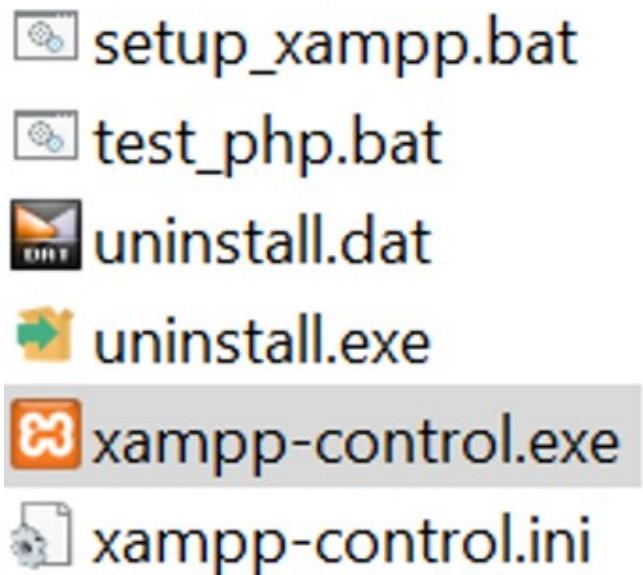
- El Panel de Control de XAMPP permite arrancar o parar servicios, cambiar los archivos de configuración, entre otras tareas.



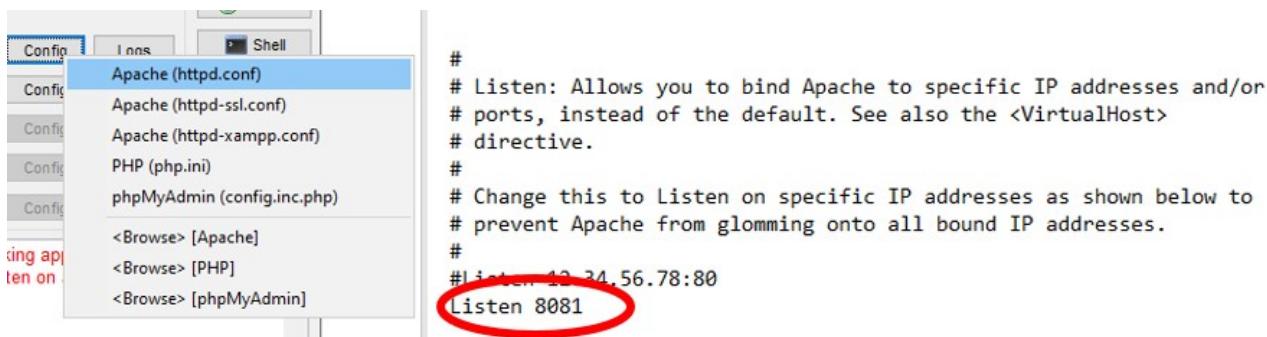
- El Panel de Control de XAMPP también aparece en la barra de notificaciones de Windows.
- Para cerrarlo hay que pulsar con el botón derecho del ratón sobre el icono de XAMPP y Quit.



- El ejecutable (xampp-control.exe) del Panel de Control de XAMPP se encuentra en el directorio C:\xampp (directorio de instalación por defecto).



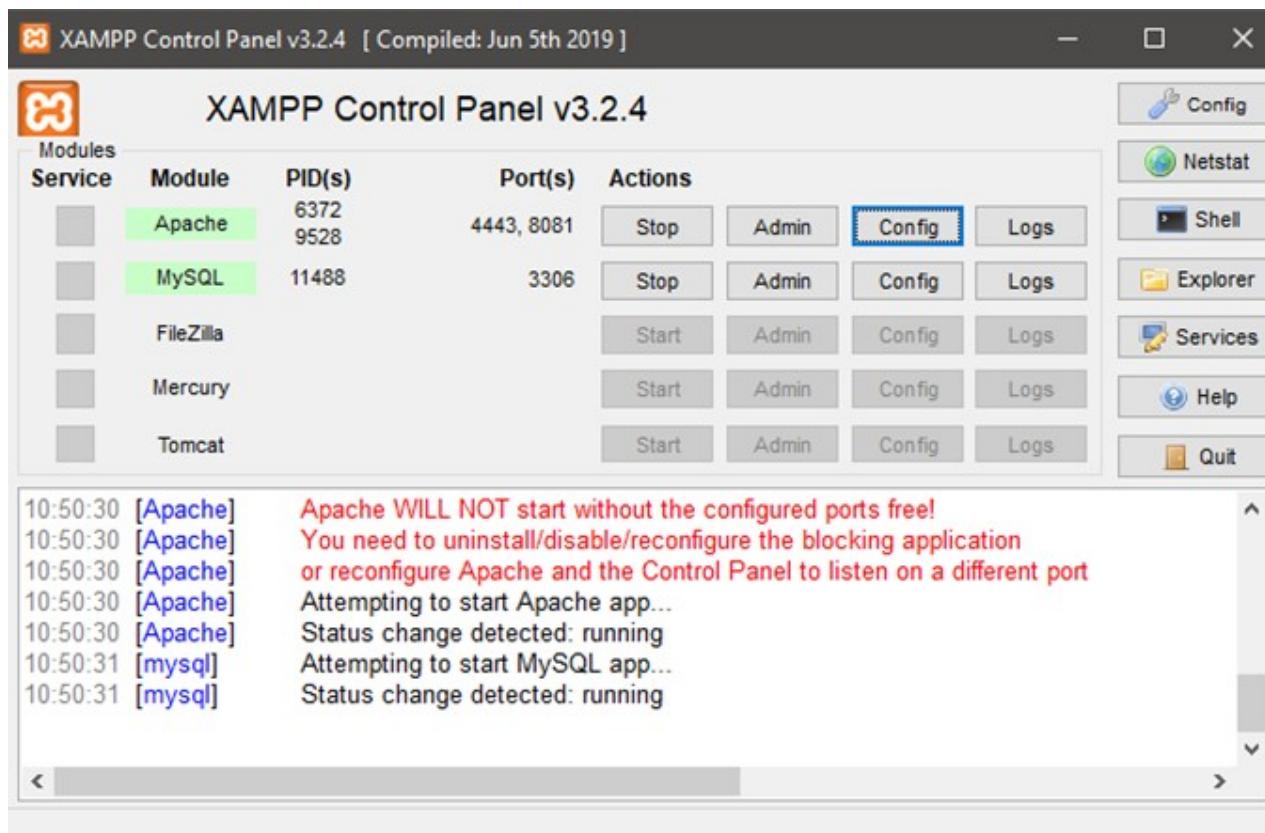
- En Windows pueden presentarse problemas para abrir puertos por debajo del 1024 (puertos bien conocidos).
- El servidor Apache sirve peticiones HTTP en el puerto 80, por lo que será necesario cambiarlo a uno superior al 1024 desde el archivo httpd.conf



- 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



- 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.

Apache Friends Applications FAQs HOW-TO Guides PHPInfo phpMyAdmin

XAMPP Apache + MariaDB + PHP + Perl

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 interface with the following sections:

- Configuraciones generales:** Cotejamiento de la conexión al servidor: utf8mb4_unicode_ci.
- Configuraciones de apariencia:** Idioma - Language: Español - Spanish; Tema: pmahomme; Tamaño de fuente: 82%.
- Servidor de base de datos:**
 - Servidor: 127.0.0.1 vía TCP/IP
 - Tipo de servidor: MariaDB
 - Conexión del servidor: No se está utilizando SSL
 - Versión del servidor: 10.4.11-MariaDB - mariadb.org binary distribution
 - Versión del protocolo: 10
 - Usuario: root@localhost
 - Conjunto de caracteres del servidor: UTF-8 Unicode (utf8mb4)
- Servidor web:**
 - Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.4.1
 - Versión del cliente de base de datos: libmysql - mysqld 7.4.1
 - extensión PHP: mysqli curl mbstring
 - Versión de PHP: 7.4.1
- phpMyAdmin:**
 - Acerca de esta versión: 4.9.2, versión estable más reciente: 4.9.3
 - Documentación
 - Página oficial de phpMyAdmin
 - Contribuir
 - Obtener soporte
 - Lista de cambios
 - Licencia

- 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 | | Sí | Editar privilegios Exportar |
| root | ::1 | No | ALL PRIVILEGES | | Sí | Editar privilegios Exportar |
| root | localhost | No | ALL PRIVILEGES | | Sí | Editar privilegios Exportar |

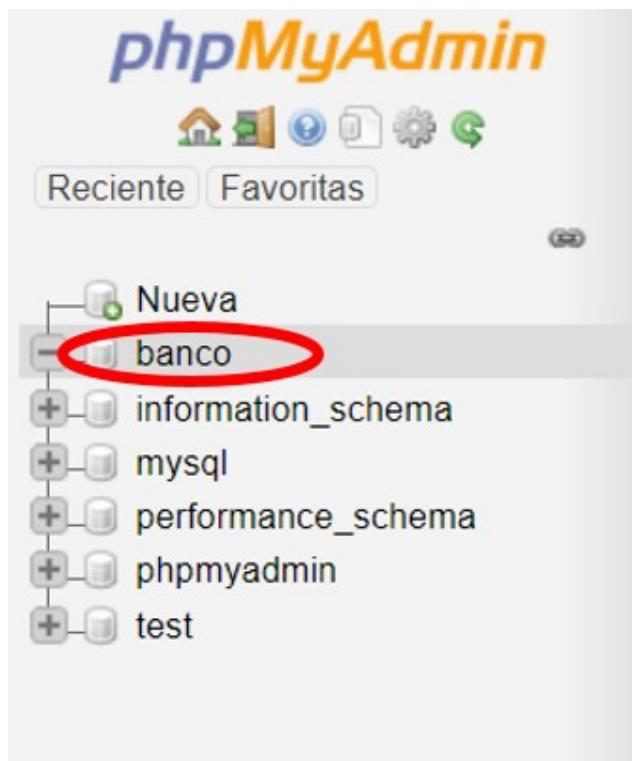
↑ Seleccionar todo Para los elementos que están marcados: Exportar

Nuevo Agregar cuenta de usuario

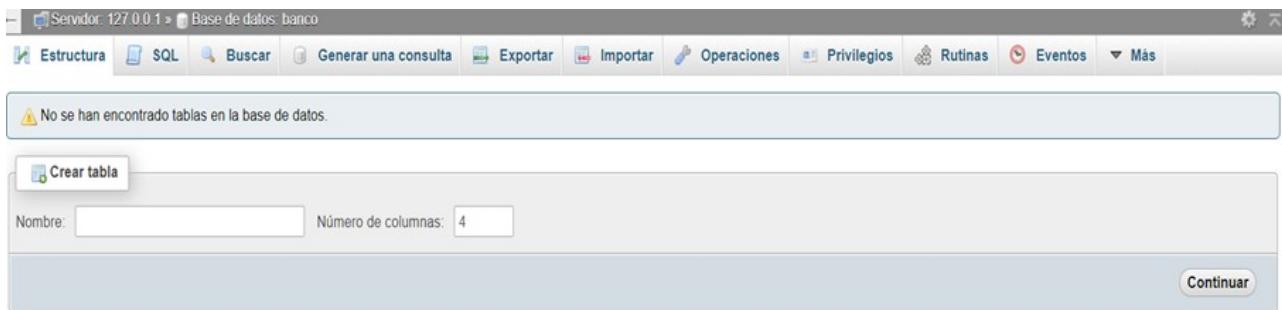
- Para crear un nuevo usuario se establece: nombre de usuario, nombre de host (el valor debe ser localhost), contraseña (dos veces). Finalmente se crea una base de datos (con el mismo nombre que el usuario) otorgando todos los privilegios al usuario creado.

Agregar cuenta de usuario

- 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.

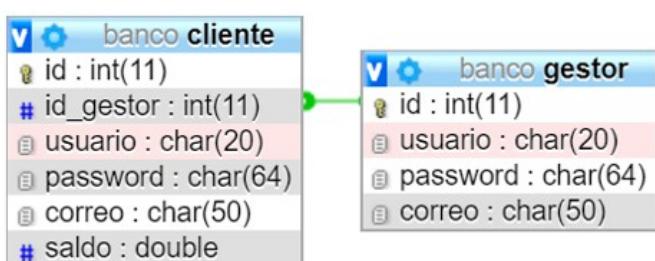


Creación de tablas

- 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 |
| <u>id_gestor</u> | INT | FK |
| usuario | CHAR(20) | |
| password | CHAR(64) | |
| correo | CHAR(50) | |
| saldo | DOUBLE | |

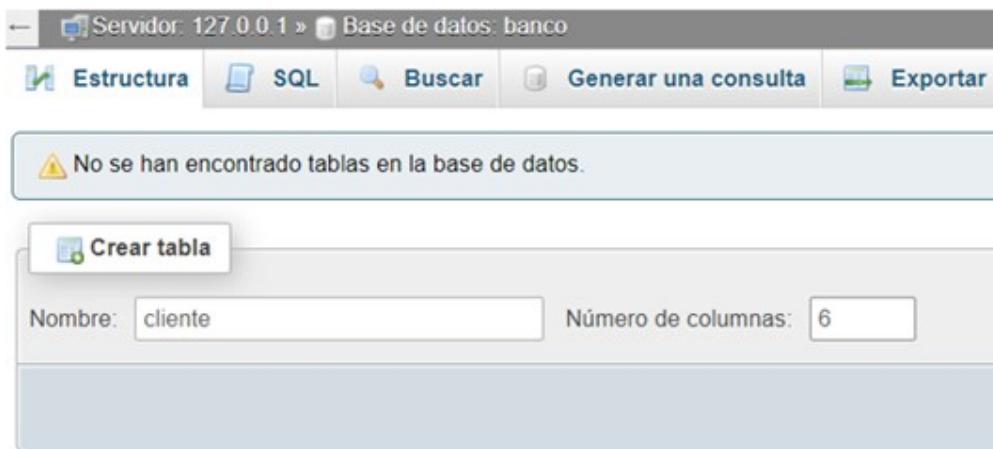


PK → clave primaria
FK → clave foránea
AI → autoincremental

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 | |

Índice
A_I

Índice

PRIMARY
 PRIMARY

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 | 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 | |

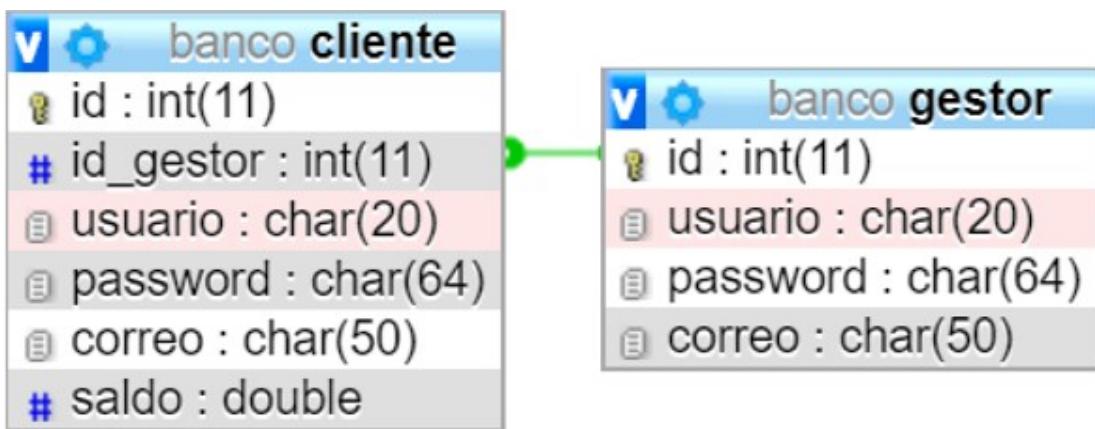
- En la tabla anterior puede declararse las columnas usuario y correo como índices de tipo único (UNIQUE).
- Si se declaran varias columnas como de clave única, entonces la clave única es de campo múltiple (por eso es mejor declarar usuario y correo como tipo UNIQUE).
- Pueden consultarse los datos de la tabla desde la pestaña "Examinar".
- Inicialmente la tabla usuario no presenta ningún registro.

- El procedimiento para crear la tabla gestor es similar.

| Nombre | Tipo | Longitud/Valores | Predeterminado | Cotejamiento | Atributos | Nulo | Índice | AJ |
|----------|------|------------------|----------------|--------------|-----------|---------|---------|----|
| id | INT | | Ninguno | | | PRIMARY | PRIMARY | |
| usuario | CHAR | 20 | Ninguno | | | | | |
| password | CHAR | 64 | Ninguno | | | | | |
| correo | CHAR | 50 | Ninguno | | | | | |

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|-----------------|----------|--------------------|-----------|---------|----------------|-------------|----------------|------------------------|
| 1 | id 🛡 | int(11) | | No | Ninguna | | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | usuario | char(20) | utf8mb4_general_ci | No | Ninguna | | | | Cambiar Eliminar Más |
| 3 | password | char(64) | utf8mb4_general_ci | No | Ninguna | | | | Cambiar Eliminar Más |
| 4 | correo | 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).

| Columna | Restricción de clave foránea (INNODB) |
|-----------|---------------------------------------|
| id_gestor | Base de datos Tabla Columna |
| | banco gestor id |

- 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 | 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
 Agregar a columnas centrales Eliminar de las columnas centrales

Imprimir Planteamiento de la estructura de tabla Hacer seguimiento a la tabla Mover columnas Normalizar

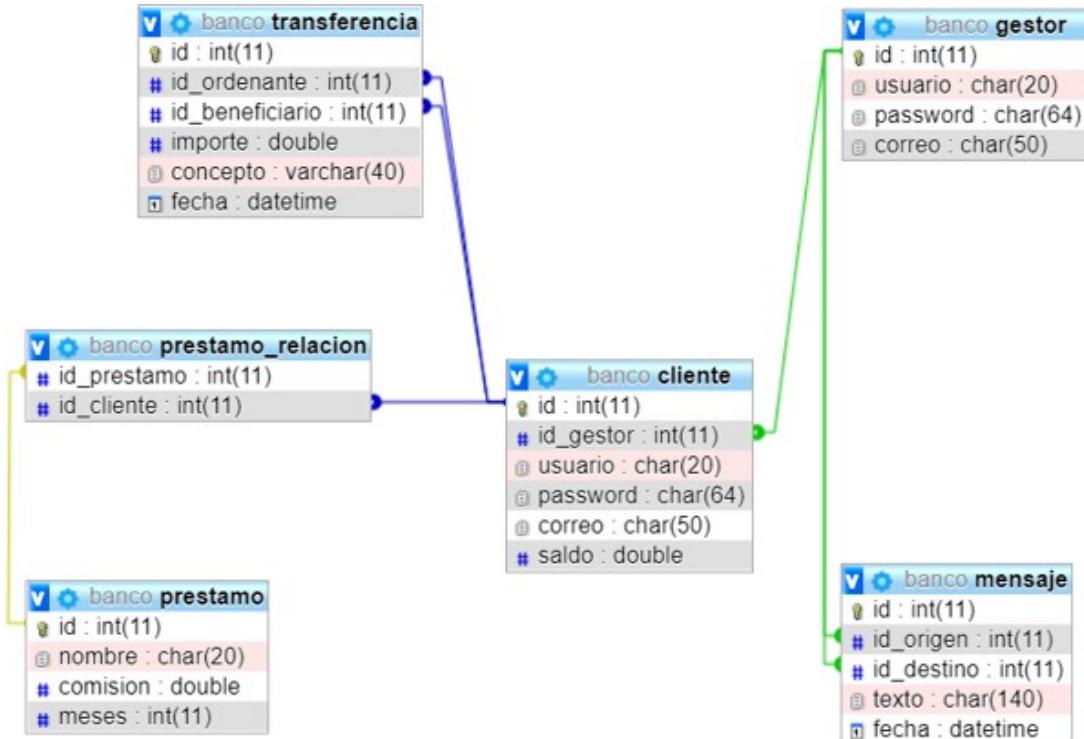
Agregar 1 columna(s) después de saldo Continuar

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".

- 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 | |
| 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 | | ← T → | ▼ | id | usuario | password | correo |
|--------------------------|------------------------|--|--|----|------------------------|---|--------------------------|
| <input type="checkbox"/> | Editar | Copiar | Borrar | 1 | gestor1 | d94d9ca1462655945362f71cdd0bbb31e4033e50462bf50a35... | gestor1@correo.com |
| ↑ | ↓ | <input checked="" type="checkbox"/> Seleccionar todo | Para los elementos que están marcados: | | Editar | Copiar | Borrar |
| | | | | | | | Exportar |

- 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... 1fd9cf30447deae1ecda3 |
| 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"/> | Editar Copiar Borrar | 1 | 1 | pepito | 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7 | pepito@correo.com | 2000 |
| <input type="checkbox"/> | Editar Copiar Borrar | 2 | 1 | juanito | 44e235daceb8ee492f6bb5249ffb0aecc993a330d8c1fd9cf3... | juanito@correo.com | 1000 |

Seleccionar todo Para los elementos que están marcados: [Editar](#) [Copiar](#) [Borrar](#) [Exportar](#)

- 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="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 1 | 1 | pepito | 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7 | pepito@correo.com | 2000 |
|--------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---|---|---------|--|--------------------|------|
| <input type="checkbox"/> | <input type="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 2 | 1 | juanito | 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7 | juanito@correo.com | 1000 |

↑ Seleccionar todo Para los elementos que están marcados:

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

Operaciones sobre los resultados de la consulta

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="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 1 | 1 | pepito | 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7 | pepito@correo.com | 2000 |
| <input checked="" type="checkbox"/> | <input type="button" value="Editar"/> | <input type="button" value="Copiar"/> | <input type="button" value="Borrar"/> | 2 | 1 | juanito | 44e235daceb8ee492f6bb5249ff0aecc993a330d8c1fd9cf3... | juanito@correo.com | 1000 |

↑ Seleccionar todo Para los elementos que están marcados:

Columna | Tipo | Función | Nulo | Valor

| | | | | |
|-----------|----------|--|------|--|
| id | int(11) | | Nulo | 1 |
| id_gestor | int(11) | | Nulo | 1 - gestor1 |
| usuario | char(20) | | Nulo | pepito |
| password | char(64) | | Nulo | 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73ec482824269e5d7 |
| correo | char(50) | | Nulo | pepito@correo.com |
| saldo | double | | Nulo | 2000 |

- 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
 Editar Borrar 1 pepito 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73e... pepito@correo.com 2000
 Editar Borrar 1 juanito 44e235daceb8ee492f6bb5249ffb0aecc993a330d8c1fd9cf3... juanito@correo.com 1000

↑ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

+ Opciones id id_gestor usuario password correo saldo
 Editar Borrar 1 pepito 627a6ab55608711e421a1132be531a7a7fec1df7b253dda73e... pepito@correo.com 2000

↑ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

- Desde la pestaña “Operaciones” de una tabla se puede renombrarla, copiarla, vaciarla (TRUNCATE) o borrarla (DROP), entre otras funciones.

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
 Editar Borrar 1 gestor1 d94d9ca1462655945362f71cdd0bbb31e4033e50462bf50a35... gestor1@correo.com

↑ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

- Desde la pestaña “Operaciones” de una base de datos se puede renombrarla, eliminarla o copiarla, entre otras funciones.

Servidor: 127.0.0.1 > Base de datos: banco

Estructura SQL Buscar Generar una consulta Exportar Importar Operaciones Privilegios Rutinas Eventos

Filtros

Que contengan la palabra: []

| Tabla | Acción | Filas | Tipo | Cotejamiento | Tamaño | Residuo a depurar |
|-------------------|---|-------|--------|--------------------|---------|-------------------|
| cliente | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| gestor | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| mensaje | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| prestamo | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| prestamo_relacion | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| transferencia | Examinar Estructura Buscar Insertar Vaciar Eliminar | 0 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| 6 tablas | Número de filas | | | | | |

Seleccionar todo Para los elementos que están marcados: ▾

- El botón “Exportar” permite descargar en un archivo SQL todos los datos y la estructura de todas las tablas de una base de datos.

phpMyAdmin

Reciente Favoritas

- Nueva
- banco**
- information_schema
- mysql
- performance_schema
- phpmyadmin
- test

Servidor: 127.0.0.1 > Base de datos: banco

Estructura SQL Buscar Generar una consulta Exportar

Filtros

Que contengan la palabra: []

- 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:

Plantillas existentes:

Plantilla: -- Seleccionar plantilla --

Método de exportación:

- Rápido - mostrar sólo el mínimo de opciones de configuración
- Personalizado - mostrar todas las opciones de configuración posibles

Formato:

- 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's a tree view of databases: 'Nueva', 'banco' (highlighted with a red circle), 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', and 'test'. On the right, the main panel shows the 'Importar' button highlighted with a red circle. The URL in the browser bar is 'Servidor: 127.0.0.1 » Base de datos: banco'.

- 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 el formato [formato].[compresión]. Por ejemplo: .sql.zip

Buscar en su ordenador | **Seleccionar archivo** | Ningún archivo seleccionado (Máximo: 40MB)

También puede arrastrar un archivo en cualquier página.

Conjunto de caracteres del archivo: utf-8 ▾

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: 0

Otras opciones:

Habilite la revisión de las claves foráneas

Formato:

SQL ▾

Opciones específicas al formato:

Modalidad SQL compatible: NONE ▾

No utilizar AUTO_INCREMENT con el valor 0

Continuar

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

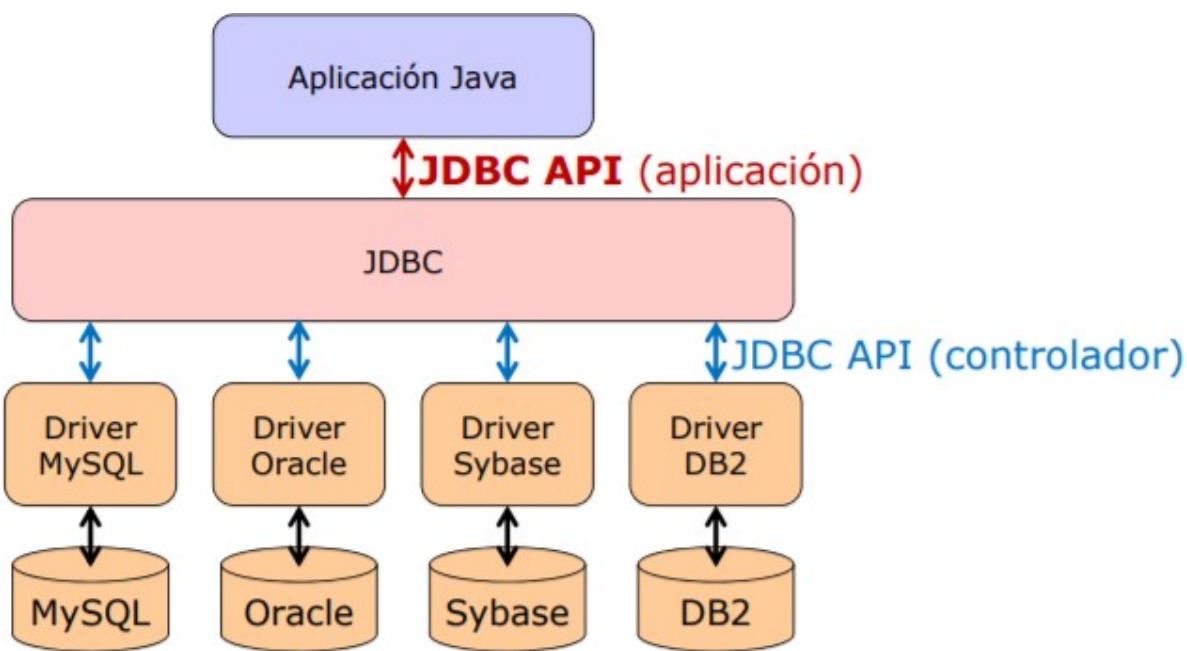


JDBC

- ◆ 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.

④ MySQL Community Downloads

◀ Connector/J

General Availability (GA) Releases

Connector/J 8.0.18

Select Operating System: Platform Independent ▾

Looking for previous GA versions?

| Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.18.tar.gz) | 8.0.18 | 3.7M | Download |
|---|--------|------|-----------------|
| Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.18.zip) | 8.0.18 | 4.4M | Download |

! 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());
}

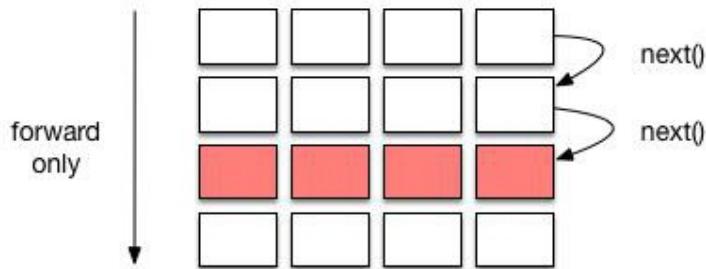
// ejecutar instrucción con el método executeUpdate

```

```
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.

| Tipo SQL | Tipo Java |
|-------------|----------------------|
| 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,

```

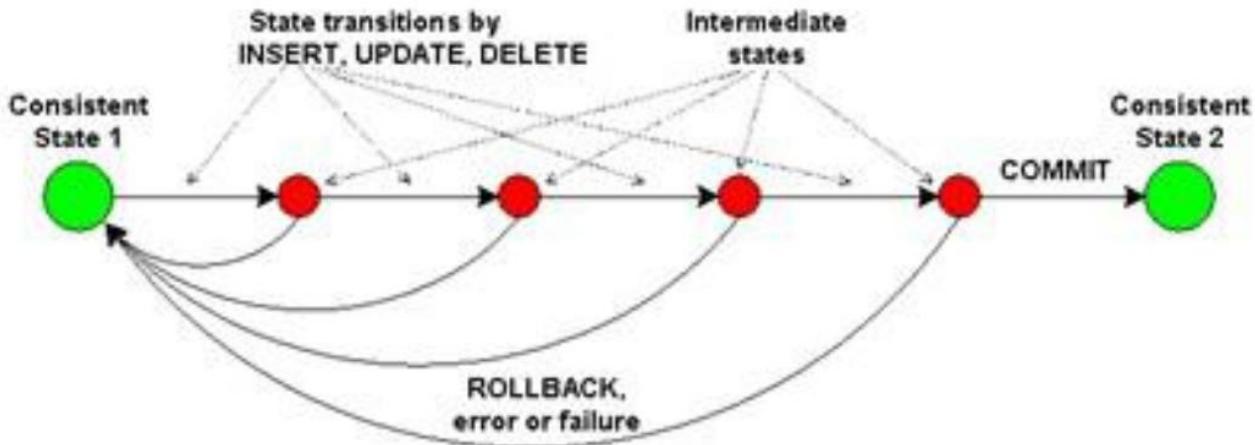
```
password, correo) VALUES (?, ?, ?));  
  
// se sustituye la primera aparición del carácter ? con el valor gestor10  
ps.setString(1, "gestor10");  
  
// se sustituye la segunda aparición del carácter ? con el valor gestor10  
ps.setString(2, "gestor10");  
  
// se sustituye la tercera aparición 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

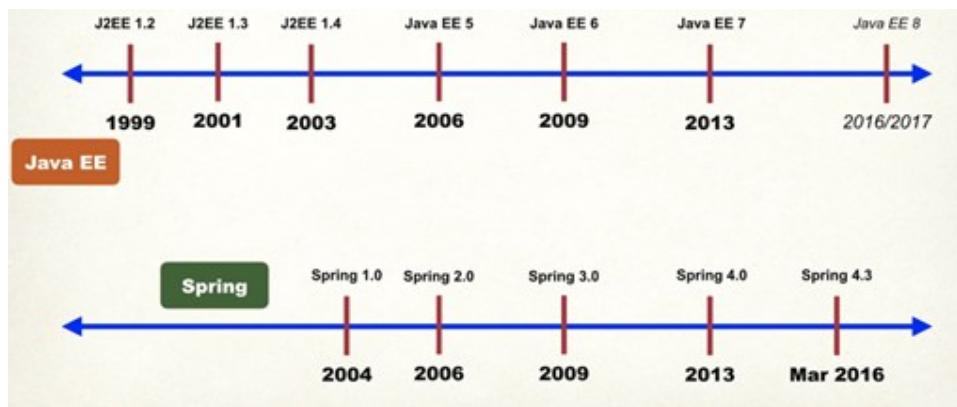
- ◆ Introducción
- ◆ Preparativos de instalación
- ◆ Instalación de Tomcat
- ◆ Conectar Tomcat con Eclipse
- ◆ Instalar Spring sin Maven
- ◆ Extensión de Spring para Eclipse

Introducción

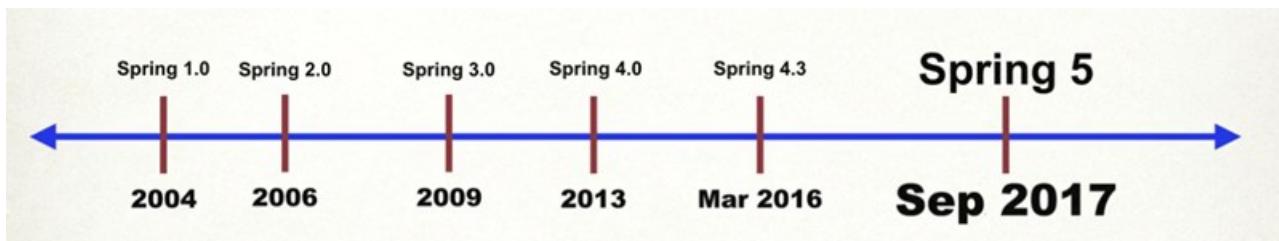
- ◆ Spring es un framework muy popular para el desarrollo de aplicaciones web empresariales basadas en el lenguaje Java.
- ◆ Inicialmente fue una alternativa simple y ligera a J2EE (antiguo Java EE/Jakarta EE).
- ◆ Con el tiempo ha ido adquiriendo una mayor relevancia en la comunidad y actualmente ha 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.



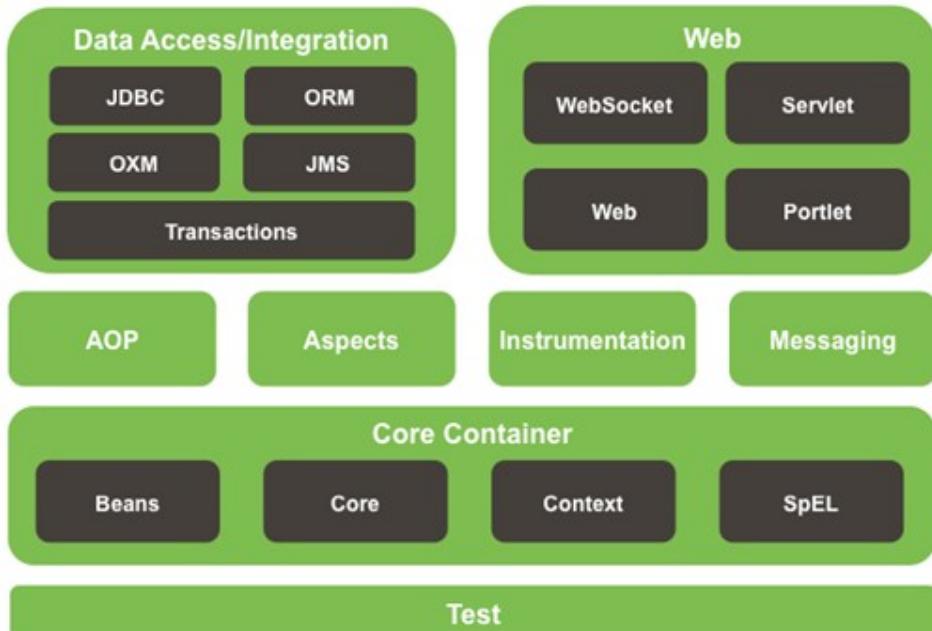
- Los EJB mejoraron mucho en su versión 3.1 (2013) a partir de Java EE 7, pero para entonces Spring ya había cobrado demasiado importancia en la industria.
- Actualmente no hay ninguna característica o funcionalidad importante de Spring que no ofrezca Java EE, pero los desarrolladores prefieren utilizar 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.

Core Container

Beans

Core

Context

SpEL

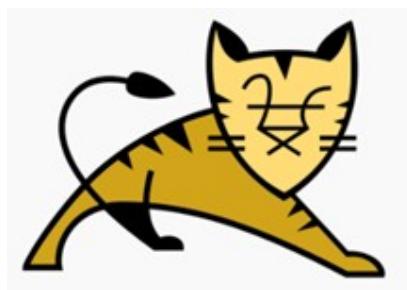
- El *Data Access/Integration* está compuesto por:
 - JDBC: permite trabajar con bases de datos de forma más simple que con las clases clásicas de JDBC.
 - ORM: facilita el mapeo de objetos mediante JPA (Java Persistence API) o Hibernate.
 - JMS: es un sistema de mensajería asíncrono.
 - Transactions: 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 \(pgp, sha512\)](#)

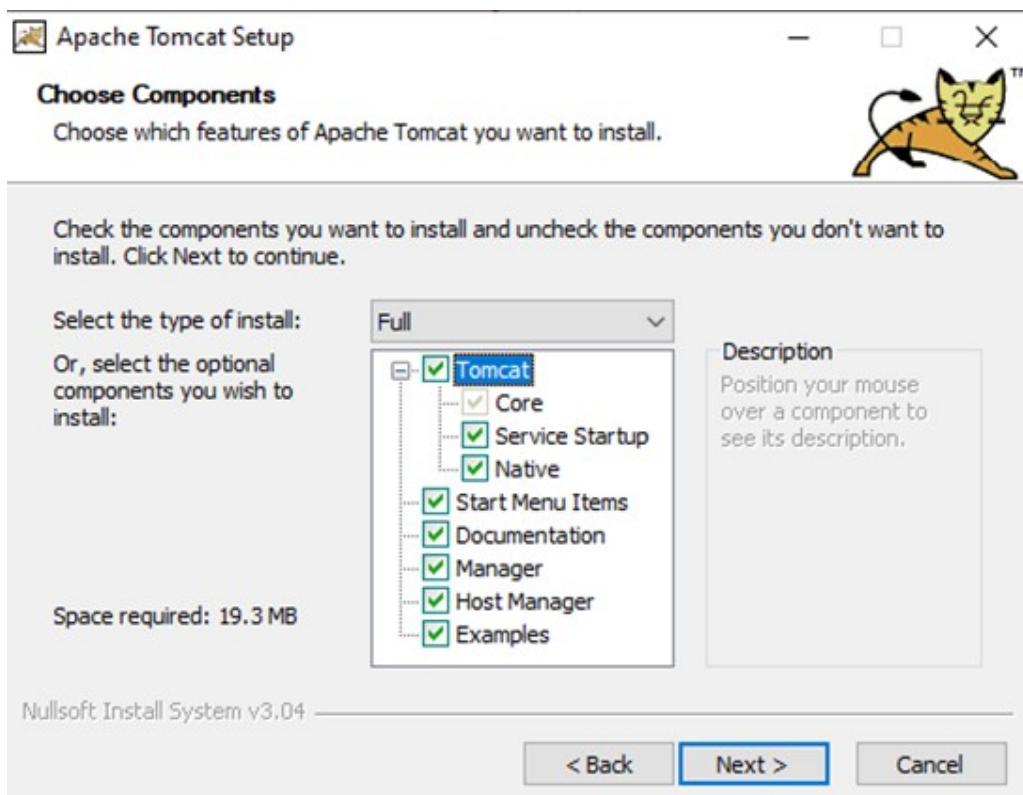
Source Code Distributions

- [tar.gz \(pgp, sha512\)](#)
- [zip \(pgp, sha512\)](#)

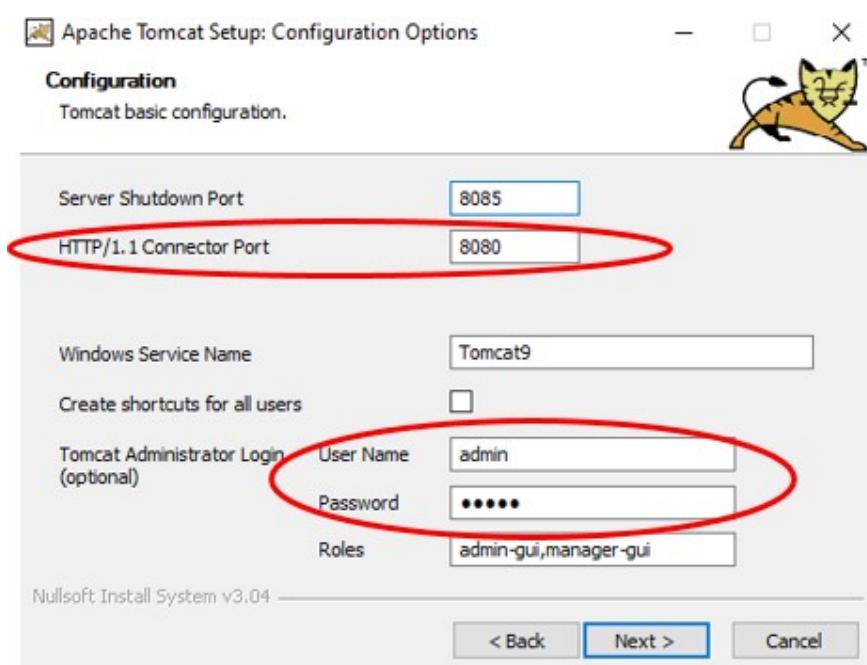
- El procedimiento de instalación es muy simple mediante un asistente.



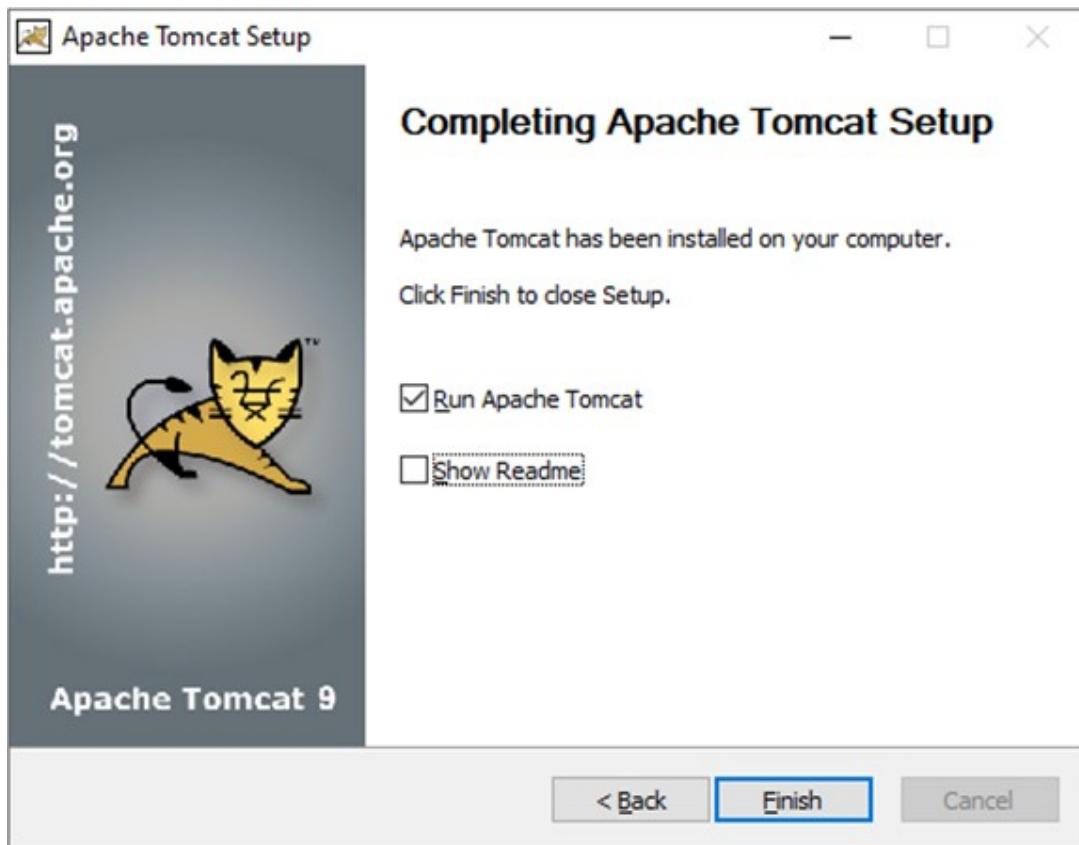
- Es recomendable realizar la instalación completa (Full) y asegurarse que Tomcat se instala como servicio de Windows.



- 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

[Home](#) [Documentation](#) [Configuration](#) [Examples](#) [Wiki](#) [Mailing Lists](#)
[Find Help](#)

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 Data Sources](#)

[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/BUNDLING.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:

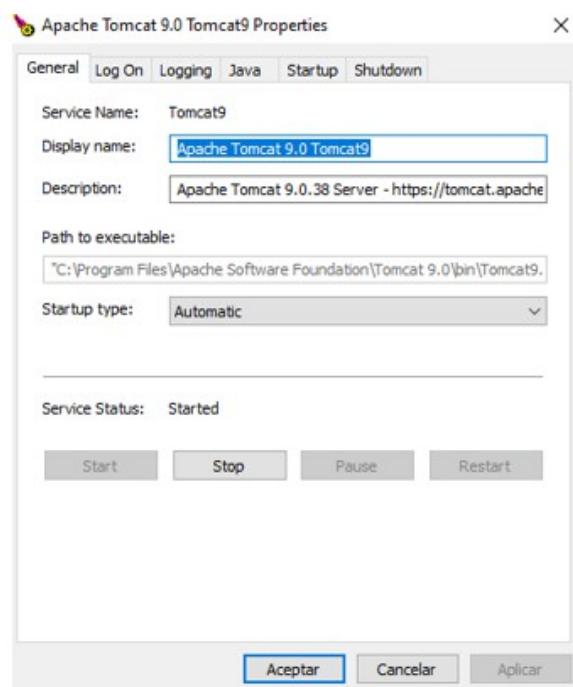
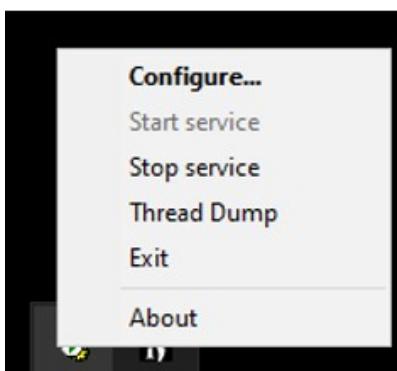
[tomcat-announce](#)
 Important announcements, releases, security vulnerability notifications. (Low volume).

[tomcat-users](#)
 User support and discussion

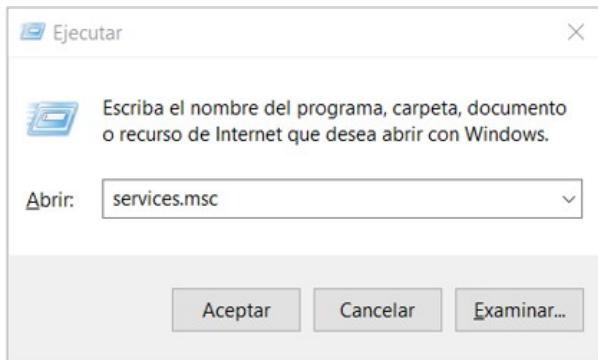
[taglibs-user](#)
 User support and discussion for [Apache Taglibs](#)

[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.



- Tomcat se encontrará en ejecución como servicio de Windows.

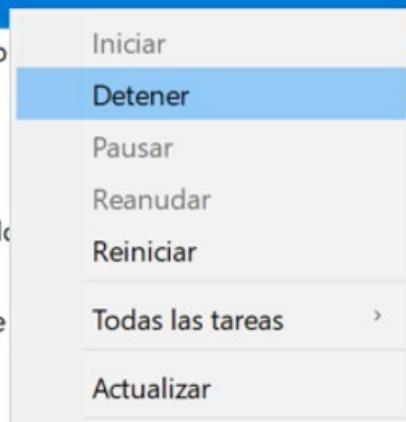


| | | | |
|--|-----------------------------|-------------------|----------------|
| Almacenamiento de datos de usuarios_525fe190 | Controla el a... | Manual | Sistema local |
| Apache Tomcat 9.0 Tomcat9 | Apache Tom... En ejecu... | Automático | Servicio local |
| Aplicación auxiliar de NetBIOS sobre TCP/IP | Proporciona ... En ejecu... | Manual (desen...) | Servicio local |
| Aplicación auxiliar IP | Proporciona ... En ejecu... | Automático | Sistema local |
| Aplicación del sistema COM+ | Administra l... | Manual | Sistema local |
| Archivos sin conexión | El servicio de... | Manual (desen...) | Sistema local |

- Para una correcta integración con Tomcat es preferible parar el servicio (se arrancará de nuevo posteriormente en Eclipse).

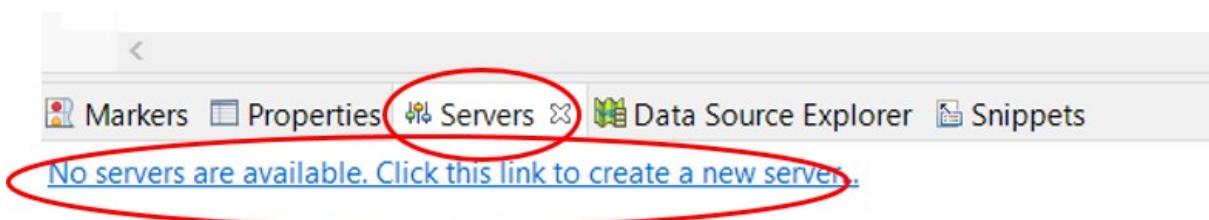
Almacenamiento de datos de usuarios_7a043

- 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

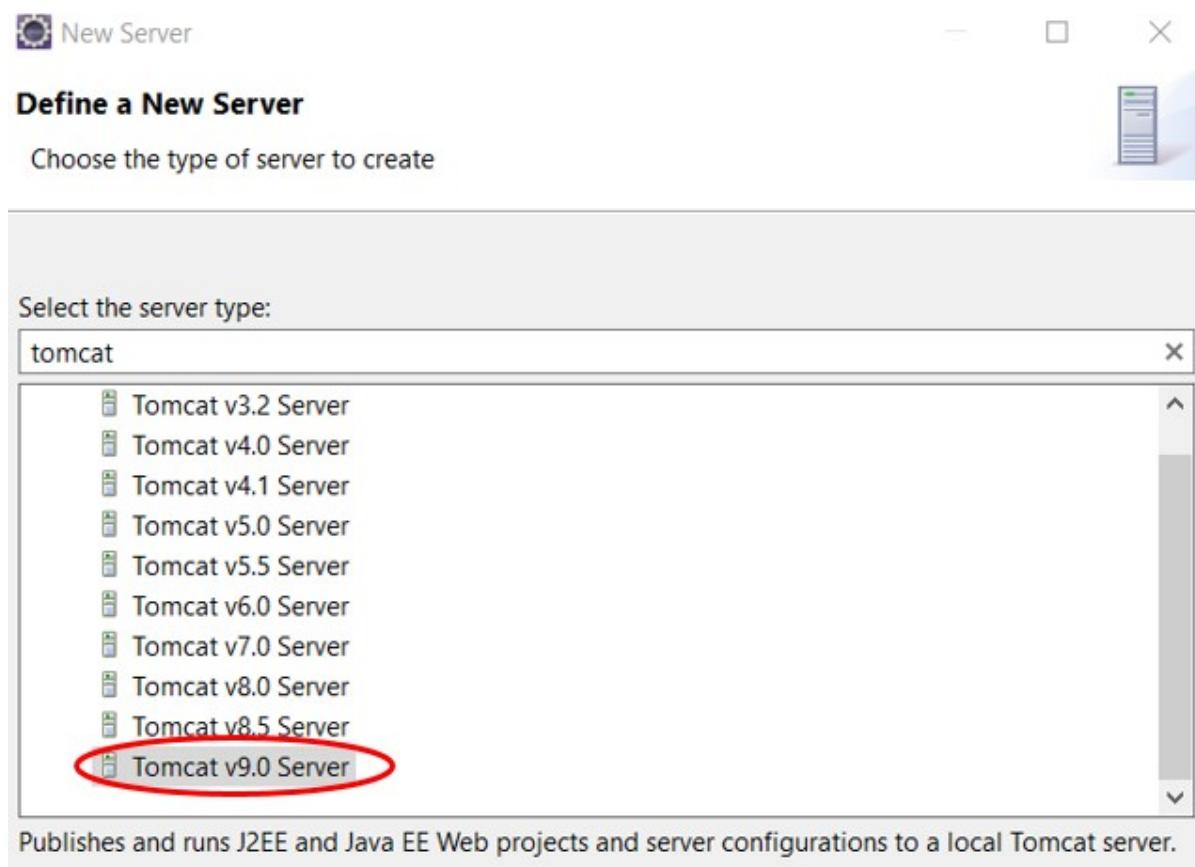


Conectar Tomcat con Eclipse

- 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:

Apache Tomcat v9.0

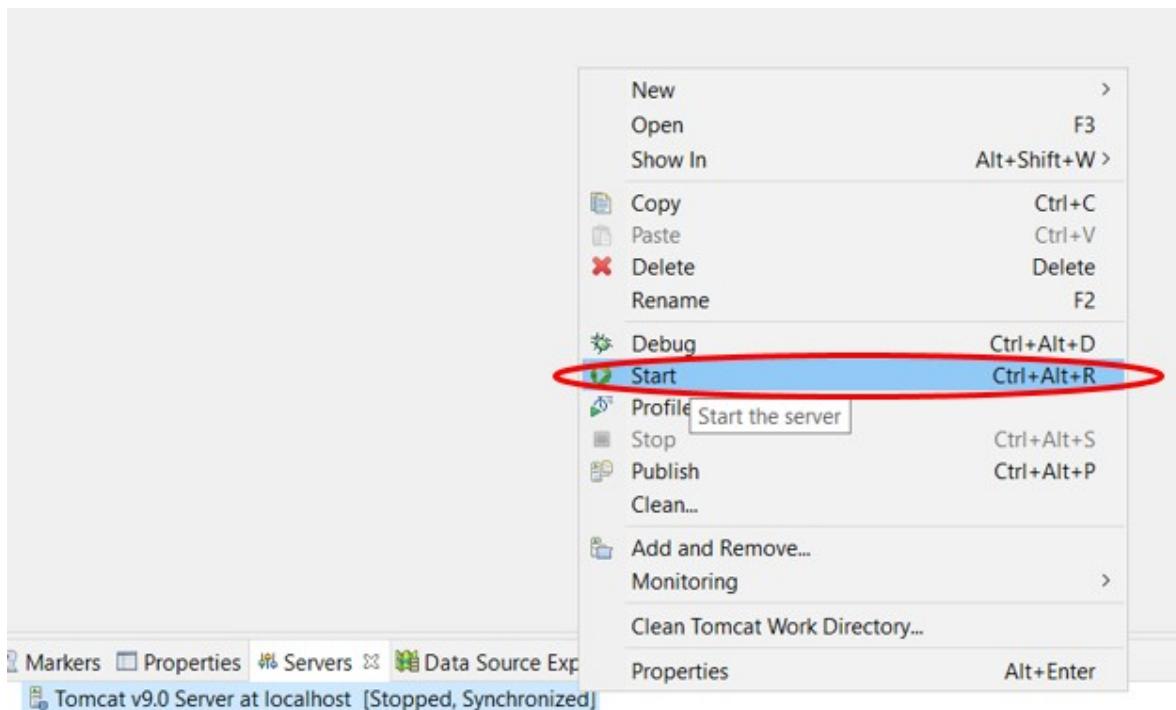
Tomcat installation directory:

C:\Program Files\Apache Software Foundation\Tomcat 9.0

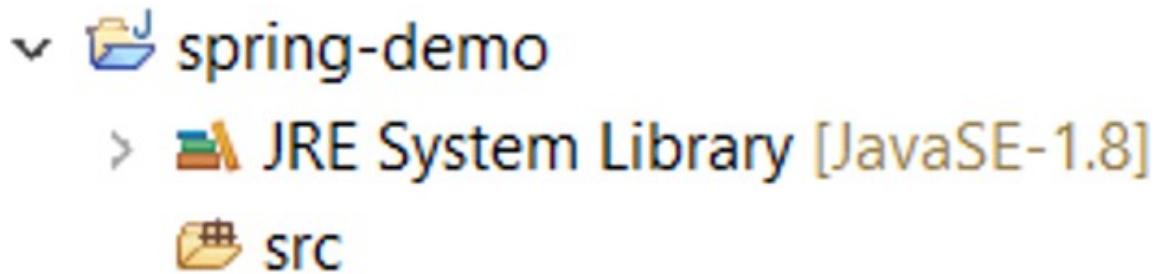
Browse...

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.

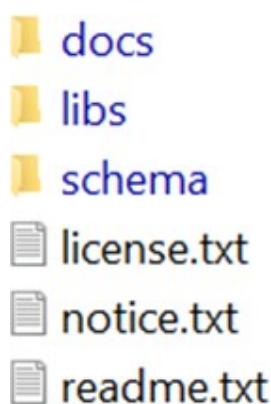


- Spring puede descargarse desde su página oficial.

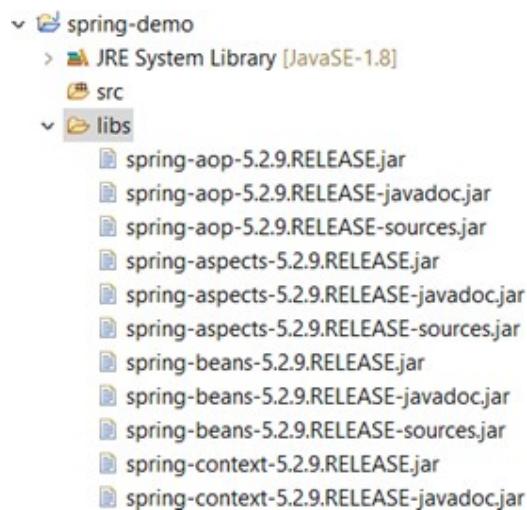
Index of release/org/springframework/spring

| Name | Last modified | Size |
|---------------------------|-------------------|------|
| .. / | | |
| 1.0 / | 27-Apr-2017 02:22 | - |
| 1.0-m4 / | 27-Apr-2017 12:56 | - |
| 1.0-rc1 / | 27-Apr-2017 13:31 | - |
| 1.0.1 / | 27-Apr-2017 18:06 | - |
| 1.1 / | 30-Apr-2017 07:54 | - |
| 1.1-rc1 / | 26-Jan-2017 01:14 | - |
| 1.1-rc2 / | 18-Jan-2017 23:39 | - |
| 1.1.1 / | 27-Apr-2017 08:49 | - |
| 1.1.2 / | 30-Apr-2017 07:56 | - |
| 1.1.3 / | 28-Apr-2017 07:11 | - |
| 1.1.4 / | 28-Apr-2017 21:26 | - |
| 1.1.5 / | 28-Apr-2017 10:31 | - |
| 1.2 / | 27-Apr-2017 13:43 | - |
| 1.2-rc1 / | 27-Apr-2017 13:32 | - |

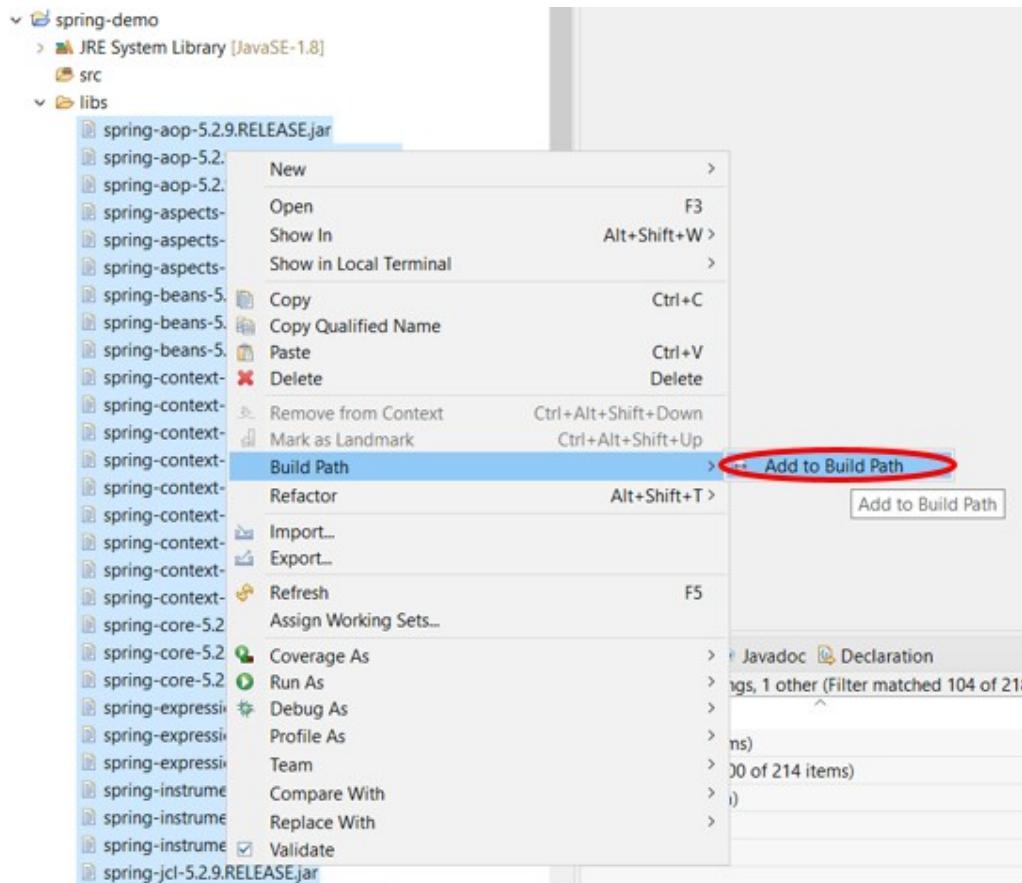
- La versión recomendada a instalar es la última disponible (actualmente 5.x.x RELEASE), seleccionando para descargar el archivo spring-5.x.x-RELEASE-dist.zip
- El directorio más importante del comprimido descargado es *libs*



- 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: All Markets All Categories 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)

Install

- Será necesario seleccionar todos los componentes para instalar.

Eclipse Marketplace

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



< Install More

Confirm >

Finish

Cancel

- 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



< Back

Next >

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();  
    }  
}
```

```

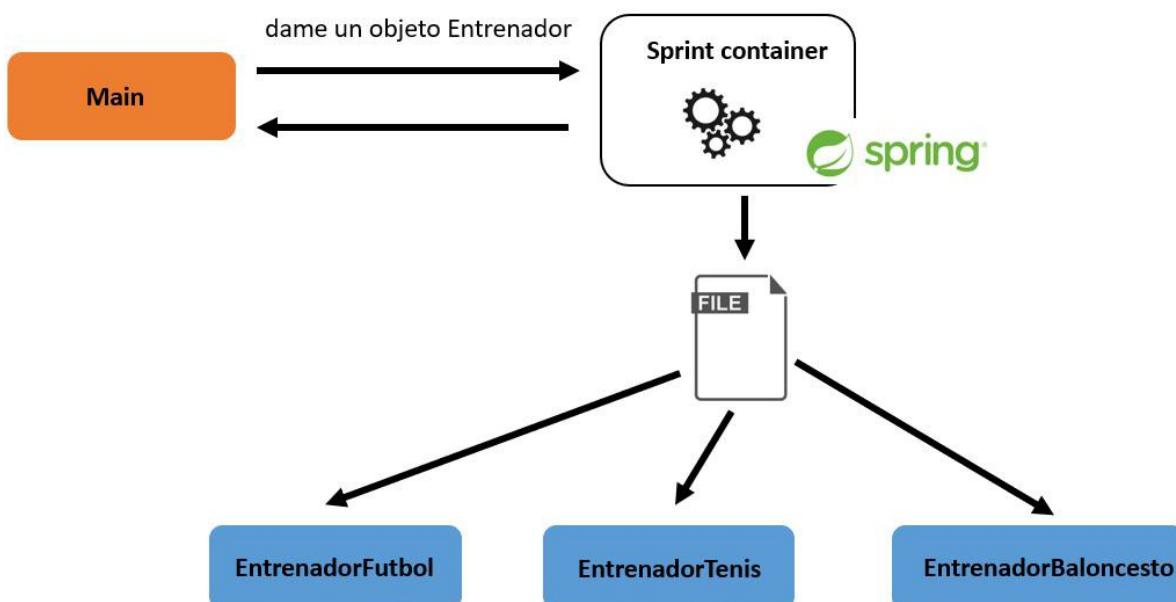
System.out.println(entrenadorFutbol.getEntrenamiento());
System.out.println(entrenadorBaloncesto.getEntrenamiento());
}
}

```

- 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

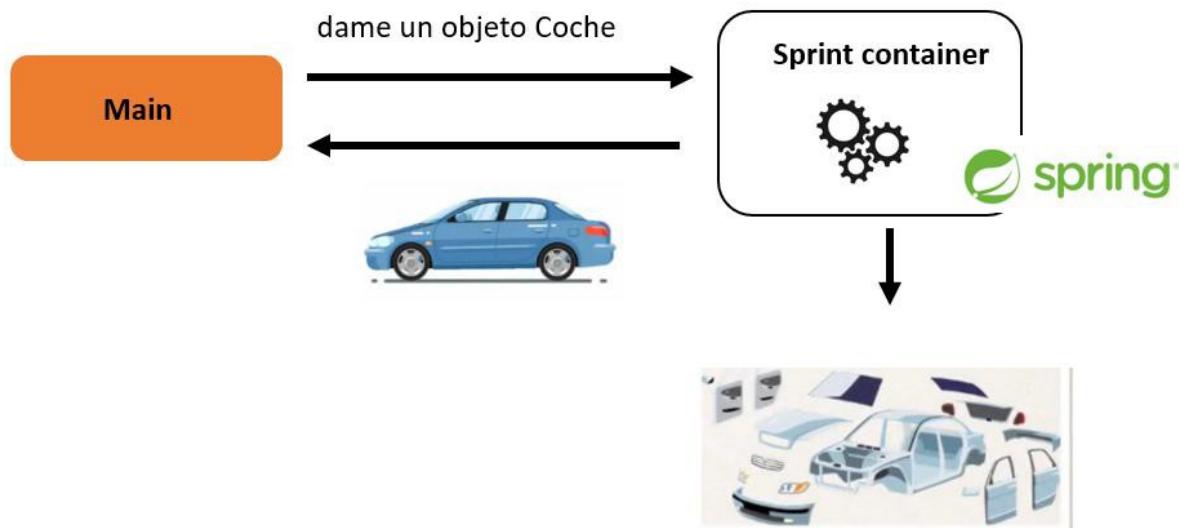


Inyección de dependencias

- ◆ Introducción
- ◆ Inyección de dependencias en Spring con archivo XML
 - Inyección en el constructor
 - Inyección en un método *setter*
 - Inyectar valores literales
 - Inyectar 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
 - Inyectar valores desde un archivo properties

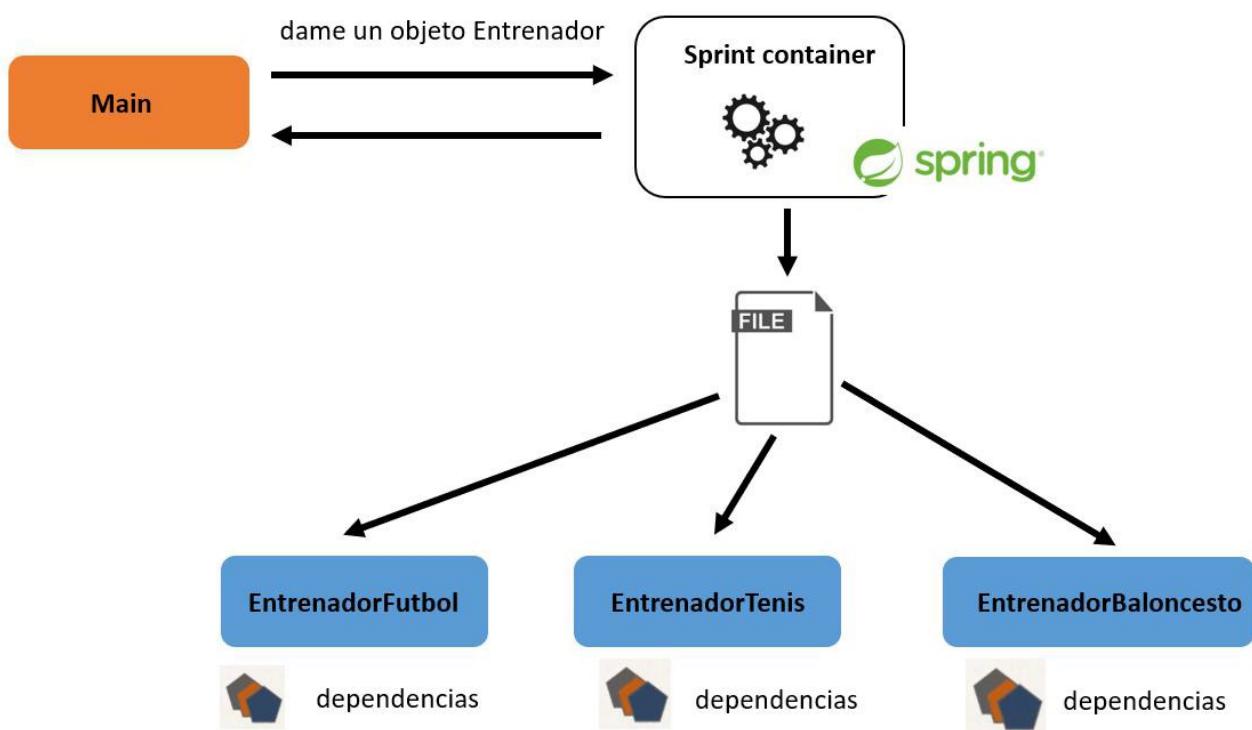
Introducción

- ◆ 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 inyectados (de ahí el concepto de dependencia).
 - El *container* será el encargado de inyectar 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 injectar 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="correoedeprueba@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();
    }
}
```

Injectar 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
email=correoedeprueba@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.example.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 inyectar las dependencias directamente en un atributo, incluido 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;
}
```

Injectar valores desde un archivo properties

- Puede utilizarse la anotación `@Value` para injectar 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

Ámbito y ciclo de vida de un bean



Ámbito y ciclo de vida de un bean

- Ámbito de los beans
- Ciclo de vida de un bean
- Métodos personalizados de inicialización y destrucción

Ámbito de los beans

- El ámbito o scope de los beans hace referencia a cómo son creados los beans.
- Por defecto, el ámbito de un bean es singleton.

```
<!-- bean que no tiene definido ningún ambito y, por tanto, es singleton -->
<bean id="mientrenador" class="com.ejemplo.EntrenadorFutbol">

</bean>
```

- Las características de un bean u objeto con ámbito singleton son:
 - Solamente existe una instancia.
 - Se encuentra cacheado en memoria.
 - Todas las inyecciones del bean desde cualquier parte de la aplicación devuelve la misma referencia al mismo bean (a la única instancia que existe).

```
// todos los entrenadores harán referencia a la misma instancia porque el ámbito del
// bean mientrenador es singleton
Entrenador entrenador1 = context.getBean("mientrenador", Entrenador.class);
Entrenador entrenador2 = context.getBean("mientrenador", Entrenador.class);
Entrenador entrenador3 = context.getBean("mientrenador", Entrenador.class);
```

- Si el ámbito es singleton se dice que el bean es stateless (sin estado).
- El ámbito de un bean se define utilizando el atributo *scope*

```
<!-- en este caso no tiene ningún efecto sobre el bean porque por defecto el ámbito es
singleton -->
<bean id="mientrenador" scope="singleton" class="com.ejemplo.EntrenadorFutbol">

</bean>
```

- Los posibles valores que puede tomar el atributo *scope* son:
 - *singleton* (defecto): crea una instancia única compartida.

- *prototype*: crea una instancia nueva por cada petición. En este caso, el bean no se crea cuando se abre el contexto (como sucede cuando el ámbito es *singleton*), sino cuando se solicita explícitamente el bean.
 - *request*: instancia restringida a una petición HTTP (solamente utilizada para aplicaciones web).
 - *session*: instancia restringida a una sesión HTTP (solamente utilizada para aplicaciones web).
 - *global-session*: instancia restringida a una sesión global HTTP (solamente utilizada para aplicaciones web).
- El scope de un bean también puede establecerse mediante la anotación @Scope

```
// EntrenadorBaloncesto.java

@Component
// el scope pasa a ser prototype y deja de ser singleton
@Scope("prototype")
public class EntrenadorBaloncesto implements Entrenador {

    @Autowired
    private ExperienciaServicio experienciaServicio;

    @Override
    public int getExperienciaEntrenador() {
        return 2;
    }
}
```

```
// MainSpring.java

Entrenador entrenador1 = context.getBean("entrenadorBaloncesto", Entrenador.class);
Entrenador entrenador2 = context.getBean("entrenadorBaloncesto", Entrenador.class);

// apuntan a diferentes direcciones de memoria porque el scope es prototype
System.out.println(entrenador1);
System.out.println(entrenador2);
```

Ciclo de vida de un bean

- Una vez arranca el contexto de Spring, el ciclo de vida de un bean es el siguiente:
 1. Instanciación del bean
 2. Dependencias inyectadas
 3. Procesamiento interno de Spring
 4. Invocación del método personalizado de inicialización del bean

- Cuando el bean tiene ámbito *singleton*, este método de inicialización se invoca al abrir el contexto.
 - Cuando el bean tiene ámbito *prototype*, este método de inicialización se invoca al solicitar o inyectar el bean.
5. Contexto cerrado.
6. Invocación del método personalizado de destrucción del bean
- Cuando el bean tiene ámbito *singleton*, este método de destrucción se invoca al cerrar el contexto.
 - Cuando el bean tiene ámbito *prototype*, este método de destrucción no se invoca automáticamente.
 - Para controlar la destrucción de un bean con ámbito *prototype*, el bean debe implementar la interfaz *DisposableBean* (con el método *destroy*), registrar el método *destroy* en el XML mediante el atributo *destroy-method* y crear una nueva clase (ver el ejemplo de *MyCustomBeanProcessor.java* en la carpeta de tutoriales) que debe registrarse también en el archivo *applicationContext.xml*

Métodos personalizados de inicialización y destrucción

- Los métodos personalizados de inicialización de un bean permiten realizar operaciones relacionadas con la lógica de negocio de la aplicación o la apertura de recursos externos (bases de datos, sockets, archivos, etc.).
- Igualmente, los métodos personalizados de destrucción de un bean permiten cerrar recursos externos abiertos.
- Estos métodos de inicialización o destrucción tienen las siguientes características:
 - Pueden tener cualquier tipo de modificador de acceso (*public*, *protected* o *private*)
 - Pueden retornar cualquier tipo de dato, aunque *void* generalmente es la opción recomendada debido que no será posible capturar el valor de retorno.
 - Puede utilizarse cualquier nombre para el método.
 - No puede recibir argumentos.
- Para crear un método personalizado de inicialización o destrucción hay que indicarlo primero en el archivo *applicationContext.xml* mediante la etiqueta *init-method* o *destroy-method* respectivamente.

```
<!-- los métodos init y destroy deben estar implementados en la clase EntrenadorFutbol --
->
<bean id="mientrenador" init-method="init" destroy-method="destroy"
      class="com.ejemplo.EntrenadorFutbol">

</bean>
```

```
// Entrenador.java
public interface Entrenador {

    public void init();
    public void destroy();
}
```

```
public class EntrenadorFutbol implements Entrenador {

    // este método se invocará cuando se abra el contexto si el ámbito del bean es
    singleton. Si es prototype, entonces se invocará cuando se solicite el bean
    explícitamente
    @Override
    public void init() {
        System.out.println("Inicialización");
    }

    // este método se invocará cuando se cierre el contexto
    @Override
    public void destroy() {
        System.out.println("Destrucción");
    }
}
```

- Las anotaciones para definir métodos personalizaciones de inicialización y destrucción son `@PostConstruct` y `@PreDestroy`

```
@Component
public class EntrenadorBaloncesto implements Entrenador {

    @PostConstruct
    public void init() {
        System.out.println("Inicialización");
    }

    @PreDestroy
    public void destroy() {
        System.out.println("Destrucción");
    }
}
```

- Al igual que sucede con los archivos XML, el método de destrucción no se invocará cuando el *scope* sea *prototype* excepto si se configura y registra correctamente una clase que implemente la interfaz *DisposableBean* (tal y como se comentó anteriormente).

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Configuración con anotaciones



Configuración de Spring con anotaciones

- Introducción
- Definir beans
- Inyección de dependencias
- Inyectar valores desde un archivo properties

Introducción

- El container de Spring puede configurarse a través de anotaciones en lugar de un archivo XML.
- El proceso es el siguiente:
 1. Crear una nueva clase Java con una anotación @Configuration
 2. Añadir la anotación @ComponentScan (opcional si no se pretende realizar un escaneo por los paquetes para buscar beans).
 3. Leer la configuración desde la clase principal.
 4. Obtener los beans desde el container de Spring.
- En primer lugar se crea la nueva clase con la anotación @Configuration y opcionalmente la anotación @ComponentScan

```
// Configuracion.java

@Configuration
@ComponentScan("com.ejemplo")
public class Configuracion {

}

/*
```

Esta clase de Java equivaldría al siguiente archivo de configuración XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

<context:component-scan base-package="com.ejemplo"/>

</beans>
*/
```

- Ahora puede leerse la configuración del container desde otra clase a través de la clase *AnnotationConfigApplicationContext*

- La obtención de los beans se realiza de la misma forma.

```
// MainSpring.java

public class MainSpring {

    public static void main(String[] args) {

        // se carga la configuración del container de Spring
        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext(Configuracion.class);

        Entrenador entrenador = context.getBean("entrenadorBaloncesto", Entrenador.class);
        System.out.println(entrenador);

        context.close();
    }
}
```

Definir beans

- Los beans se definen en un archivo Java de configuración a través de la anotación @Bean sobre métodos que devuelven el bean específico.
 - El nombre del método será el nombre del id del bean.
- Los beans establecidos explícitamente en el archivo Java de configuración mediante la anotación @Bean no requieren de la anotación @Component
 - La anotación @Component en los beans se utiliza principalmente cuando se realiza una escaneo mediante la anotación @ComponentScan en el archivo de configuración Java

```
// Configuracion.java

@Configuration
public class Configuracion {

    // el id del bean será el nombre del método (entrenadorBaloncesto)
    @Bean
    public Entrenador entrenadorBaloncesto() {

        // el método crea un objeto de la clase EntrenadorBaloncesto y lo retorna
        EntrenadorBaloncesto entrenador = new EntrenadorBaloncesto();
        return entrenador;
    }
}
```

}

/*

Esta clase de Java equivaldría al siguiente archivo de configuración XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

<bean id="entrenadorBaloncesto" class="com.ejemplo.EntrenadorBaloncesto">
</beans>
```

- El procedimiento para obtener el bean a través del contexto no varía.

```
public class MainSpring {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext(Configuracion.class);

        // el id del bean es el nombre del método declarado en el archivo de configuración
        // Java
        Entrenador entrenador = context.getBean("entrenadorBaloncesto", Entrenador.class);
        System.out.println(entrenador.getExperienciaEntrenador());

        context.close();
    }
}
```

Inyección de dependencias

- La inyección de dependencias se realiza directamente en el paso de parámetros del constructor, invocando al método que devuelve el bean a inyectar.

```
// EntrenadorBaloncesto.java

// no se establece la anotación @Component
public class EntrenadorBaloncesto implements Entrenador {

    private ExperienciaServicio experienciaServicio;

    public EntrenadorBaloncesto(ExperienciaServicio experienciaServicio) {
        this.experienciaServicio = experienciaServicio;
    }
}
```

```
@Override
public int getExperienciaEntrenador() {
    return experienciaServicio.getExperiencia();
}
```

```
// Configuracion.java

@Configuration
public class Configuracion {

    // el id del bean es experienciaServicio
    @Bean
    public ExperienciaServicio experienciaServicio() {
        ExperienciaServicio experienciaServicio = new PocaExperienciaServicio();
        return experienciaServicio;
    }

    // el id del bean es entrenadorBaloncesto
    @Bean
    public Entrenador entrenadorBaloncesto() {

        // se inyecta la dependencia experienciaServicio al bean entrenadorBaloncesto
        Entrenador entrenador = new EntrenadorBaloncesto(experienciaServicio());
        return entrenador;
    }
}
```

- El procedimiento para obtener el bean a través del contexto no varía.

```
// MainSpring.java

public class MainSpring {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext(Configuracion.class);

        Entrenador entrenador = context.getBean("entrenadorBaloncesto", Entrenador.class);
        System.out.println(entrenador.getExperienciaEntrenador());

        context.close();
    }
}
```

Injectar valores desde un archivo properties

- Para cargar un archivo properties desde un archivo de configuración Java se utiliza la anotación @PropertySource
- Seguidamente se inyecta en los atributos de la clase mediante la anotación @Value y la sintaxis \${}

```
# datos.properties
email=correodeprueba2@mail.com
equipo=Powerhouse
```

```
// Entrenador.java

public interface Entrenador {
    public String getEmail();
    public String getEquipo();
}
```

```
// EntrenadorBaloncesto.java

public class EntrenadorBaloncesto implements Entrenador {

    private String email;
    private String equipo;

    // el constructor espera recibir los valores de email y equipo cargados desde el archivo
    // properties
    public EntrenadorBaloncesto(String email, String equipo) {
        this.email = email;
        this.equipo = equipo;
    }

    @Override
    public String getEmail() {
        return email;
    }

    @Override
    public String getEquipo() {
        return equipo;
    }
}
```

```
// Configuracion.java

@Configuration
// se carga el archivo de configuración, que se encuentra en la ruta del classpath y su
nombre es datos.properties
@PropertySource("classpath:datos.properties")
public class Configuracion {

    // se inyecta el valor de la clave email del archivo properties en el atributo email
    @Value("${email}")
    private String email;

    // se inyecta el valor de la clave equipo del archivo properties en el atributo equipo
    @Value("${equipo}")
    private String equipo;

    @Bean
    public Entrenador entrenadorBaloncesto() {

        // los valores leídos desde el archivo properties son pasados al constructor de la clase
        Entrenador entrenador = new EntrenadorBaloncesto(email, equipo);
        return entrenador;
    }
}
```

```
public class MainSpring {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext(Configuracion.class);

        Entrenador entrenador = context.getBean("entrenadorBaloncesto", Entrenador.class);
        System.out.println(entrenador.getEmail());
        System.out.println(entrenador.getEquipo());

        context.close();
    }
}
```

- Sin embargo, los valores cargados desde el archivo properties pueden ser inyectados directamente en cualquier bean, por lo que es más simple hacerlo allí que desde el archivo de configuración Java.

```
// Configuracion.java

@Configuration
@PropertySource("classpath:datos.properties")
public class Configuracion {

    @Bean
    public Entrenador entrenadorBaloncesto() {
        Entrenador entrenador = new EntrenadorBaloncesto();
        return entrenador;
    }
}
```

```
// EntrenadorBaloncesto.java

public class EntrenadorBaloncesto implements Entrenador {

    // EntrenadorBaloncesto es un bean, por lo que puede realizarse la inyección en sus
    // atributos
    @Value("${email}")
    private String email;

    @Value("${equipo}")
    private String equipo;

    @Override
    public String getEmail() {
        return email;
    }

    @Override
    public String getEquipo() {
        return equipo;
    }
}
```

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Introducción a Spring MVC

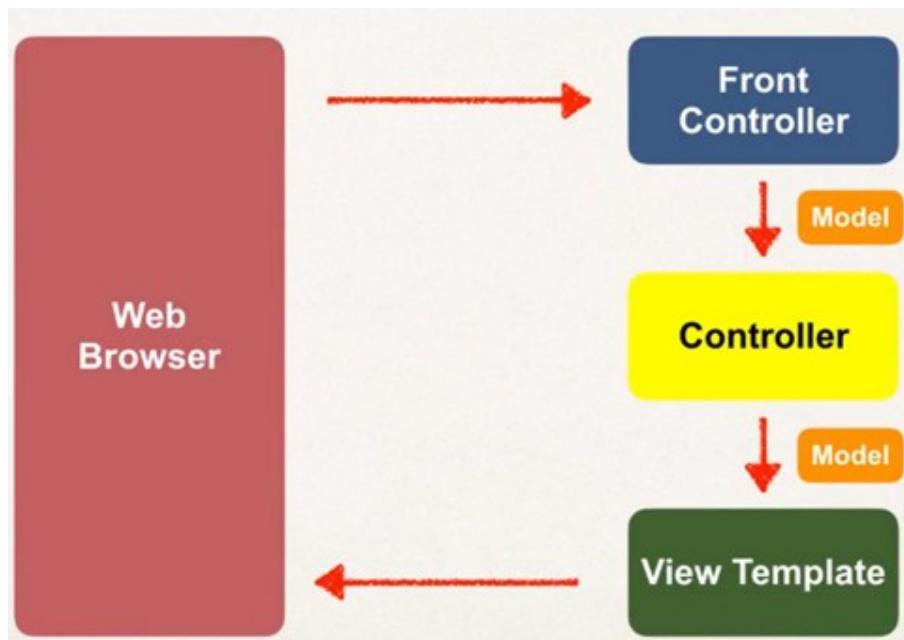


Introducción a Spring MVC

- Introducción
- Primer proyecto con Spring MVC
- Despliegue del proyecto

Introducción

- Spring MVC es un framework para construir aplicaciones web con Java utilizando el patrón de diseño MVC (Modelo - Vista/Plantilla - Controlador) y las ventajas proporcionadas por el control de inversión y la inyección de dependencias.

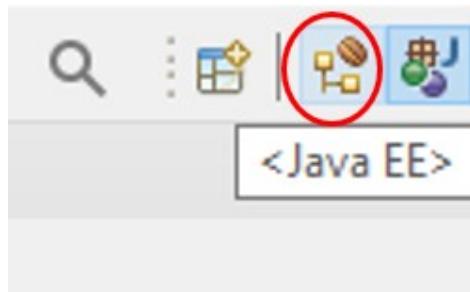


- Las diferencias partes que constituyen el patrón MVC son:
 - Controlador: es el elemento que gestiona las peticiones recibidas desde un navegador web o cliente HTTP. Spring recibe en primer lugar las peticiones en un controlador frontal (Front Controller, implementado por la clase DispatcherServlet de Spring) y posteriormente las redirige hacia el controlador donde el desarrollador implementa el código necesario para manejar las solicitudes.
 - Modelo: hace referencia a los datos necesario para manejar la solicitud y devolver la respuesta al cliente. El controlador crea el modelo generalmente a través de los datos devueltos por una base de datos.
 - Vista/Plantilla: la vista o plantilla se encarga de dar formato a la respuesta devuelta al cliente (generalmente en formato HTML) a través de un motor de plantillas (ej: JSP, JSTL, Thymeleaf, Freemarker, Velocity, Mustache, etc).
- Los componentes de una aplicación con Spring MVC son:
 - Un conjunto de páginas web que definen los componentes gráficos de la aplicación.

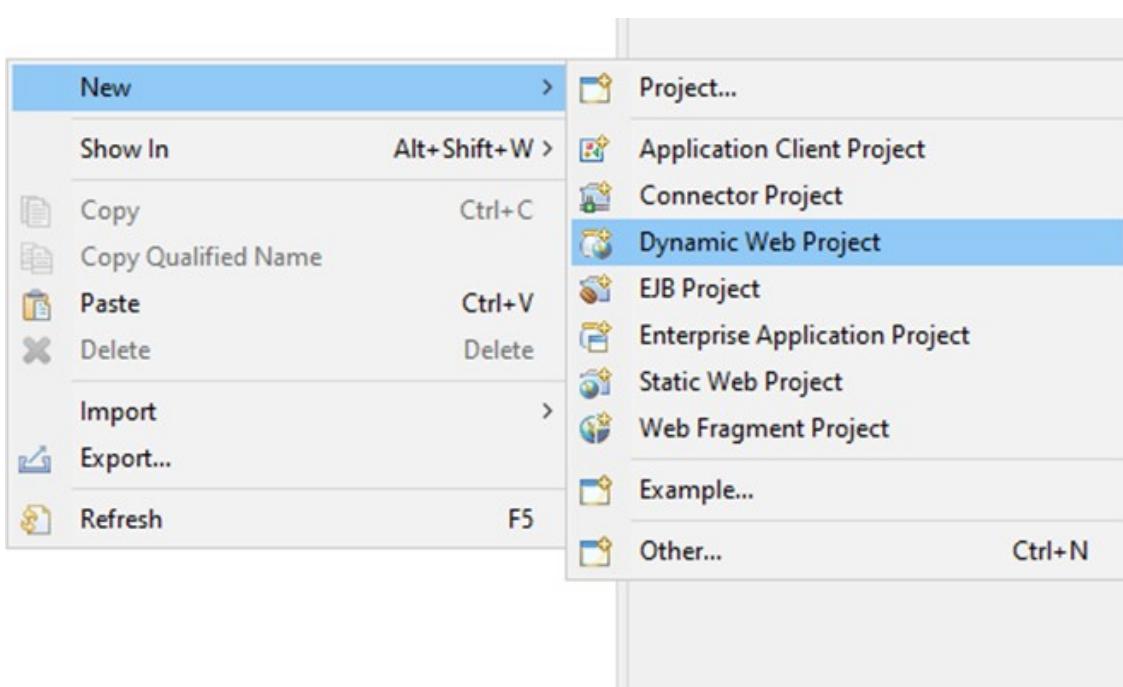
- Una colección de beans (controladores, servicios, etc).
- Un archivo de configuración (XML o anotaciones Java).

Primer proyecto con Spring MVC

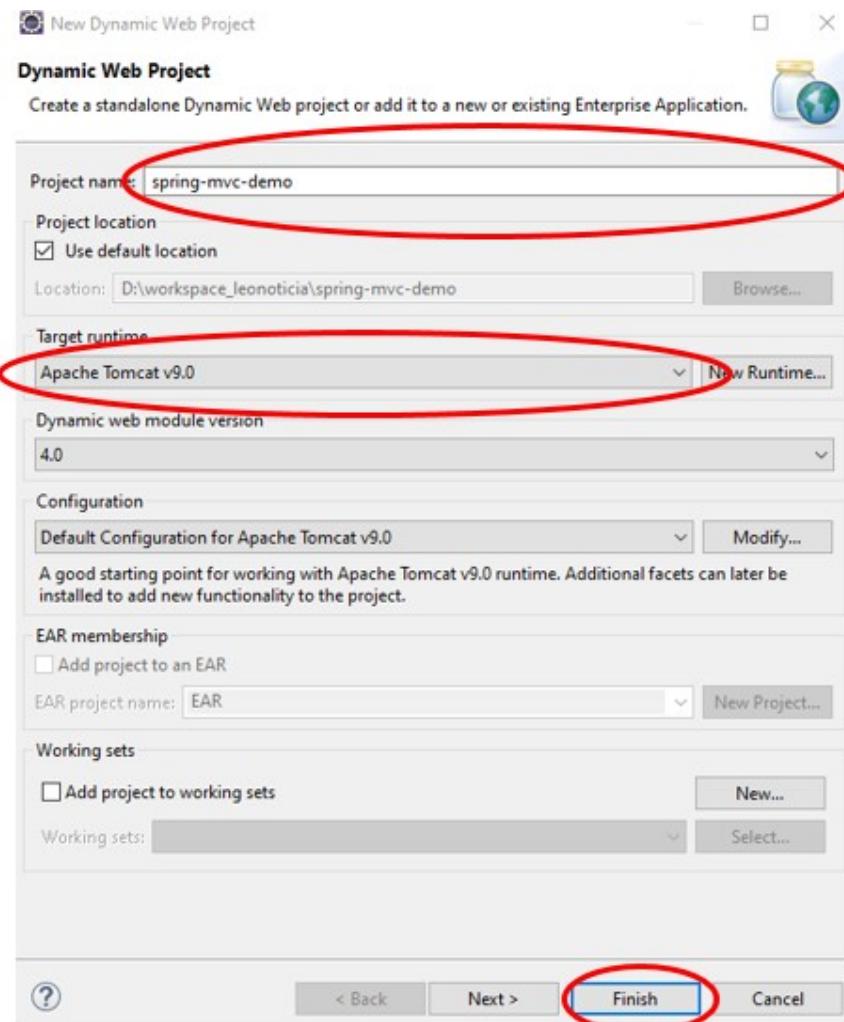
- Spring MVC requiere de Tomcat, un IDE (como Eclipse) y generalmente también un conector que integre Tomcat con el IDE.
- Seguidamente es necesario crear un proyecto en Eclipse, descargar las librerías de Spring MVC y realizar una serie de configuraciones.
- Es buena práctica realizar este proceso solamente una vez y una vez terminado, guardar todo el proyecto para reutilizarlo en futuros programas.
- En primer lugar es recomendable cambiar la perspectiva de Eclipse a Java EE desde alguna de las siguientes opciones:
 - Window -> Perspective -> Open Perspective -> Java EE
 - Pulsando el botón de Java EE en la esquina superior derecha de la ventana.



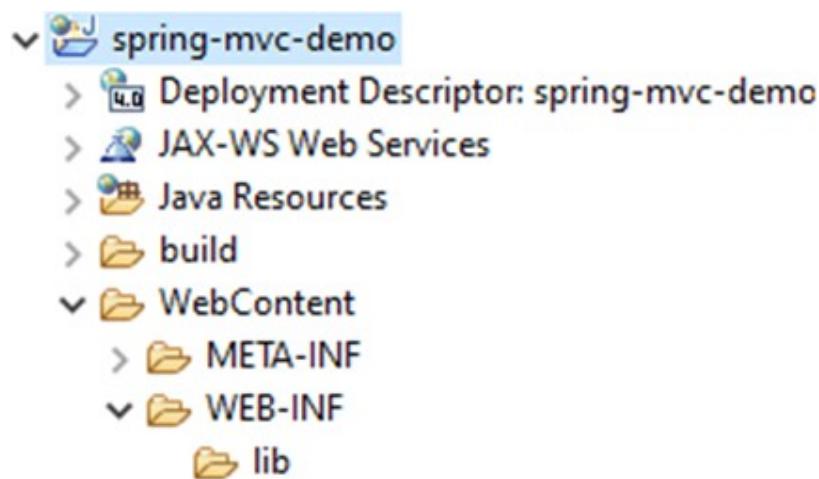
- El tipo de proyecto a crear en Eclipse para trabajar con Spring MVC es *Dynamic Web Project*.



- En la ventana siguiente solamente será necesario establecer el nombre del proyecto, aunque debe comprobarse que el servidor a utilizar será Tomcat. A continuación, se pulsa en *Finish*

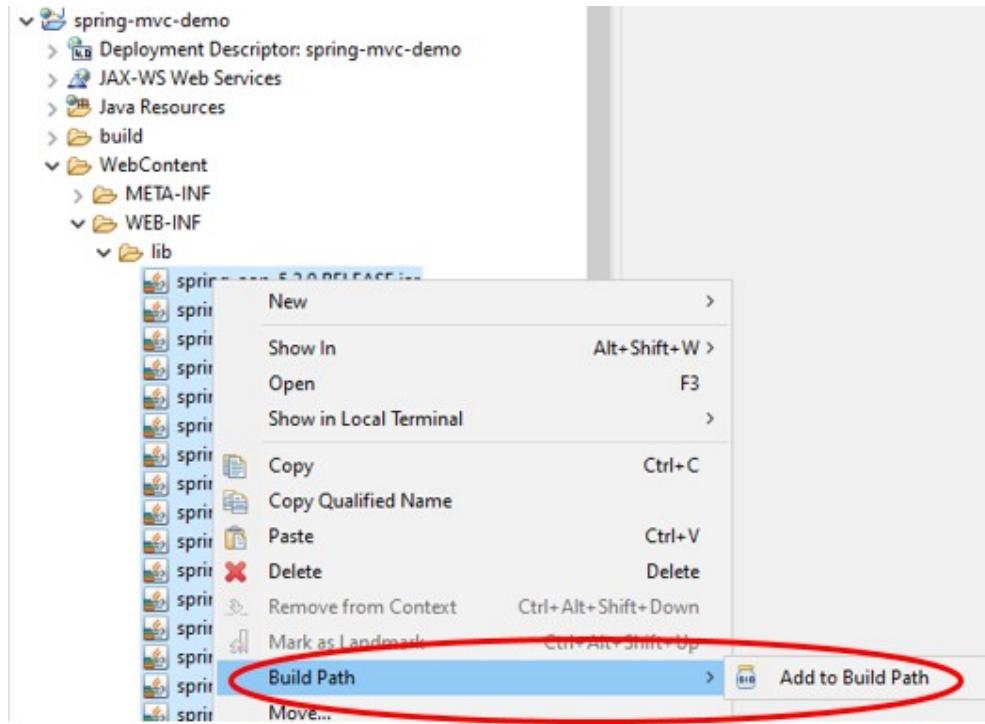


- El nuevo proyecto aparecerá en el explorador de proyectos.

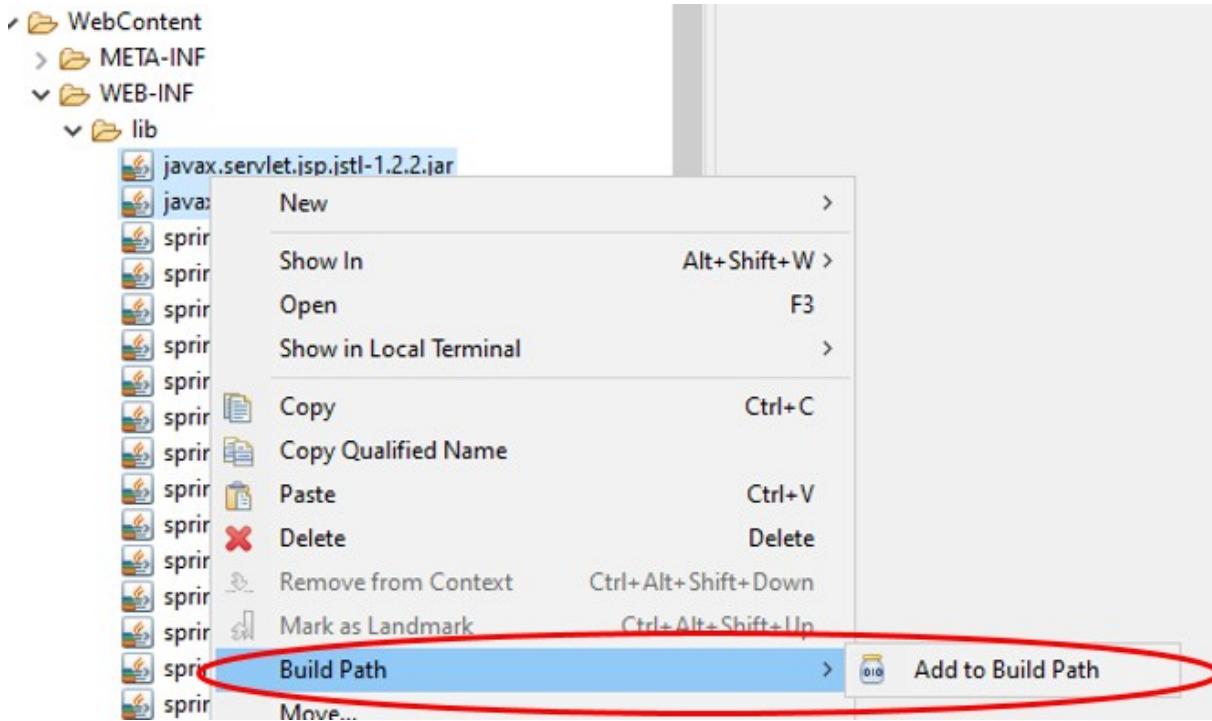


- El siguiente paso es descargar las librerías de Spring (archivos JAR) en su última versión y copiarlas dentro del directorio del proyecto WebContent/WEB-INF/lib

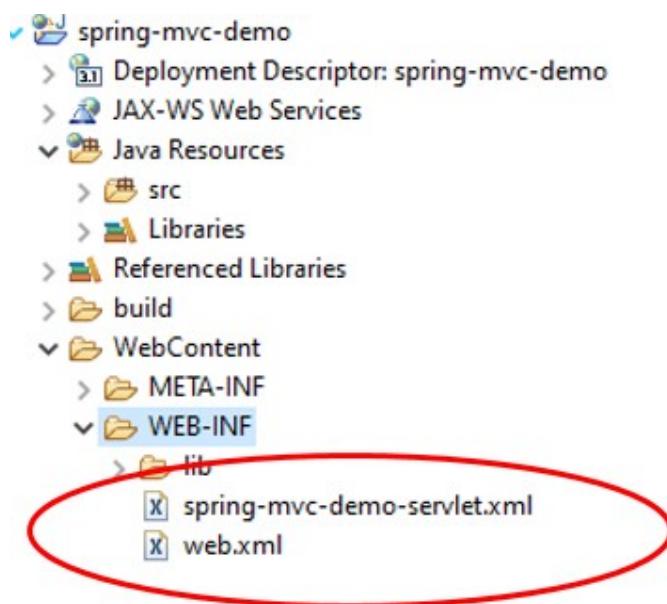
- Una vez copiados todos los JAR, se añaden al *Build Path*, seleccionando todos los archivos y, a continuación, pulsando el botón derecho -> *Build Path* -> *Add to Build Path*.



- En este primer proyecto con Spring MVC se utilizarán páginas JSP como plantillas. Por tanto, también será necesario integrar las librerías JAR de JSP en el proyecto. Los archivos necesarios son:
 - [javax.servlet.jsp.jstl-1.2.2](#)
 - [javax.servlet.jsp.jstl-api-1.2.2](#)
- Puede descargarse desde el repositorio oficial de Maven y también están disponibles en el directorio *utilidades/spring-mvc-demo/lib* (versión 1.2.1 y versión 1.2.2, pero es recomendable tomar la 1.2.2).
- De la misma forma que anteriormente, se copiarán al directorio *WebContent/WEB-INF/lib* y se añadirán al *Build Path*.



- Seguidamente se accede al directorio *utilidades/spring-mvc-demo* y se copian los siguientes archivos en el directorio del proyecto *WebContent/WEB-INF*:
 - spring-mvc-demo-servlet.xml*
 - web.xml*



- El archivo *web.xml* no es realmente específico de Spring, sino de aplicaciones web con Java EE.

```
<!-- configuración del DispatcherServlet (Front Controller) -->
<!-- pueden declararse varios servlet -->
<servlet>

    <!-- nombre del servlet (puede ser cualquier nombre) -->
    <servlet-name>dispatcher</servlet-name>

    <!-- ubicación del DispatcherServlet en las librerías de Spring -->
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!-- parámetros de configuración del DispatcherServlet -->
    <init-param>

        <!--
            indica que la configuración del contexto se encuentra en el archivo
            /WEB-INF/spring-mvc-demo-servlet.xml
        -->
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>
    </init-param>
```

- La estructura del archivo spring-mvc-demo-servlet.xml es similar a la ya descrita por el applicationContext.xml, con algunas etiquetas adicionales para el soporte, formateo y validación, así como la configuración de plantillas.

```
<!-- escaneo de beans en el paquete com.ejemplo -->
<context:component-scan base-package="com.ejemplo" />

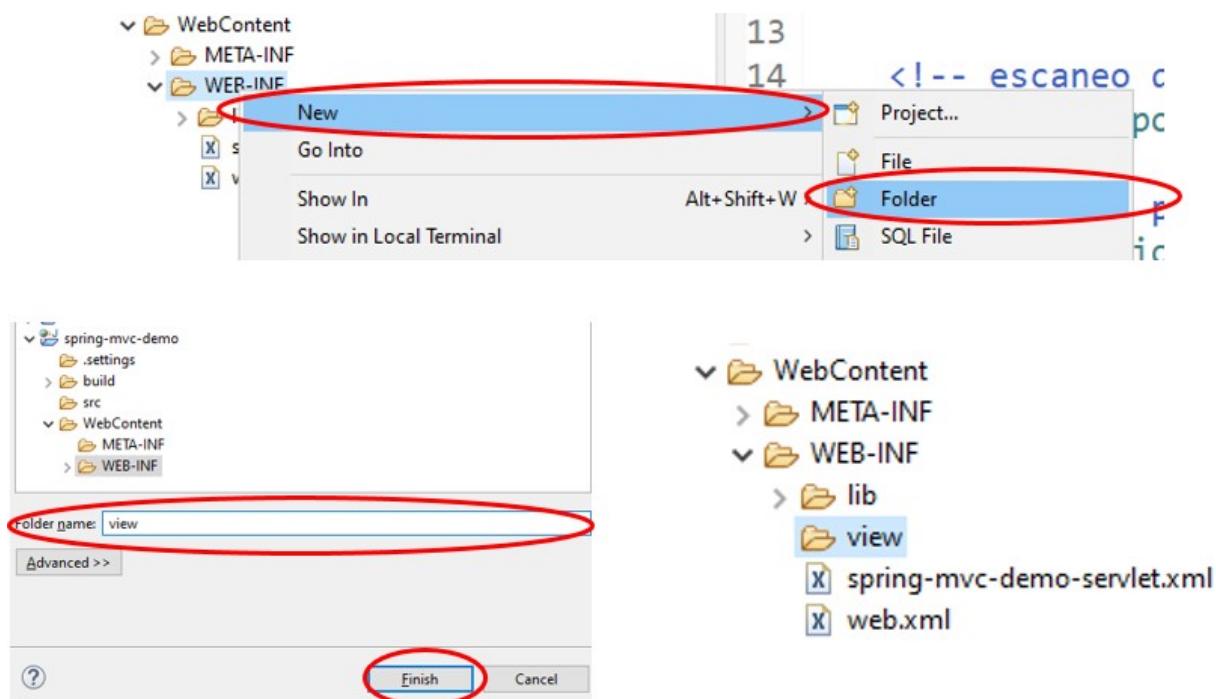
<!-- soporte para la conversión, formateo y validación en Spring -->
<mvc:annotation-driven/>

<!-- definición de un resolver para las plantillas -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

    <!-- las plantillas se ubicarán en el directorio /WEB-INF/view/ -->
    <property name="prefix" value="/WEB-INF/view/" />

    <!-- las plantillas tendrán la extensión jsp -->
    <property name="suffix" value=".jsp" />
</bean>
```

- Será necesario crear el directorio de las plantillas /WEBINF/view.

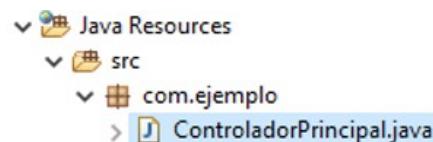


- A continuación puede crearse el primer controlador y plantilla. El proceso es el siguiente:
 - Creación de una clase Controlador.
 - Creación de un método Controlador.
 - Adición de un mapeo al método del Controlador.
 - Retorno del nombre de la plantilla.
 - Implementación de la plantilla.
- Todo el código Java debe crearse dentro de:
 - *Java Resources -> src*
- La clase Controlador utiliza la anotación `@Controller`
- La anotación `@Controller` soporta el escaneo automático en el paquete especificado, por lo que no es necesario establecer el Controlador en el archivo de configuración.

```
@Controller
public class ControladorPrincipal {

    // este método gestionará todas los tipos de peticiones (GET, POST, ...)
    // en la ruta /
    @RequestMapping("/")
    public String index() {

        // devolverá el contenido de la plantilla /WEB-INF/view/index.jsp
        return "index";
    }
}
```



- A continuación se crea la plantilla JSP en el directorio /WEB-INF/view/

index.jsp

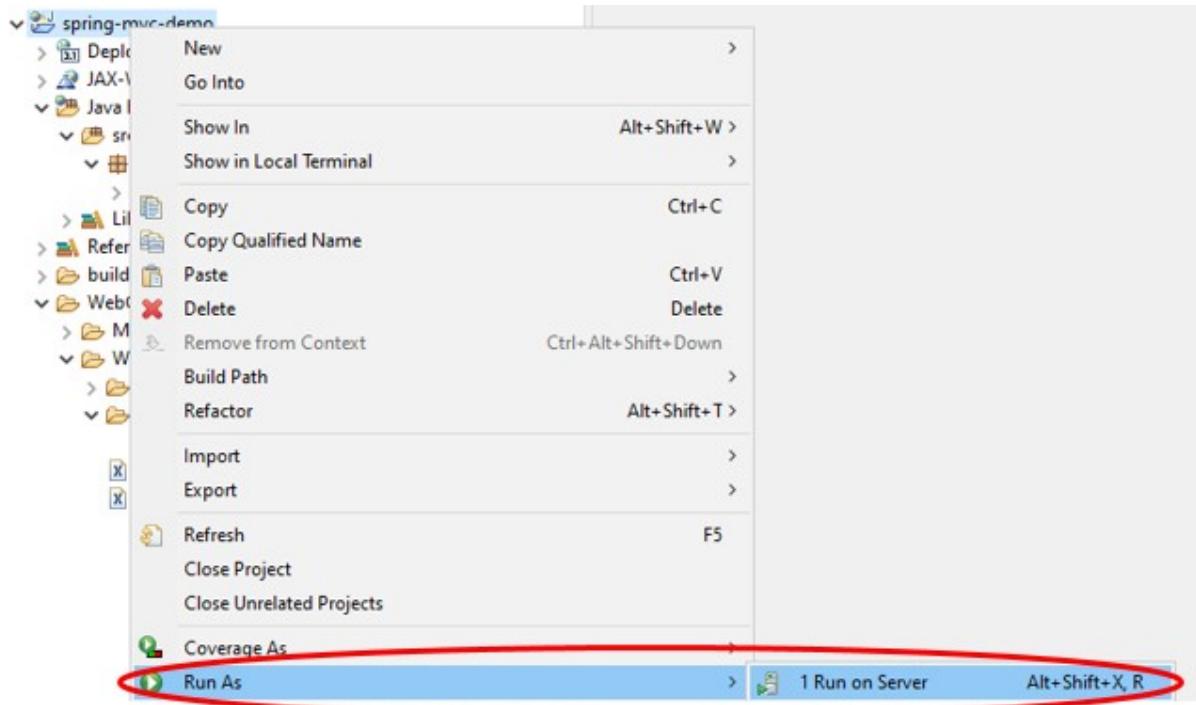
```
1 <!DOCTYPE html>
2 <html>
3 <head> </head>
4 <body>
5     Hello world!
6 </body>
7 </html>
```

WebContent

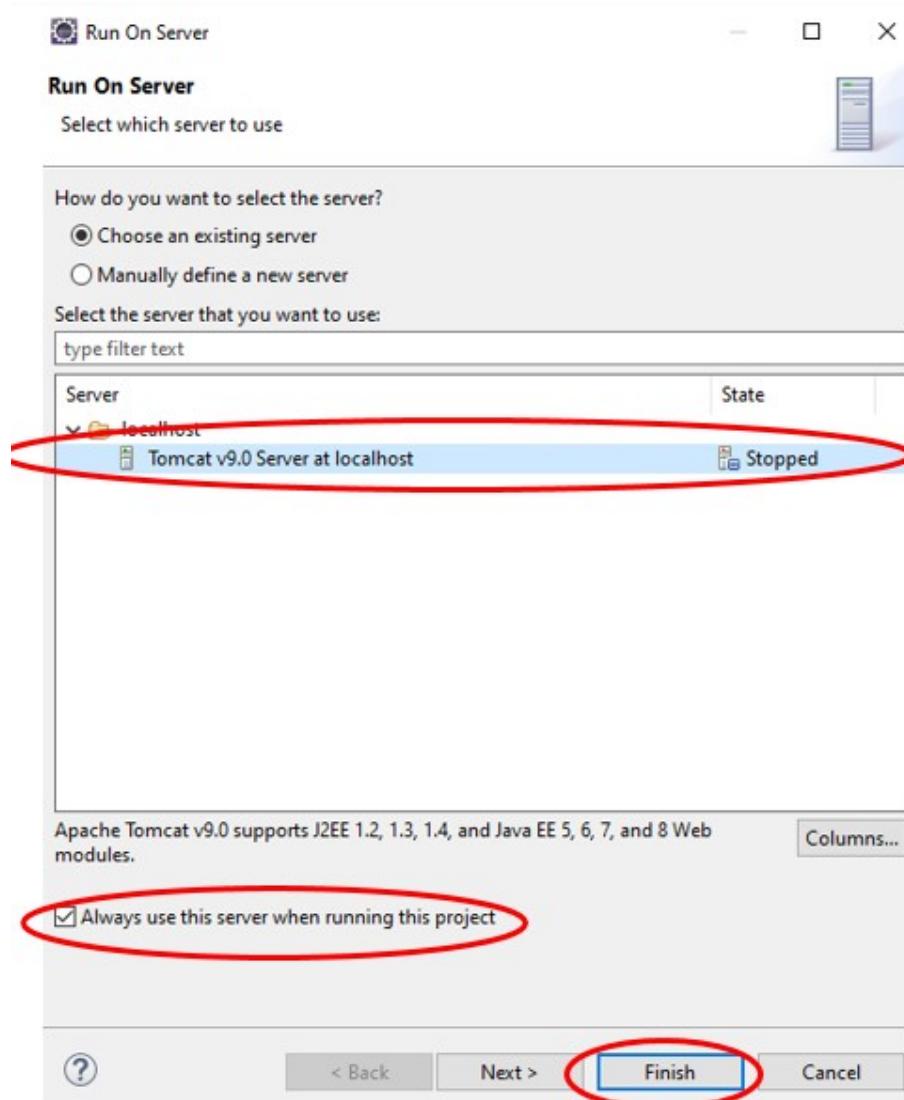
- META-INF
- WEB-INF
- lib
- view

index.jsp

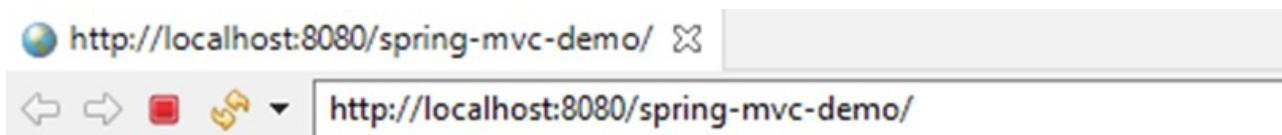
- Por último, es el momento de ejecutar la aplicación.
- Para ello se pulsa el botón derecho sobre el proyecto y, a continuación:
 - Run -> Run as Server



- En el asistente de ejecución se selecciona el servidor de Tomcat, se selecciona la casilla *Always use this server when running this project* y finalmente se pulsa en el botón *Finish*.

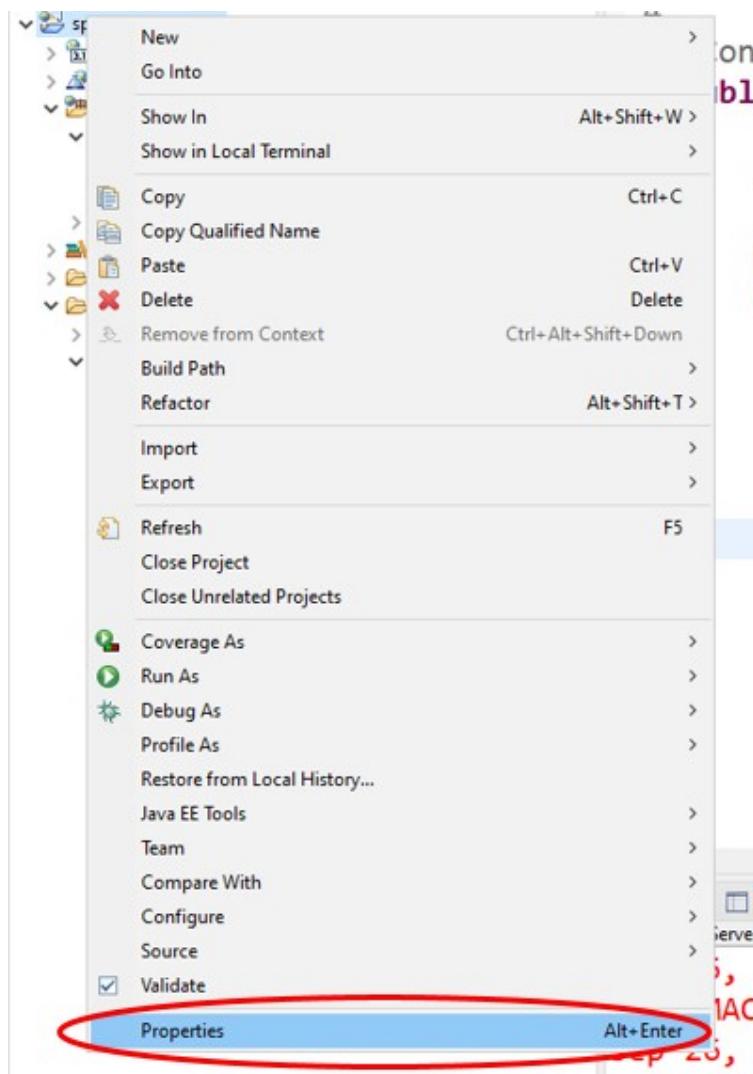


- Si todo es correcto, automáticamente se abrirá un navegador interno de Eclipse con el contenido de la plantilla para la ruta /
- Por defecto la ruta colgará del nombre del proyecto (*spring-mvc-demo* en este caso).

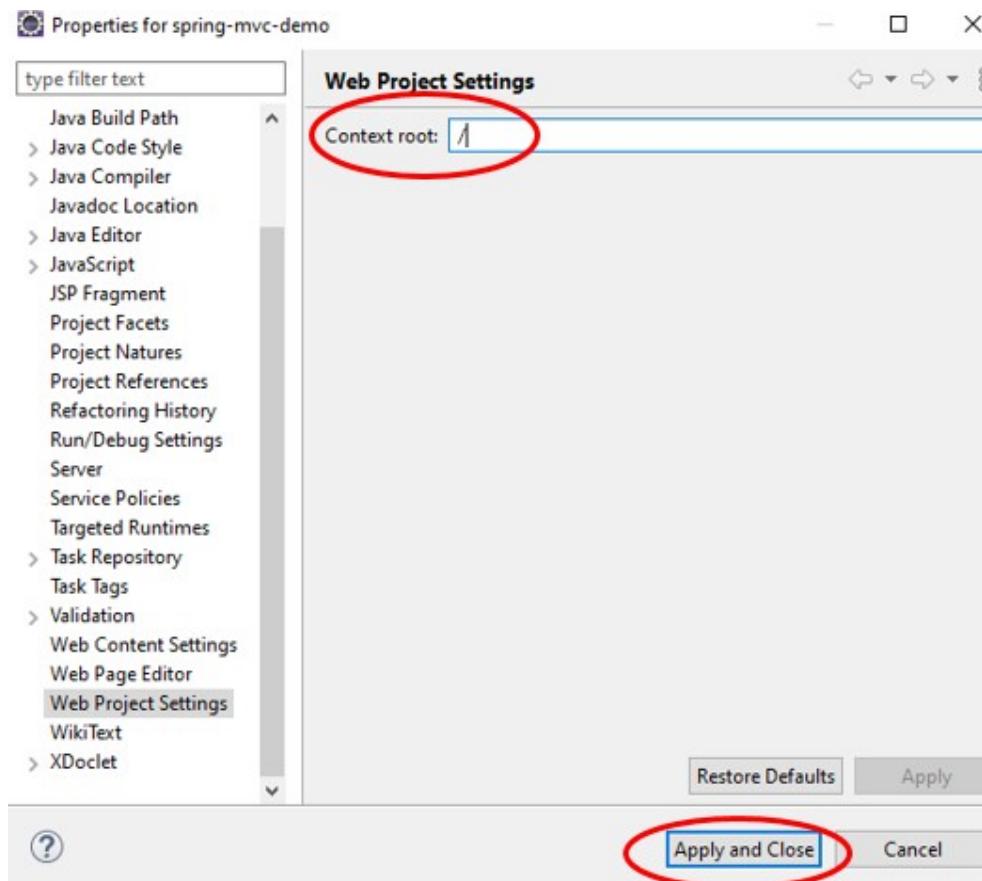


- Para cambiar la ruta raíz a las propiedades del proyecto:

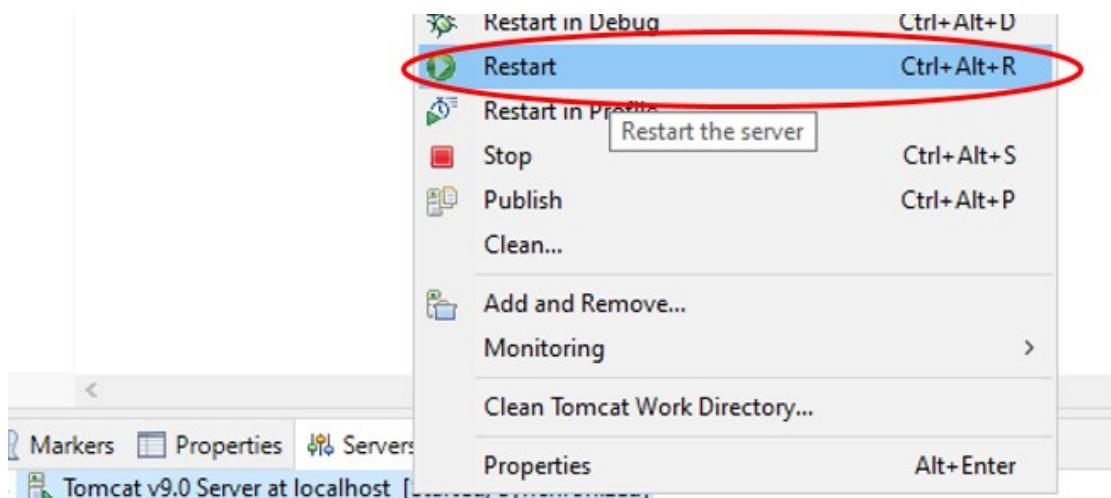
- Botón derecho en el proyecto -> *Properties*



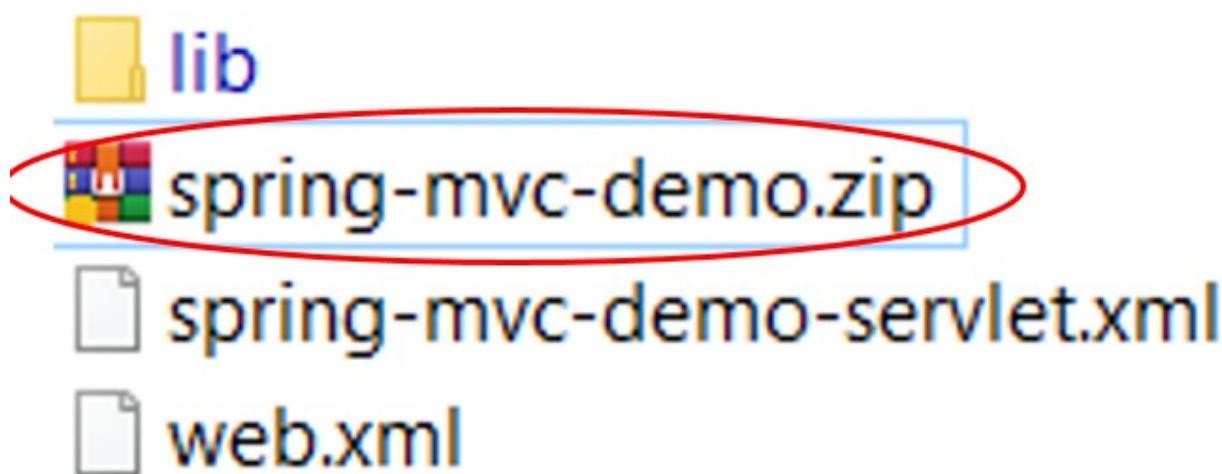
- Se accede a la sección *Web Project Settings* y se modifica el *Context root* a /



- Finalmente se reinicia el servidor Tomcat.



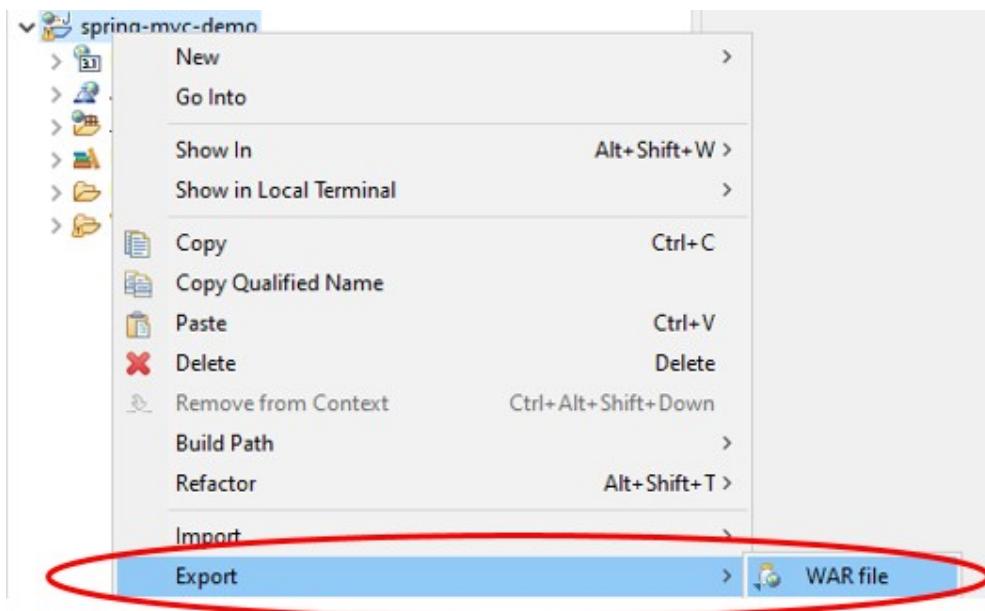
- Una buena práctica para terminar es comprimir todo el proyecto ya funcionando para poder acceder a él en futuras aplicaciones y evitar repetir todo el proceso anterior.



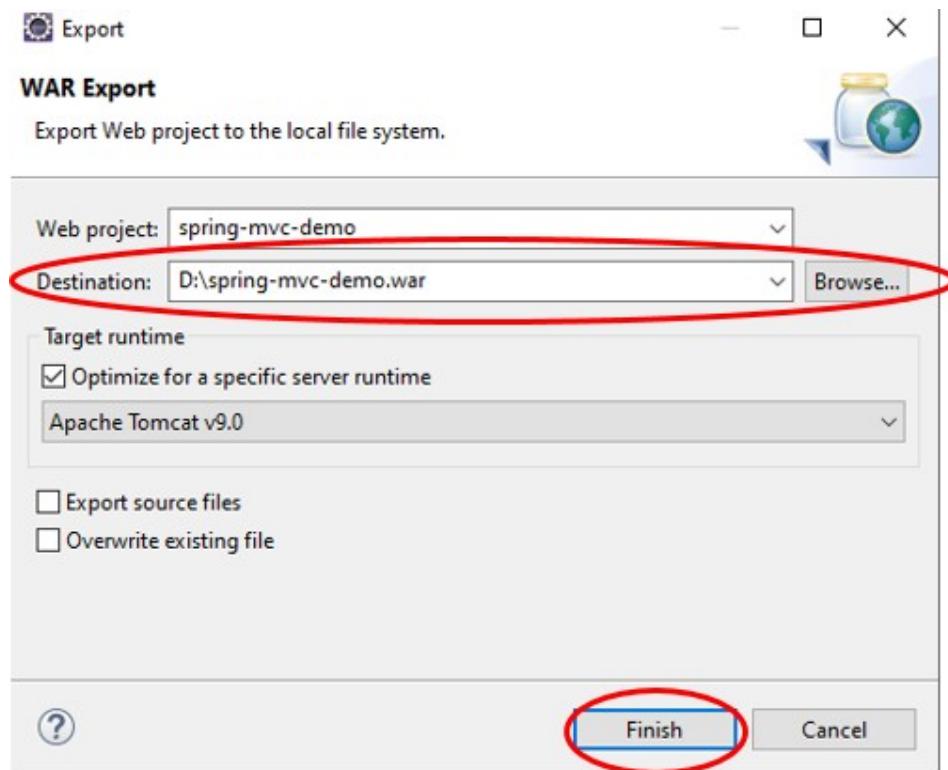
Despliegue del proyecto

- Para desplegar la aplicación en un servidor Tomcat sin utilizar Eclipse es necesario exportar todo el proyecto a un archivo WAR (Web Application Archive).

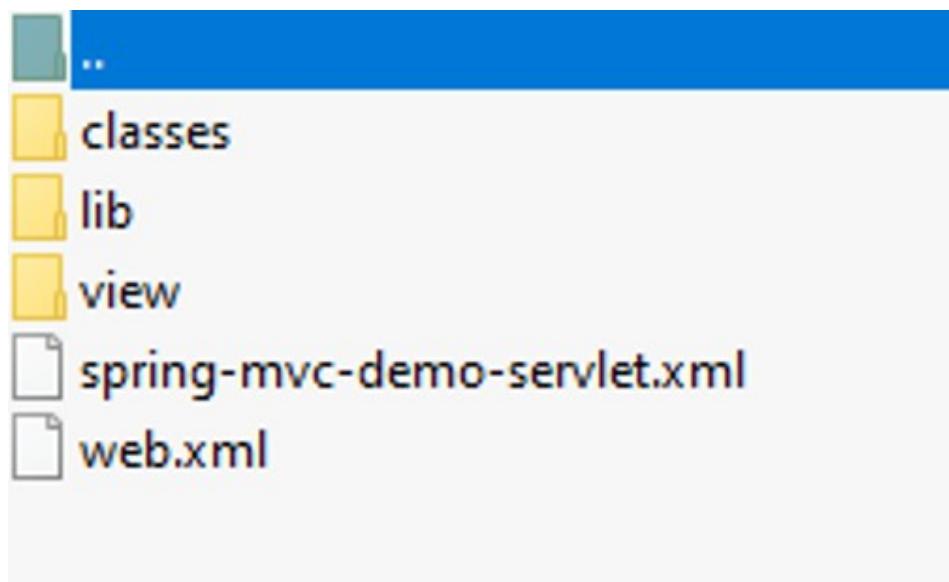
- El archivo WAR es una versión comprimida en formato ZIP de toda la aplicación, concretamente el contenido del directorio WebContent, además de las clases de Java compiladas.
- El formato de archivo WAR forma parte de la especificación Java EE y, por tanto, puede desplegarse en cualquier servidor que soporte este estándar (WildFly, GlassFish, IBM WebSphere y, por supuesto, Tomcat).
- Para generar el archivo WAR de todo el proyecto en Eclipse se pulsa el botón derecho en el proyecto y, a continuación: *Export -> WAR File*



- A continuación se especifica la ruta donde se guardará el archivo WAR.

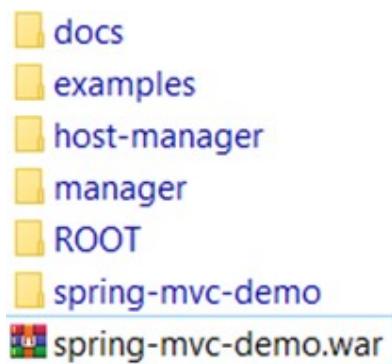


- El archivo WAR generado puede abrirse con cualquier software de compresión de archivos.

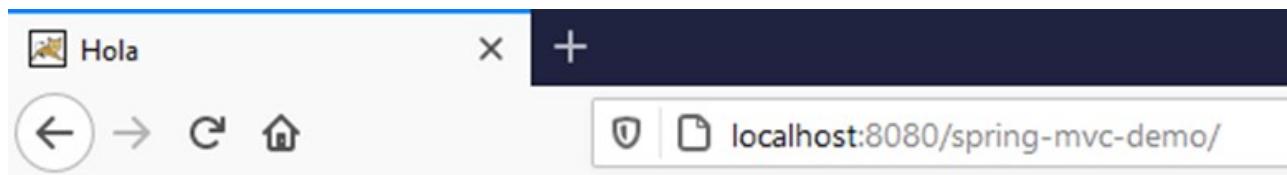


- Seguidamente se accede al directorio donde se encuentra instalado Tomcat 9 (por defecto es C:\Program Files\Apache Software Foundation\Tomcat 9.0) y se accede al directorio webapps.
- En este directorio se copia el archivo WAR.

- Tomcat creará automáticamente un directorio nuevo con el mismo nombre y la aplicación ya estará desplegada.



- La URL de la aplicación es: `http://localhost:8080/<nombre_proyecto>`



Hello world!

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Spring MVC - Fundamentos



Spring MVC

- ◆ Configurar *DispatcherServlet* sin archivo de configuración XML
- ◆ Leer datos de un formulario
- ◆ Añadir datos al modelo
- ◆ Anotaciones @GetMapping y @PostMapping
- ◆ Integrar imágenes, CSS y JavaScript (archivos estáticos)
- ◆ Parámetros de petición
- ◆ Rutas anidadas
- ◆ Cuerpo de la petición
- ◆ Cabeceras
- ◆ Respuesta HTTP
- ◆ Respuestas personalizadas a errores

Configurar *DispatcherServlet* sin archivo de configuración XML

- ◆ Es posible configurar el *DispatcherServlet* de Spring sin necesidad de archivos de configuración XML, utilizando solamente código Java.
- ◆ Los pasos a seguir son los siguientes:
 1. Eliminar los archivos web.xml y spring-mvc-demo-servlet.xml
 2. Crear un nuevo paquete Java para almacenar el archivo de configuración (por ejemplo com.ejemplo.config).
 3. Añadir los dos siguientes archivos de Java al paquete creado anteriormente.

```
// DemoAppConfig.java

@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.luv2code.springdemo")
public class DemoAppConfig {

    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");

        return viewResolver;
    }
}
```

```
// MySpringMvcDispatcherServletInitializer.java

public class MySpringMvcDispatcherServletInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

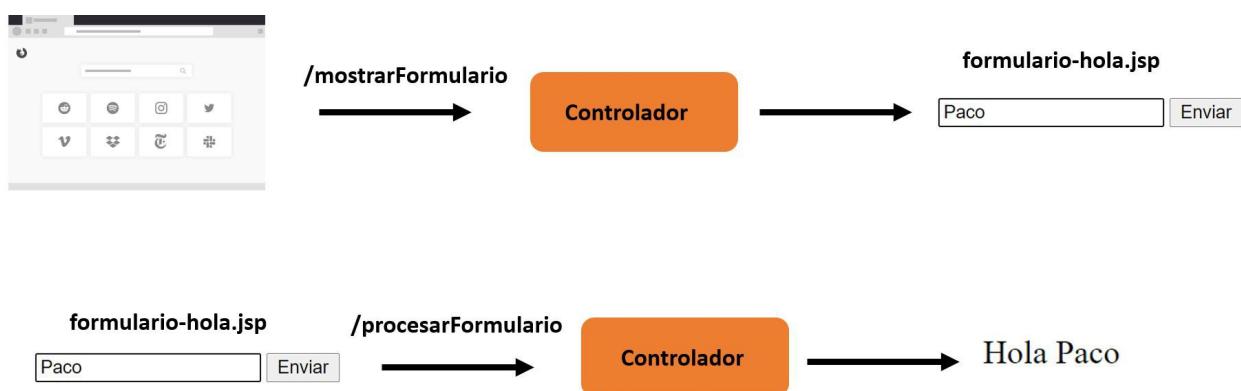
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { DemoAppConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

- La aplicación debería funcionar de forma idéntica.

Leer datos de un formulario

- El ejemplo más sencillo para entender el flujo de datos que sucede desde el navegador web hasta el controlador de Spring, plantillas y viceversa es mediante un simple formulario.



- En este caso, el controlador debe procesar dos rutas y devolver su respectiva plantilla JSP.

```
// Controlador.java

@Controller
public class Controlador {
```

```
// http://localhost:8080/mostrarFormulario
@RequestMapping("/mostrarFormulario")
public String mostrarFormulario() {

    // /WEB-INF/views/formulario-hello.jsp
    return "formulario-hola";
}

// http://localhost:8080/procesarFormulario
@RequestMapping("/procesarFormulario")
public String procesarFormulario() {

    // /WEB-INF/views/hello.jsp
    return "hola";
}
```

- El formulario simplemente está compuesto por una caja de texto y un botón.

```
<!-- formulario-hola.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
    <title>Hola - Formulario</title>
</head>
<body>

    <!-- el formulario enviará los datos a la url /procesarFormulario a través del método
GET de HTTP -->
    <form action="procesarFormulario" method="GET">

        <!-- caja de texto donde se escribirá el nombre -->
        <input type="text" name="nombre" placeholder="Escribe tu nombre" />

        <!-- botón para enviar la información -->
        <input type="submit" />

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

- El valor enviado por el formulario (el nombre escrito en la caja de texto) debe ser redirigido a la otra plantilla JSP.

- Esto lo realiza automáticamente Spring y el valor puede ser accedido directamente en la otra plantilla mediante la sintaxis \${}, haciendo referencia a *param.* y, a continuación, el valor del atributo *name* del campo de formulario.

```
<!-- hola.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

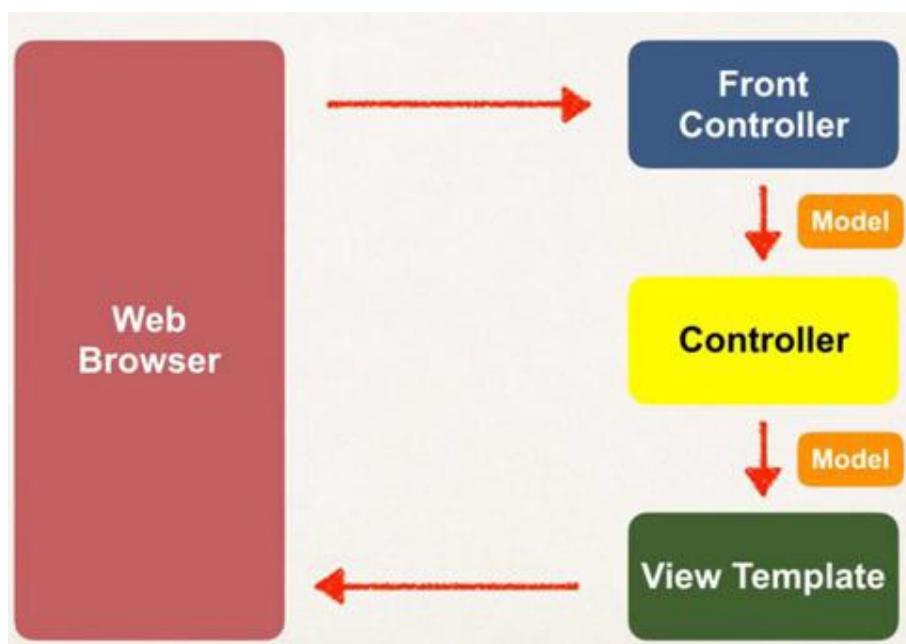
<!DOCTYPE html>
<html>
<head>
<title>Hola</title>
</head>
<body>

<!-- nombre es el valor del atributo name para el campo de texto del formulario
anterior -->
Hola ${param.nombre}

</body>
</html>
```

Añadir datos al modelo

- Los valores de los campos de un formulario están disponibles en la plantilla porque el modelo fluye de forma transparente entre el controlador (donde llegan los datos del formulario enviados por el usuario) y la plantilla.



- El modelo es realmente un contenedor para todos los datos de la aplicación: enteros, booleanos, objetos, información de la base de datos, ...
- Toda la información del modelo puede ser accedida desde la plantilla JSP.
- Desde el controlador es posible manipular el contenido del modelo (modificando, eliminando o añadiendo nueva información).
- Para acceder a los datos de la petición del cliente y al modelo se utiliza el objeto de la clase HttpServletRequest y el objeto de la clase Model respectivamente.
 - No es obligatorio que un método con la anotación @RequestMapping reciba objetos de estas clases.

```
// Controlador.java

@Controller
public class Controlador {

    @RequestMapping("/mostrarFormulario")
    public String mostrarFormulario() {
        return "formulario-hola";
    }

    // request permite acceder a los datos de la petición del cliente y model al modelo
    @RequestMapping("/procesarFormulario")
    public String procesarFormulario(HttpServletRequest request, Model model) {

        // se extrae el valor del atributo name del campo del formulario formulario
        String nombre = request.getParameter("nombre");

        // si el usuario no ha introducido ningún valor (null), entonces se iguala a una cadena
        // vacía
        String mensaje = (nombre == null) ? "undefined" : nombre.toUpperCase();
        System.out.println("El valor enviado por el usuario es " + mensaje);

        // crear un nuevo elemento en el modelo
        model.addAttribute("mensaje", mensaje);

        return "hola";
    }
}
```

- Ahora puede accederse al nuevo elemento desde la plantilla JSP.

```
<!-- hola -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Hola</title>
</head>
<body>
  Hola ${mensaje}
</body>
</html>
```

Anotaciones @GetMapping y @PostMapping

- La anotación @RequestMapping captura todas las peticiones hacia una URL, independientemente del tipo de método HTTP (GET, POST, PUT, etc.).
- Para capturar solamente las peticiones GET y POST pueden utilizar las anotaciones @GetMapping y @PostMapping respectivamente.

```
// Controlador.java

@Controller
public class Controlador {

    // para peticiones GET en la ruta /formulario
    @GetMapping("/formulario")
    public String mostrarFormulario() {
        return "formulario-hola";
    }

    // para peticiones POST en la ruta /formulario
    @PostMapping("/formulario")
    public String procesarFormulario(HttpServletRequest request, Model model) {

        String nombre = request.getParameter("nombre");
        String mensaje = (nombre == null) ? "undefined" : nombre.toUpperCase();
        System.out.println("El valor enviado por el usuario es " + mensaje);
        model.addAttribute("mensaje", mensaje);
        return "hola";
    }
}
```

- Con la anotación @RequestMapping también se puede establecer la ruta y adicionalmente el método HTTP.

```
// Controlador.java

@Controller
public class Controlador {

    /* .. */

    // para peticiones GET en la ruta /formulario
    @RequestMapping(path = "/formulario", method=RequestMethod.GET)
    public String procesarFormulario(HttpServletRequest request, Model model) {

        /* .. */

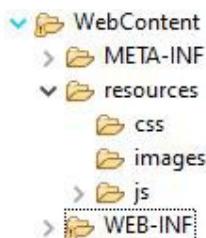
    }

}
```

- También existen las anotaciones @PatchMapping, @PutMapping, @DeleteMapping...

Integrar imágenes, CSS y JavaScript (archivos estáticos)

- Las plantillas JSP requieren generalmente de archivos CSS, JavaScript y demás.
- Todos estos archivos suelen almacenarse en los siguientes directorios:
 - CSS: *WebContent/resources/css*
 - JavaScript: *WebContent/resources/js*
 - Imágenes: *WebContent/resources/images*



- Es necesario añadir esta ruta en el archivo de configuración *spring-mvc-demo-servlet.xml*

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

<context:component-scan base-package="com.ejemplo" />

<!-- referencia a los archivos estáticos -->
<mvc:resources mapping="/resources/**" location="/resources/"></mvc:resources>

<!-- soporte para la conversión, formateo y validación en Spring -->
<mvc:annotation-driven/>
```

<!-- ... -->

- Para probar el funcionamiento puede crearse una nuevo archivo JavaScript en la ruta establecida anteriormente.

```
// WebContent/resources/js/main.js

alert('Prueba');
```

- En la plantilla JSP se puede hacer referencia a los archivos estáticos a partir de la variable `pageContext.request.contextPath`, que hace referencia a la ruta del contexto desde donde realiza la petición el cliente.

```
<!-- index.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
    <script src="${pageContext.request.contextPath}/resources/js/main.js"></script>
    <title>Hola</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Parámetros de petición

- Tal y como se comentó anteriormente, puede accederse al valor de los parámetros de la URL (generada, por ejemplo, por un formulario con `method=GET`) mediante el objeto de la clase `HttpServletRequest`
 - La URL generada posee una forma `http:...?clave1=valor1&clave2=valor2`
- Otra opción más simple es utilizar la anotación `@RequestParam`, que recibe como parámetro el nombre del parámetro que se desea recibir.
- Si se utiliza la anotación `@RequestParam`, entonces el parámetro que se espera recibir es obligatorio. Si el cliente no envía el parámetro, entonces se genera una excepción `MissingServletRequestParameterException`

```
// Controlador.java

// el parámetro nombre es mapeado a la variable nombre de tipo String
public String procesarFormulario(@RequestParam("nombre") String nombre, Model
model) {

String mensaje = (nombre == null) ? "undefined" : nombre.toUpperCase();
System.out.println("El valor enviado por el usuario es " + mensaje);
model.addAttribute("mensaje", mensaje);
return "formulario-hola";
}
```

- Otra forma habitual de pasar valores mediante la URL es mediante la sintaxis `http:.../valor1/valor2/...`
- La anotación `@RequestMapping` puede recibir valores dinámicos a partir de la sintaxis `{}`
- Los valores son manejados después en el código mediante la anotación `@PathVariable`

```
// http://localhost:8080/enviar/{nombre}
// {nombre} representa un fragmento de la URL dinámica
@RequestMapping("/enviar/{nombre}")

// el valor del fragmento nombre de la URL es mapeada a la variable String nombre
// el nombre de la variable en Java debe coincidir con el nombre del fragmento URL
// establecido en @RequestMapping
public String enviar(@PathVariable String nombre, Model model) {
    String mensaje = (nombre == null) ? "undefined" : nombre.toUpperCase();
    System.out.println("El valor enviado por el usuario es " + mensaje);
    model.addAttribute("mensaje", mensaje);
    return "formulario-hola";
}
```

Rutas anidadas

- Una clase puede definir una ruta raíz mediante la anotación `@RequestMapping`, de tal forma que todos sus métodos la tomen como prefijo.

```
@Controller
// ruta raíz --> http://localhost:8080/home
@RequestMapping("/home")
public class Controlador {

    // http://localhost:8080/home/mostrarFormulario
    @RequestMapping("/mostrarFormulario")
    public String mostrarFormulario()
```

```

        return "formulario-hola";
    }

// http://localhost:8080/home/procesarFormulario
@RequestMapping("/procesarFormulario")
public String procesarFormulario() {
    return "formulario-hola";
}
}

```

Cuerpo de la petición

- A veces los datos son enviados mediante el método POST de HTTP.
- En este caso los datos se encontrarían en el cuerpo (body) del paquete HTTP.

```

<!-- formulario-hola.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
    <title>Hello - Formulario</title>
</head>
<body>

    <!-- ahora el método es POST -->
    <form action="procesarFormulario" method="`POST`">
        <input type="text" name="nombre" placeholder="Escribe tu nombre" />
    </form>

</body>
</html>

```

- Para obtener la información desde Spring se utiliza la anotación @RequestParam, como si fuera una petición GET.
- Puede probarse también el envío de solicitudes HTTP POST desde Postman seleccionando un formato *x-www-form-urlencoded* para el *body*
- Para acceder a todo el contenido del cuerpo puede utilizarse la anotación @RequestBody

```

@RequestMapping("/procesarFormulario")
public String procesarFormulario(@RequestParam("nombre") String nombre,
@RequestBody String body, Model model) {

```

```
// la sintaxis del body cuando se envía datos de un formulario es
clave1=valor1&clave2=valor2...
System.out.println("El cuerpo es " + body);

String mensaje = (nombre == null) ? "undefined" : nombre.toUpperCase();
System.out.println("El valor enviado por el usuario es " + mensaje);
model.addAttribute("mensaje", mensaje);
return "hola";
}
```

- Con la anotación @RequestMapping también se puede establecer el tipo de método HTTP específico a capturar (en lugar de capturar todos).

```
// captura solamente el método POST de HTTP
@RequestMapping(value="/home/procesarFormulario", method=RequestMethod.POST)
public String procesarFormulario() {
    return "hola";
}
```

Cabeceras

- Acceder a las cabeceras HTTP de una petición también suele ser habitual.
- La forma más simple es con la anotación @RequestHeader, recibiendo como parámetro el nombre de la cabecera cuyo valor se desea obtener.

```
@RequestMapping("/mostrarFormulario")

// se accede al valor de la cabecera accept-language y se almacena en la variable
acceptLanguage
public String mostrarFormulario(@RequestHeader("accept-language") String
acceptLanguage) {
    System.out.println("El valor de la cabecera accept-language es " + acceptLanguage);
    return "formulario-hola";
}
```

- También se puede acceder a todos los elementos de la cabecera asignando la anotación @RequestHeader a una variable de tipo Map

```
@RequestMapping("/mostrarFormulario")
public String mostrarFormulario(@RequestHeader Map<String, String> cabeceras) {

// se recorren todos los elementos de la cabecera
cabeceras.forEach((clave, valor) -> {
```

```

        System.out.println(clave + ":" + valor);
    });

    return "formulario-hola";
}

```

Respuesta HTTP

- Por defecto, Spring devolver el **código HTTP** 200 ante una respuesta exitosa.
- Sin embargo, puede devolverse otro código HTTP mediante la anotación `@ResponseStatus`, pasándole por parámetro el identificador del código a través de la clase `HttpStatus`

```

@Controller
@RequestMapping("/home")
public class Controlador {

    @RequestMapping("/mostrarFormulario")
    // código de estado HTTP 201 (CREATED)
    @ResponseStatus(HttpStatus.CREATED)
    public String mostrarFormulario() {
        return "formulario-hola";
    }
}

```

- Dentro del método manejador puede generarse una excepción asociada a un código HTTP mediante la clase `ResponseStatusException`, que recibe como parámetro el identificador del código a través de la clase `HttpStatus`

```

@Controller
@RequestMapping("/home")

public class Controlador {
    @RequestMapping("/mostrarFormulario")

    // en principio, método debería devolver el código de estado HTTP 201 (CREATED)
    @ResponseStatus(HttpStatus.CREATED)
    public String mostrarFormulario() {

        // pero se devuelve el código de estado NOT_FOUND (404)
        throw new ResponseStatusException(HttpStatus.NOT_FOUND);
    }
}

```

- Puede crearse también un controlador global para manejar las excepciones mediante la anotación @ControllerAdvice

```
// ManejadorExcepciones.java
```

```
@ControllerAdvice
public class ManejadorExcepciones {

    // maneja todo tipo de excepciones (Exception.class)
    @ExceptionHandler(Exception.class)
    public String manejadorExpcion(HttpServletRequest req, ResponseStatusException ex){

        System.out.println("Manejador de excepciones");

        // redirige a la página error.jsp
        return "error";
    }

    // maneja solamente la excepción ResponseStatusException
    @ExceptionHandler(ResponseStatusException.class)
    public String manejadorResponseStatusException(HttpServletRequest req,
    ResponseStatusException ex){

        System.out.println("Manejador de la excepción ResponseStatusException");

        // redirige a la página error.jsp
        return "error";
    }
}
```

- En caso de que existan varios métodos para manejar la misma excepción, Spring invocará a la más específica.
- No pueden existir dos métodos manejadores que capturan la misma excepción.
- Spring devolverá un error 500 al cliente si se genera una excepción no controlada.
- Spring devolverá un error 404 al cliente si se accede a una URL o página JSP inexistente.

Respuestas personalizadas a errores

- Para construir respuestas personalizadas a errores específico es necesario modificar el archivo web.xml para indicar a qué URL serán redirigidos los errores.

```
<error-page>
```

```
<!-- los errores se redirigirán a esta URL -->
```

```
<location>/errors</location>
```

```
</error-page>
```

- Seguidamente será necesario reiniciar el servidor Tomcat y crear un controlador nuevo para controlar los errores.

```
// ManejadorError.java

@Controller
public class ManejadorError {

    @RequestMapping(value = "errors", method = RequestMethod.GET)
    public String renderErrorHandler(HttpServletRequest httpRequest, Model model) {

        String msg = "";

        // obtiene el código HTTP
        int codigo = (Integer) httpRequest.getAttribute("javax.servlet.error.status_code");

        switch (getErrorCode(httpRequest)) {

            case 400: {
                msg = "Código de estado: 400 - Petición errónea";
                break;
            }

            case 401: {
                msg = "Código de estado: 401 - No autorizado";
                break;
            }

            case 404: {
                msg = "Código de estado: 404 - No encontrado";
                break;
            }

            case 500: {
                msg = "Código de estado: 500 - Error interno";
                break;
            }
        }

        model.addAttribute("msg", msg);
        return "error";
    }
}
```

```
<!-- error.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>

${msg}
```

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Introducción a Hibernate



Introducción a Hibernate

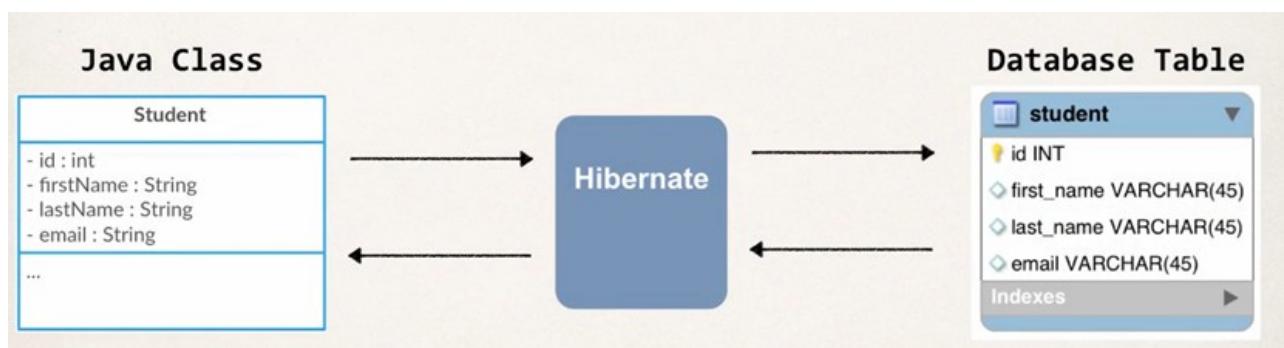
- ◆ Introducción
- ◆ Integración de Hibernate en un proyecto

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) → (highlighted with a red circle)

Development (6.0) →

About

- [Releases](#)
- [Documentation](#)
- [Books](#)
- [Migration guides](#)
- [Roadmap](#)
- [Tooling](#)
- [Envers](#)
- [Contribute](#)
- [Paid support](#)
- [FAQ](#)

- [Source code](#)
- [Issue tracker](#)
- [Security issue](#)
- [Forum](#)
- [Wiki](#)

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

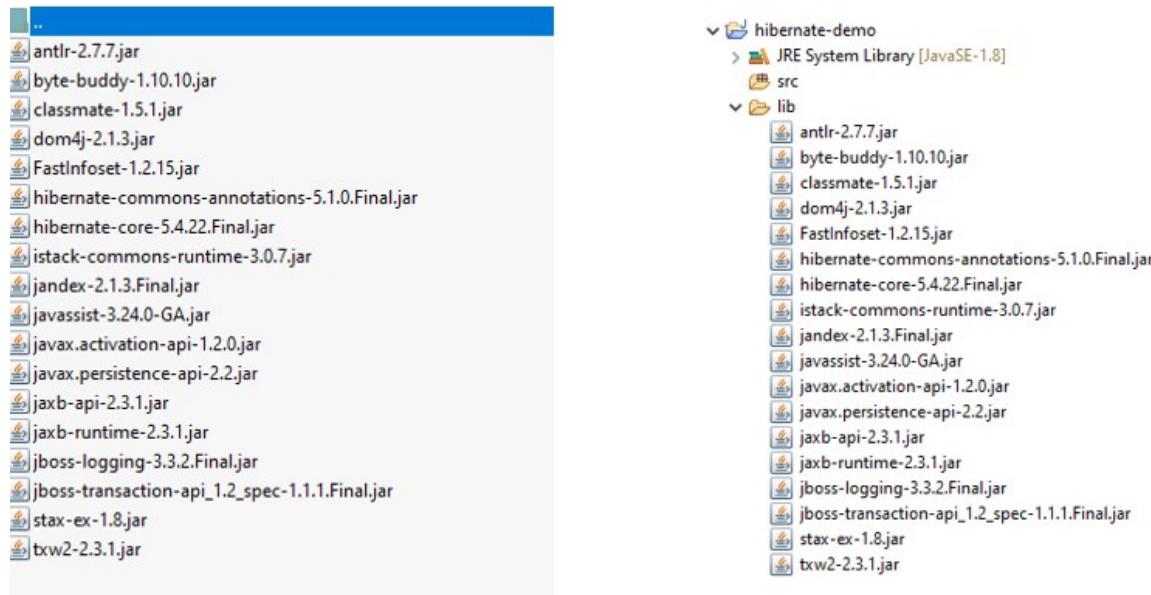
- Compatibility
- Documentation
- How to get it
- What's new
- Migrate
- Releases in this series

Zip archive

Direct download is available from SourceForge:

[Download Zip archive](#)

- 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.

⌚ 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.

Connector/J 8.0.21

Select Operating System:

(Platform Independent)

Looking for previous GA versions?

| 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 | Download |
| Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.21.zip) | 8.0.21 | 4.5M | Download |

Tip: We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

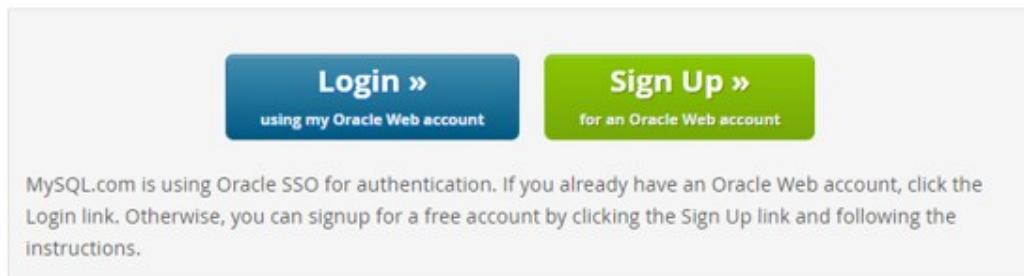
- No será necesario registrarse para comenzar la descarga.

⌚ MySQL Community Downloads

Login Now or Sign Up for a free account.

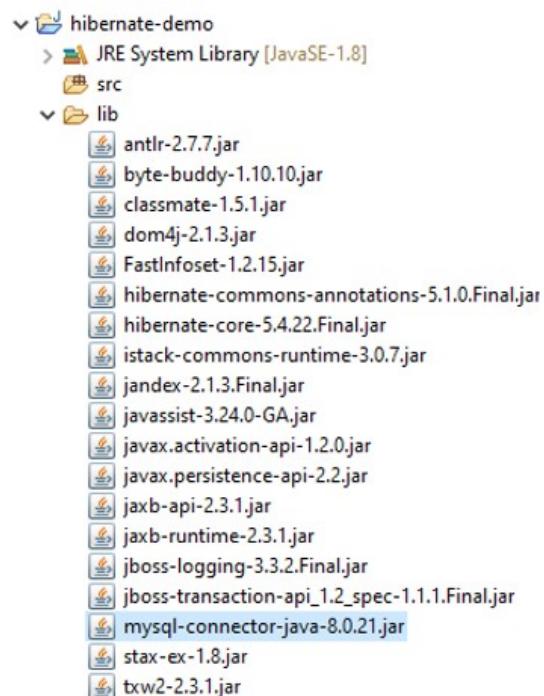
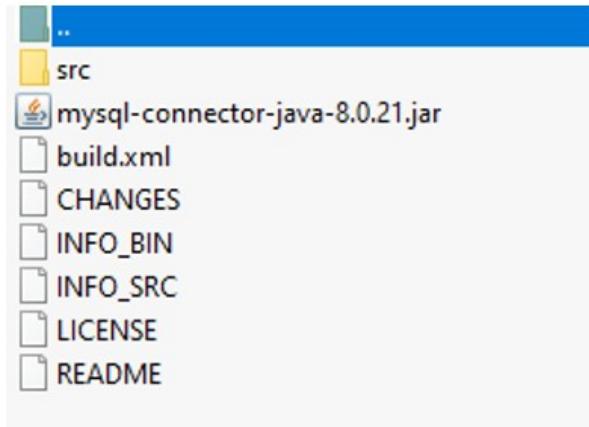
An Oracle Web Account provides you with the following advantages:

- 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

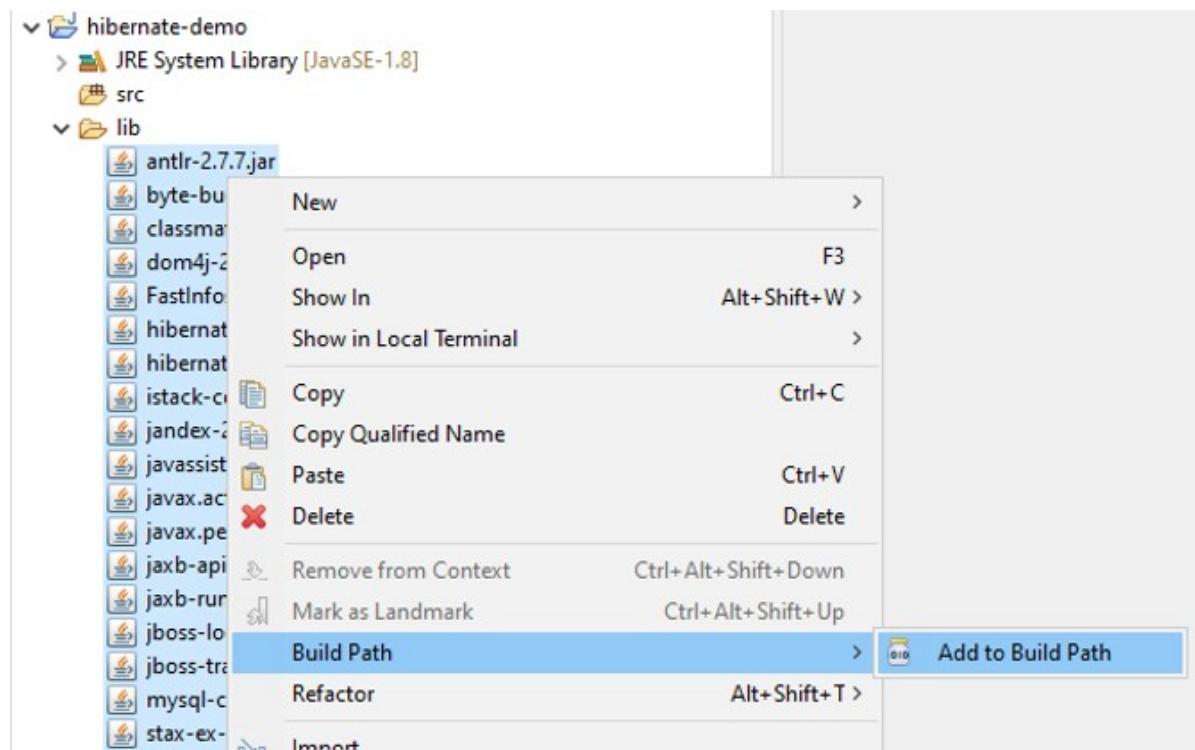


No thanks, just start my download.

- 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 ~

Módulo 5

Spring e Hibernate

Hibernate básico



Hibernate básico

- ◆ Configuración
- ◆ Anotaciones
- ◆ Creación y almacenamiento de objetos
- ◆ Clave primaria
- ◆ Lectura de objetos
- ◆ Actualización de objetos
- ◆ Eliminación de objetos
- ◆ Trabajar con fechas y horas

Configuración

- ◆ El archivo de configuración de Hibernate permitirá establecer los parámetros de conexión con la base de datos.
- ◆ Este archivo debe ubicarse en el directorio *src* del proyecto.

```
<!-- hibernate.cfg.xml -->

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <!-- parámetros de conexión -->
    <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/hibernate-demo?
useSSL=false&serverTimezone=UTC</property>
    <property name="connection.username">hibernate-demo</property>
    <property name="connection.password">hibernate-demo</property>

    <!-- número de conexiones máxima que puede establecer Hibernate al mismo tiempo
        con la base de datos -->
    <property name="connection.pool_size">10</property>

    <!--
        aunque SQL es un estándar, existen diferencias entre sistemas de bases de datos
        (dialectos).
        Pueden revisar los diferentes dialectos de hibernate en
        https://docs.jboss.org/hibernate/orm/5.4/javadocs/org/hibernate/dialect/package-
summary.html
    -->
```

```

<!-- dialecto SQL utilizado por Hibernate (MySQL8Dialect recomendable para últimas
versiones de MySQL y MariaDB)-->
<!-- también puede utilizarse el dialecto MySQL5InnoDBDialect -->
<prop key="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect </prop>

<!--muestra todas las sentencias SQL por consola -->
<property name="show_sql">true</property>

<!--
el contexto será el hilo donde se abra la sesión (de esta forma se recuperará
la misma sesión si se trata de abrir de nuevo la sesión desde el mismo hilo)
-->
<property name="current_session_context_class">thread</property>

<!--
manipula la base de datos al comienzo de la conexión. Los posibles valores son:
 * update: actualiza el esquema de tablas (su uso no es recomendado)
 * create: crea el esquema de tablas, destruyendo los datos previos
 * create-drop: elimina el esquema cuando el SessionFactory es cerrado
 * none: no realiza ninguna acción
-->
<property name="hbm2ddl.auto">update</property>

</session-factory>

</hibernate-configuration>

```

Anotaciones

- Hibernate utiliza el concepto de clase entidad, que es una clase de Java que es mapeada a una tabla en una base de datos.
- Una clase entidad es simplemente una clase de Java con atributos, métodos getters/setters y anotaciones de Hibernate (o archivos de configuración XML).
- Por un lado se mapea la clase a una tabla de la base de datos y, por otro, los atributos de la clase a las columnas de la tabla.

```

// Estudiante.java

// Hibernate recomienda utilizar las librerías de JPA (un estándar) para importar
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

// indica que la clase se convierte en una clase entidad
@Entity

```

```
// indica que realice un mapeo entre esta clase y la tabla estudiante
@Table(name="estudiante")
public class Estudiante {

    // el atributo id será una clave primaria única
    @Id

    // autoincremental
    @GeneratedValue(strategy = GenerationType.IDENTITY)

    /*
    en este caso (y en los siguientes), el atributo name es redundante porque tiene el
    mismo
    nombre que tendrá la columna de la base de datos
    */

    // id (atributo) --> id (columna en la tabla estudiante)
    @Column(name="id")
    private int id;

    // nombre (atributo) --> nombre (columna en la tabla estudiante)
    @Column(name="nombre")
    private String nombre;

    // apellidos (atributo) --> apellidos (columna en la tabla estudiante)
    @Column(name="apellidos")
    private String apellidos;

    // edad (atributo) --> edad (columna en la tabla estudiante)
    @Column(name="edad")
    private int edad ;

    // es obligatorio que exista un constructor por defecto
    public Estudiante() {}

    // es recomendable definir un constructor parametrizado con todos los atributos
    // (excepto el id)
    // este método puede generarse automáticamente con Eclipse
    public Estudiante(String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }

    // para depuración es recomendable definir un método *toString* que incluye atributos
    // y valores
    // este método puede generarse automáticamente con Eclipse
    @Override
    public String toString() {
        return "Estudiante [nombre=" + nombre + ", apellidos=" + apellidos + ", edad=" +
        edad + "]";
    }
}
```

```
}
```

```
public Integer getId() {
    return id;
}
```

```
public String getNombre() {
    return nombre;
}
```

```
public void setNombre(String nombre) {
    this.nombre = nombre;
}
```

```
public String getApellidos() {
    return apellidos;
}
```

```
public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}
```

```
public int getEdad() {
    return edad;
}
```

```
public void setEdad(int edad) {
    this.edad = edad;
}
}
```

Creación y almacenamiento de objetos

- Hibernate trabaja a nivel de programación con dos clases fundamentales:
 - *SessionFactory*:
 - Lee el archivo de configuración de Hibernate.
 - Crea objetos de sesión.
 - Su inicialización es costosa.
 - Solamente debe ser creado una vez para todo el ciclo de vida de la aplicación.
 - *Session*:
 - Implementa la conexión JDBC con la base de datos.
 - Es la principal clase para almacenar y obtener objetos.
 - Pueden crearse varios y tiene un tiempo de vida corto
 - Obtenido a partir del *SessionFactory*.

```
// Main.java
```

```
public class Main {
```

```

public static void main(String[] args) {

    /*
     * creación del objeto SessionFactory. Hibernate buscará por defecto el archivo
     * hibernate.cfg.xml, aunque no se le indique, por lo que no es necesario establecerlo
     * como parámetro en el método configure
    */
    SessionFactory factory = new Configuration()
        .configure("hibernate.cfg.xml")
        .addAnnotatedClass(Estudiante.class)
        .buildSessionFactory();

    // creación de un objeto de tipo Session a partir del SessionFactory
    Session session = factory.getCurrentSession();

    try {
        // obtención/manipulación/almacenamiento de objetos
    }
    finally {
        // cierre del objeto SessionFactory
        factory.close();
    }
}
}

```

- Despues de la creación del objeto de tipo Session puede comenzarse a trabajar con la base de datos.
- El almacenamiento de un objeto en la base de datos se realiza mediante el método *save* del objeto *Session*. Previamente es necesario iniciar una transacción con *beginTransaction* y posteriormente finalizarla con *getTransaction().commit()*

```

// Main.java

// ...
try {

    // crea un nuevo objeto
    Estudiante estudiante = new Estudiante("Marcos", "González", 25);

    // comienza la transacción
    session.beginTransaction();

    // guarda el objeto
}

```

```

    session.save(estudiante);

    // finalizar y guardar la transacción (los cambios se efectúan en la base de datos)
    session.getTransaction().commit();
}

// ...

```

- Hibernate asigna automáticamente el valor al id del objeto (coincidente con el almacenado en la base de datos) cuando al guardar.

```

Estudiante estudiante = new Estudiante("Marcos", "González", 25);

// null (en caso de que id haya sido declarado como Integer) o 0 (en caso de int)
System.out.println(estudiante.getId());

// almacenamiento del objeto
session.beginTransaction();
session.save(estudiante);
session.getTransaction().commit();

// devuelve un número entero (igual o mayor que 1)
System.out.println(estudiante.getId());

```

Clave primaria

- La clave primaria sobre un campo en una tabla posee las siguientes características:
 - Identifica de forma única a una fila dentro de una tabla.
 - No puede contener nunca un valor NULL.
 - Solamente puede existir una clave primaria (aunque puede estar compuesta por dos campos).
 - No es obligatorio que una tabla tenga clave primaria, aunque sí recomendable.
- Habitualmente los campos que son clave primaria también son autogenerables (no es necesario asignarles valor, sino que la base de datos es la encargada de hacerlo).
- En Hibernate se identifica a un campo como clave primaria con la anotación @Id
- En Hibernate se identifica a un campo como autogenerable mediante la anotación @GeneratedValue. Esta anotación que acepta como el parámetro strategy, que puede tomar los siguientes valores:
 - GenerationType.AUTO: establece la autogeneración recomendada por el sistema de base de datos configurado.
 - GenerationType.IDENTITY: realiza una asignación incremental (recomendable para MySQL)

- Generation.SEQUENCE: realiza una asignación basada en secuencia (generalmente utilizado en bases de datos Oracle).
- Generation.TABLE: utiliza una base de datos secundaria para garantizar unicidad.
- En Hibernate también se puede implementar una estrategia de generación personalizada para las claves primarias mediante la implementación de la clase *org.hibernate.id.IdentifierGenerator* y la sobreescritura del método *public Serializable generate(...)*
- Mediante una consulta SQL simple puede indicarse en qué número debe comenzar el valor autogenerado por la base de datos (por defecto es 1) para una tabla en particular.
 - No es posible cambiar el valor de autoincremento a un valor menor al establecido (aunque la ejecución de la instrucción *ALTER TABLE* no generará ningún error en este caso).

ALTER TABLE regla AUTO_INCREMENT=3000

- Para resetear el valor de autoincremento (y borrar todos los datos de la tabla) se utiliza *TRUNCATE*

TRUNCATE estudiante;

Lectura de objetos

- La forma más simple de obtener un objeto de la base de datos con Hibernate es mediante su id.
- La lectura de objetos se realiza invocando al método *get* del objeto *Session* y será necesario también abrir y cerrar una transacción.

```
// comienza la transacción
session.beginTransaction();

// obtiene el estudiante con el id=1
Estudiante estudiante = session.get(Estudiante.class, 1);
System.out.println(estudiante.getNombre());

// finaliza la transacción
session.getTransaction().commit();
```

- Si no existe un objeto con el id solicitado, entonces el objeto apuntará *null*

- Hibernate posee un lenguaje específico para realizar consultas llamado *Hibernate Query Language* (HQL), muy similar q SQL.
- En este lenguaje de consultas se hace referencia a los campos de la entidad y no a las columnas de la tabla asociada.

```
session.beginTransaction();

// devuelve una lista de todos los objetos de la clase Estudiante (asociada a la tabla estudiante)
List<Estudiante> estudiantes = session.createQuery("from Estudiante",
Estudiante.class).getResultSet();
for (Estudiante estudiante : estudiantes) {
    System.out.println(estudiante.getId() + ":" + estudiante.getApellidos());
}

session.getTransaction().commit();
```

- En la consulta pueden establecerse condiciones mediante *where*

```
session.beginTransaction();

// devuelve una lista de todos los objetos de la clase Estudiante (utilizando el alias s)
que cumplan que el atributo apellidos='González'
String query = "from Estudiante s where s.apellidos='González'";
List<Estudiante> estudiantes = session.createQuery(query,
Estudiante.class).getResultSet();

session.getTransaction().commit();
```

- Y también operadores lógicos como *and* y *or*

```
session.beginTransaction();

// se debe cumplir al menos una de las dos condiciones
String query = "from Estudiante s where s.apellidos='González' or s.nombre='Pepito'";
List<Estudiante> estudiantes = session.createQuery(query,
Estudiante.class).getResultSet();
for (Estudiante estudiante : estudiantes) {
    System.out.println(estudiante.getId() + ":" + estudiante.getApellidos());
}

session.getTransaction().commit();
```

- El operador *LIKE* funciona igual que en el lenguaje SQL.

```

session.beginTransaction();

// devuelve todos los objetos cuyo atributo nombre tengan el comienzo con pep
String query = "from Estudiante s where s.nombre LIKE 'pep%'";
List<Estudiante> estudiantes = session.createQuery(query,
Estudiante.class).getResultSet();
for (Estudiante estudiante : estudiantes) {
    System.out.println(estudiante.getId() + ":" + estudiante.getApellidos());
}

session.getTransaction().commit();

```

Actualización de objetos

- La actualización de objetos en la base de datos no requiere de invocación de ningún método *update* o similar, simplemente se modifica el atributo deseado y se finaliza la transacción

```

session.beginTransaction();

// obtención del estudiante con id=1
Estudiante estudiante = session.get(Estudiante.class, 1);

if (estudiante != null) {

    // modificación de un atributo
    estudiante.setNombre("Roberto");
}

// la actualización se produce en este punto del código
session.getTransaction().commit();

```

- También se pueden realizar actualizaciones mediante el lenguaje HQL.

```

session.beginTransaction();

// cambia el valor del atributo nombre a Pedro de todos los objetos cuyo valor del
atributo nombre sea Roberto
String query = "update Estudiante set nombre='Pedro' where nombre='Roberto'";
session.createQuery(query).executeUpdate();

session.getTransaction().commit();

```

Eliminación de objetos

- La eliminación de objetos en la base de datos se realiza con la invocación del método `delete` en el objeto de tipo *Session*

```
session.beginTransaction();

// obtención del estudiante con id=1
Estudiante estudiante = session.get(Estudiante.class, 1);

if (estudiante != null) {

    // eliminación del objeto
    session.delete(estudiante);
}

// la actualización se produce en este punto del código
session.getTransaction().commit();
```

- También se pueden realizar eliminaciones mediante el lenguaje HQL.

```
session.beginTransaction();

// elimina todos los objetos con el valor Roberto en el atributo nombre
String query = "delete Estudiante where nombre=Roberto";
session.createQuery(query).executeUpdate();

session.getTransaction().commit();
```

Trabajar con fechas y horas

- Hibernate ofrece la anotación `@Temporal` para definir campos de tipo fecha y hora.
- La anotación `@Temporal` acepta como parámetro el tipo de dato temporal:
 - `TemporalType.DATE`: para fechas.
 - `TemporalType.TIME`: para horas.
 - `TemporalType.TIMESTAMP`: para fechas y horas
- Sin embargo, esta anotación obliga a que el tipo definido en Java sea *Date* o *Calendar* (no es posible utilizar *LocalDate*, *LocalTime* o *LocalDateTime*), por lo que no es recomendable utilizarlo en las últimas versiones de Java.
- En las últimas versiones de Java puede optarse por dos alternativas:
 - Utilizar la anotación `@Temporal` con los respectivos parámetros `TemporalType.DATE` (para fechas), `TemporalType.TIME` (para horas) y `TemporalType.TIMESTAMP` (para

fechas y horas) sobre el tipo de dato Calendar (tanto para fechas, horas, y fechas y horas).

- No utilizar la anotación @Temporal sobre los tipos de LocalDate (para fechas), LocalTime (para horas) y Timestamp (para horas y fechas). LocalDateTime provoca problemas actualmente y no debería utilizarse.
- A continuación se expone la segunda alternativa.

```
// Estudiante.java

// ...

class Estudiante {

    // fecha
    @Column(name="fecha")
    private LocalDate fecha;

    // hora
    @Column(name="hora")
    private LocalTime hora;

    // hora y fecha
    @Column(name="hora_y_fecha")
    private Timestamp horaYFecha;

    // constructor parametrizado...

    // métodos getters y setters
}
```

- Hibernate no tiene en cuenta algunas consideraciones que ocurren en determinadas zonas horarias (por ejemplo, el horario UTC+2 en verano para España).
- Para configurar adecuadamente la zona horaria de las fechas almacenadas puede establecerse un valor para la propiedad `hibernate.jdbc.time_zone` en la configuración de Hibernate.
 - La zona horaria Europe/Madrid sí incluye la adaptación de la hora UTC+2 (en verano) y UTC+1 (para el resto del año) en España.
 - Puede revisarse un listado de todas las zonas horarias en la [Wikipedia](#).

```
<!-- hibernate.cfg.xml -->

<!-- -->

<session-factory>
```

```
<property name="hibernate.jdbc.time_zone">Europe/Madrid</property>

</session-factory>

<!-- -->
```

- Para comparar entre fechas es necesario convertir los tipos *LocalDate*, *LocalTime* o *Timestamp*, a *Calendar*.
 - No es necesario conversión alguna si se opta por *Calendar* sobre los tipos de datos de la entidad.
- Por tanto, es recomendable disponer de métodos que realicen este tipo de conversiones.

```
// LocalDate --> Calendar
public static Calendar toCalendar(LocalDate localDate) {
    Calendar calendar = Calendar.getInstance();
    calendar.clear();
    calendar.set(localDate.getYear(), localDate.getMonthValue()-1,
    localDate.getDayOfMonth());
    return calendar;
}

// LocalTime --> Calendar
public static Calendar toCalendar(LocalTime localTime) {
    Calendar calendar = Calendar.getInstance();
    calendar.clear();
    calendar.set(0, 0, 0, localTime.getHour(), localTime.getMinute(),
    localTime.getSecond());
    return calendar;
}

// LocalDateTime --> Timestamp
public static Timestamp toTimestamp(LocalDateTime localDateTime) {
    return Timestamp.valueOf(localDateTime);
}
```

```
// para la hora

// construye la consulta (TypedQuery solamente es necesario para evitar un warning)
TypedQuery<Estudiante> query = session.createQuery("from Estudiante where hora <
:ahora", Estudiante.class);

// obtiene localTime (la hora) y la muestra por pantalla
LocalTime localTime = LocalTime.now();
```

```
Calendar calendar = Main.toCalendar(localTime);
System.out.println(localTime);

// sustituye el valor en la consulta (especificando TemporalType.TIME porque es una
// hora)
query.setParameter("ahora", calendar, TemporalType.TIME);

// ejecuta la consulta
List<Estudiante> estudiantes = query.getResultList();

// muestra el número de registros coincidentes
System.out.println(estudiantes.size());
```

```
// para la fecha

// construye la consulta (TypedQuery solamente es necesario para evitar un warning)
TypedQuery<Estudiante> query = session.createQuery("from Estudiante where fecha
<= :ahora", Estudiante.class);

// obtiene localTime (la fecha) y la muestra por pantalla
LocalDate localDate = LocalDate.now();
Calendar calendar = Main.toCalendar(localDate);
System.out.println(localDate);

// sustituye el valor en la consulta (especificando TemporalType.DATE porque es una
// fecha)
query.setParameter("ahora", calendar, TemporalType.DATE);

// ejecuta la consulta
List<Estudiante> estudiantes = query.getResultList();

// muestra el número de registros coincidentes
System.out.println(estudiantes.size());
```

```
// para la hora y la fecha

// construye la consulta
TypedQuery<Estudiante> query = session.createQuery("from Estudiante where
horaYFechaActual < :ahora", Estudiante.class);

// obtiene localTime (la hora) y la muestra
LocalDateTime localDateTime = LocalDateTime.now();
Timestamp timestamp = Main.toTimestamp(localDateTime);
System.out.println(timestamp);

// sustituye el valor en la consulta (especificando TemporalType.TIMESTAMP porque es
// una fecha)
```

```
query.setParameter("ahora", timestamp, TemporalType.TIMESTAMP);

// ejecuta la consulta
List<Estudiante> estudiantes = query.getResultList();
System.out.println(estudiantes.size());
```

Almacenamiento de tipos de datos

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Spring MVC con Hibernate



Spring MVC con Hibernate

- ◆ Preparación del entorno
- ◆ Testear entorno
- ◆ Objeto de acceso a datos
- ◆ Agregar página de inicio
- ◆ Servicios
- ◆ CRUD completo

Preparación del entorno

- ◆ Un proyecto web dinámico con Spring e Hibernate (sin utilizar Maven) debe incluir:
 - Añadir los archivos de configuración web.xml y spring-mvc-servlet.xml (o anotaciones en su defecto).
 - Añadir librerías JSTL
 - Añadir librerías de Spring
 - Añadir librerías de Hibernate (lib/required y lib/optional/c3p0).
 - Añadir librería de MySQL
- ◆ Todas las librerías JAR deben ubicarse dentro del directorio WebContent/WEB-INF/lib y los archivos xml en WebContent/WEB-INF
- ◆ Respecto a la configuración de Spring (spring-mvc-servlet.xml) es necesario:
 - Definir el datasource de la base de datos: datos de conexión a la base de datos.
 - Configurar el session factory de Hibernate: el mecanismo que tiene Hibernate para conectarse a la base de datos (aquí se define el tipo de base de datos a conectar, haciendo referencia al datasource).
 - Configurar el transaction manager de Hibernate, de tal forma que simplifica en Spring la creación y destrucción de sesiones de Hibernate en Spring.
 - Habilita la configuración de anotaciones de transacción, de tal forma que simplifica en Spring el inicio o parado de la transacciones de Hibernate en Spring.

```
<!-- WebContent/WEB-INF/web.xml -->

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  id="WebApp_ID" version="3.1">

  <!-- identificador interno de toda la aplicación web -->
  <display-name>Tienda</display-name>
```

```
<!-- especifica qué fragmentos web JAR (dentro del archivo WEB-INF/web-  
fragment.xml)  
deben escanearse en busca de fragmentos y anotaciones. Un elemento <absolute-  
ordering  
/> vacío configura que ninguno sea escaneado -->  
<absolute-ordering></absolute-ordering>  
  
<!-- configuración del DispatcherServlet (Front Controller) -->  
<!-- pueden declararse varios servlet -->  
<servlet>  
  
<!-- nombre del servlet (puede ser cualquier nombre) -->  
<servlet-name>dispatcher</servlet-name>  
  
<!-- ubicación del DispatcherServlet en las librerías de Spring -->  
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
  
<!-- parámetros de configuración del DispatcherServlet -->  
<init-param>  
  
<!-- indica que la configuración del contexto se encuentra en el archivo  
/WEB-INF/spring-mvc-demo-servlet.xml -->  
<param-name>contextConfigLocation</param-name>  
<param-value>/WEB-INF/spring-mvc-servlet.xml</param-value>  
</init-param>  
  
<!-- load-on-startup >= 0 indica que el servlet se carga cuando se inicia  
el servidor (el tratamiento de la primera petición será más rápido) -->  
<!-- load-on-startup < 0 indica que el servlet se cargará cuando se recibe  
la primera petición (el tratamiento de la primera petición será más lento) -->  
<!-- cuando existen más servlets, el orden de arrancado dependerá del valor  
de load-on-startup. Primero el que tenga asignado el valor 0, después el  
1 y así sucesivamente -->  
<load-on-startup>1</load-on-startup>  
</servlet>  
  
<!-- mapeo de rutas -->  
<servlet-mapping>  
  
<!-- todas las rutas (/) son redireccionadas al servlet de nombre dispatcher -->  
<servlet-name>dispatcher</servlet-name>  
<url-pattern>/</url-pattern>  
  
</servlet-mapping>  
  
</web-app>
```

```
<!-- WebContent/WEB-INF/spring-mvc-servlet.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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    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
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- escaneo de beans en el paquete com.tienda.main -->
    <context:component-scan base-package="com.tienda" />

    <!-- soporte para la conversión, formateo y validación en Spring -->
    <mvc:annotation-driven/>

    <!-- definición de un resolver para las plantillas -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- define el datasource de la base de datos mediante la librería C3P0 de Hibernate ->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">

        <property name="driverClass" value="com.mysql.cj.jdbc.Driver" />
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/tienda?
useSSL=false&serverTimezone=UTC" />
        <property name="user" value="tienda" />
        <property name="password" value="tienda" />

        <!-- propiedades del pool de conexiones para C3P0 -->
        <property name="initialPoolSize" value="5"/>
        <property name="minPoolSize" value="5" />
        <property name="maxPoolSize" value="20" />
        <property name="maxIdleTime" value="30000" />

    </bean>

    <!-- configura el session factory de Hibernate -->
    <bean id="sessionFactory"
    class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

        <!-- apunta al id del datasource definido previamente -->

```

```

<property name="dataSource" ref="dataSource" />

<!-- apunta al paquete que va a contener las entidades -->
<property name="packagesToScan" value="com.tienda.entidades" />

<property name="hibernateProperties">
  <props>

    <!-- dialecto SQL utilizado por Hibernate (MySQL8Dialect recomendable para
últimas versiones de MySQL y MariaDB)-->
    <!-- también puede utilizarse el dialecto MySQL5InnoDBDialect -->
    <prop key="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect </prop>

    <!-- crea/actualiza las tablas de la base de datos -->
    <prop key="hibernate.hbm2ddl.auto">update</prop>

    <!-- muestra las consultas SQL realizadas por Hibernate -->
    <prop key="hibernate.show_sql">true</prop>

  </props>
</property>
</bean>

<!-- configura el transaction manager de Hibernate -->
<bean id="myTransactionManager"
  class="org.springframework.orm.hibernate5.HibernateTransactionManager">

  <!-- apunta al session factory creado previamente -->
  <property name="sessionFactory" ref="sessionFactory"/>

</bean>

<!-- habilita la configuración de anotaciones de transacción -->
<tx:annotation-driven transaction-manager="myTransactionManager" />

<!-- agregar soporte para recursos web: css, images, js, etc ... -->
<mvc:resources location="/resources/" mapping="/resources/**"></mvc:resources>

</beans>

```

Testear entorno

- Para testear el entorno puede crearse una controlador y una página jsp que devuelva los datos al cliente.

```

// ClienteControlador.java

package com.tienda.controlador;

```

```

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/cliente")
public class ClienteControlador {

    @RequestMapping("/lista")
    public String listaClientes(Model model) {
        return "lista-clientes";
    }
}

```

```

<!-- WebContent/WEB-INF/view/lista-clientes.jsp -->

<!DOCTYPE html>

<html lang="es">

    <head>
        <title>Lista de clientes</title>
    </head>

    <body>
        Lista de clientes
    </body>

</html>

```

```

# suponiendo que el nombre del proyecto es tienda
http://localhost:8080/tienda/cliente/lista

```

Objeto de acceso a datos

- Para cada entidad de datos generalmente se crea una clase auxiliar para generar un objeto de acceso a datos (DAO) que recoja la lógica de programación asociada con las operaciones CRUD de la entidad.
- Por tanto, para cada entidad debería crearse cinco archivos:
 - Bean de la entidad (con la anotación @Entity)
 - Interfaz DAO
 - Implementación DAO (con la anotación @Repository)

- Controlador (con la anotación @Controller)
- Página jsp

```
// Cliente.java
```

```
package com.tienda.entidades;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="cliente")
public class Cliente {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="apellidos")
    private String apellidos;

    @Column(name="email")
    private String email;

    public Cliente() {

    }

    public Cliente(String nombre, String apellidos, String email) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

@Override
public String toString() {
    return "Cliente [id=" + id + ", firstName=" + nombre + ", lastName=" + apellidos +
        ", email=" + email + "]";
}
```

```
// ClienteDAO.java

package com.tienda.dao;

import java.util.List;

import com.tienda.entidades.Cliente;

public interface ClienteDAO {
    public List<Cliente> getClientes();
}
```

```
// ClienteDAOImpl.java

package com.tienda.dao;
```

```
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.tienda.entidades.Cliente;

// la anotación @Repository suele utilizarse para las clases DAO (permite inyectar
posteriormente un objeto de esta clase en el controlador)

@Repository
public class ClienteDAOImpl implements ClienteDAO {

    // se inyecta el sessionFactory de Hibernate (configurado en el archivo spring-mvc-
    servlet.xml.xml)
    // el sessionFactory depende de un datasource (también configurado en el archivo
    spring-mvc-servlet.xmlg.xml)
    // el nombre del objeto debe coincidir con el id del bean asociado (sessionFactory en
    este caso)
    @Autowired
    private SessionFactory sessionFactory;

    @Override

    // la anotación @Transactional inicia y finaliza una transacción de Hibernate de forma
    automática (no será necesario escribir session.beginTransaction() y
    session.getTransaction().commit())

    @Transactional
    public List<Cliente> getClientes() {

        Session currentSession = sessionFactory.getCurrentSession();

        // crea la consulta, la ejecuta y la retorna
        Query<Cliente> query = currentSession.createQuery("from Cliente order by
apellidos",
                Cliente.class);
        return query.getResultList();
    }
}
```

```
// ClienteControlador.java

package com.tienda.controlador;

import java.util.List;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import com.tienda.dao.ClienteDAO;
import com.tienda.entidades.Cliente;

@Controller
@RequestMapping("/cliente")
public class ClienteControlador {

    // inyección de un objeto con la interfaz ClienteDAO
    @Autowired
    private ClienteDAO clienteDAO;

    @RequestMapping("/lista")
    public String listaClientes(Model model) {

        // obtiene los clientes
        List<Cliente> clientes = clienteDAO.getClientes();

        // se añaden los clientes al modelo
        model.addAttribute("clientes", clientes);

        return "lista-clientes";
    }
}

```

```

<!-- WebContent/WEB-INF/view/lista-clientes.jsp -->

<!-- soporte para etiquetas JSTL -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>

<html lang="es">

<head>

    <title>Lista de clientes</title>

    <!-- referencia a archivo CSS de estilos (WebContent/resources/css/styles.css) -->
    <!-- la ruta de los archivos estáticos está establecida en el archivo spring-mvc-servlet.xml -->
    <link
        type="text/css" rel="stylesheet"
        href="${pageContext.request.contextPath}/resources/css/style.css"
    />

```

```

</head>

<body>

<h2>Clientes</h2>

<table>

<thead>
<tr>
<th>Nombre</th>
<th>Apellidos</th>
<th>Email</th>
</tr>
</thead>

<tbody>

<c:forEach var="cliente" items="${clientes}">

<tr>
<td> ${cliente.nombre} </td>
<td> ${cliente.apellidos} </td>
<td> ${cliente.email} </td>
</tr>

</c:forEach>
</tbody>

</table>

</body>

</html>

```

Agregar página de inicio

- Puede establecerse una página de inicio para evitar un error 404 en la ruta raíz.
- Para ello se establece la etiqueta *welcome-file-list* en el archivo web.xml

```

<!-- web.xml -->

<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

```

- A continuación se crea el archivo de la página de inicio (por ejemplo, una redirección o una lista de enlaces).

```
<!-- WebContent/index.jsp -->

<!-- redirección a la URL /tienda/cliente/lista/ -->
<% response.sendRedirect("cliente/lista"); %>
```

```
<!-- WebContent/index.jsp -->

<p>
  <a href="{pageContext.request.contextPath}/cliente/lista">Lista de clientes</a>
</p>
```

Servicios

- En aplicaciones más complejas es necesario una clase intermedia entre el controlador y los objetos de acceso (DAO).

```
ClienteControlador --> ClienteServicio -> ClienteDAO --> Cliente
                                ↓ VentaDAO --> Venta
                                ↓ LicenciaDAO --> Licencia
```

- En Spring estas clases intermedias son servicios y utilizan la anotación @Service

```
// ClienteServicio.java

package com.tienda.servicio;

import java.util.List;
import com.tienda.entidades.Cliente;

public interface ClienteServicio {
    public List<Cliente> getClientes();
}
```

```
// ClienteServicioImpl.java

package com.tienda.servicio;

import java.util.List;
```

```

import javax.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.tienda.dao.ClienteDAO;
import com.tienda.entidades.Cliente;

// los servicios utilizan el decorador @Service
@Service
public class ClienteServiceImpl implements ClienteServicio {

    // se inyectan todos los DAO a utilizar
    @Autowired
    private ClienteDAO clienteDAO;

    @Override

    // las transacciones las iniciará ahora los servicios, por lo que serán eliminadas de
    los métodos de las clases DAO
    @Transactional
    public List<Cliente> getClientes() {
        return clienteDAO.getClientes();
    }
}

```

```

// ClienteDAOImpl.java

@Repository
public class ClienteDAOImpl implements ClienteDAO {

    /* ... */

    // se elimina la anotación @Transactional porque ahora la transacción es iniciada por el
    servicio
    @Override
    public List<Cliente> getClientes() {
        /* ... */
    }

    /* ... */
}

```

- Y el controlador apuntará ahora al servicio, en lugar de al DAO.

```

@Controller
@RequestMapping("/cliente")
public class ClienteControlador {

```

@Autowired

private ClienteServicio clienteServicio;

/* ... */

@RequestMapping("/lista")

public String listaClientes(Model model) {

List<Cliente> clientes = clienteServicio.getClientes();

/* ... */

}

}

CRUD completo

- Una vez construida toda la arquitectura de clases puede comenzar a implementarse la lógica.

// ClienteControlador.java

@Controller

@RequestMapping("/cliente")

public class ClienteControlador {

/* ... */

@GetMapping("/nuevo")

public String nuevoCliente(Model model) {

model.addAttribute("cliente", new Cliente());
return "nuevo-cliente";

}

// en el JSP nuevo-cliente.jsp:

// <form:form action="guardar" modelAttribute="cliente" method="post">

@PostMapping("/guardar")

public String guardarCliente(@ModelAttribute("cliente") Cliente cliente) {

// guarda el nuevo cliente

clienteServicio.guardarCliente(cliente);

return "redirect:/cliente/lista";

}

}

<!-- nuevo-cliente.jsp -->

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<!DOCTYPE html>
<html>
<head>
    <title>Nuevo cliente</title>
</head>
<body>

<form:form action="guardar" modelAttribute="cliente" method="post">

    <label for="nombre">Nombre</label>
    <form:input type="text" path="nombre" />
    <br>

    <label for="apellidos">Apellidos</label>
    <form:input type="text" path="apellidos"/>
    <br>

    <label for="email">Email</label>
    <form:input type="text" path="email" />
    <br>

    <input type="submit" />

</form:form>
</body>
</html>
```

```
// ClienteServicio.java

public interface ClienteServicio {

    public List<Cliente> getClientes();
    public void guardarCliente(Cliente cliente);
}
```

```
// ClienteServiceImpl.java

@Service
public class ClienteServiceImpl implements ClienteServicio {

    @Autowired
    private ClienteDAO clienteDAO;
```

```
/* ... */  
  
@Override  
@Transactional  
public void guardarCliente(Cliente cliente) {  
    clienteDAO.guardarCliente(cliente);  
}  
}
```

```
// ClienteDAO.java  
  
public interface ClienteDAO {  
  
    public List<Cliente> getClientes();  
    public void guardarCliente(Cliente cliente);  
}
```

```
// ClienteDAOImpl.java  
  
@Repository  
public class ClienteDAOImpl implements ClienteDAO {  
  
    @Autowired  
    private SessionFactory sessionFactory;  
  
    /* ... */  
  
    @Override  
    public void guardarCliente(Cliente cliente) {333  
        Session session = sessionFactory.getCurrentSession();  
        session.save(cliente);  
    }  
}
```

Tu carrera digital ~

Módulo 5

Spring e Hibernate

Spring con Maven



Spring con Maven

- ◆ Instrucciones
- ◆ Instrucciones sin archivos de configuración XML
- ◆ Problemas con la importación de proyectos
- ◆ Renombrado de proyectos

Instrucciones

- ◆ La importación de librerías es más sencillo con Maven añadiendo las dependencias en el archivo *pom.xml*
- ◆ Existen muchos arquetipos básicos para comenzar un proyecto en Spring MVC(buscar por spring-mvc-). Sin embargo, también se puede crear un proyecto con Spring a partir de un arquetipo de aplicación web: maven-archetype-webapp (org.apache.maven.archetypes).
- ◆ El error que aparece al crearse el proyecto se resuelve añadiendo la dependencia de los Servlet en el archivo *pom.xml*. Es posible que sea necesario actualizar el proyecto o reiniciar Eclipse para eliminar los errores.

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
```

- ◆ Es necesario actualizar el *Build path* para que Maven tome la versión del JDK instalada en el sistema.
- ◆ También debe cambiarse la versión de Java en el archivo *pom.xml*

```
<!-- pom.xml -->

<!-- ... -->

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <!-- versión 11 del JDK -->
  <maven.compiler.source>1.11</maven.compiler.source>
  <maven.compiler.target>1.11</maven.compiler.target>
</properties>

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

- Se debe asegurar que se está utilizando la última versión del estándar Servlet y también la versión de Java desde las preferencias del proyecto (Botón derecho en el proyecto - Preferences - Project faces). Si no es posible cambiar la versión desde aquí, entonces debe abrirse el archivo `.settings/org.eclipse.wst.common.project.facet.core.xml` del proyecto y modificar manualmente la versión de facet. Tras reiniciar Eclipse, aparecerá la versión correcta en las preferencias del proyecto.

```
<!-- .settings/org.eclipse.wst.common.project.facet.core.xml -->

<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>

<!-- es necesario modificar la versión del facet jst.web -->
  <installed facet="jst.web" version="4.0"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
  <installed facet="java" version="11"/>
</faceted-project>
```

- Seguidamente se crea el directorio `java` dentro de `src/main` (si no existe ya). Al actualizar el proyecto (ALT+F5), aparecerá el directorio dentro de `Java Resources`.
- A continuación se añaden todas las librerías necesarias en el `pom.xml`: Spring MVC, Hibernate, JSTL y AspectJ

```
<?xml version="1.0" encoding="UTF-8"?>

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

  <groupId>com.tienda</groupId>
  <artifactId>tienda-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>tienda-maven Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.11</maven.compiler.source>
    <maven.compiler.target>1.11</maven.compiler.target>
```

```
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.3.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>5.3.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>5.3.2</version>
    </dependency>
```

```
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet.jsp/javax.servlet.jsp-api -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.27.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.22</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.5</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.9.6</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.6</version>
    <scope>runtime</scope>
</dependency>

</dependencies>

<build>
    <finalName>tienda-maven</finalName>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven
        defaults (may be moved to parent pom) -->
        <plugins>
            <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-war-
                plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.1</version>
            </plugin>

            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
                <version>3.1.0</version>
            </plugin>
            <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->
            <plugin>
                <artifactId>maven-resources-plugin</artifactId>
                <version>3.0.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.1</version>
            </plugin>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-install-plugin</artifactId>
                <version>2.5.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-deploy-plugin</artifactId>
                <version>5.13</version>
            </plugin>
```

```
<version>2.8.2</version>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

- Por último, se modifica el archivo web.xml para adaptarlo a un proyecto Spring, el archivo spring-mvc-servlet.xml y las vistas en el directorio view (todo ello dentro del directorio src/main/WEB-INF)

Instrucciones sin archivos de configuración XML

- En este caso será necesario indicar a Maven que no se utilizará un archivo web.xml. Para ello se añade un plugin (maven-war-plugin) en el pom.xml (si no se encuentra ya añadido como *)

```
<!-- pom.xml -->

<!-- ... -->

<plugins>

    <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-war-plugin --
->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.1</version>
    </plugin>

    <!-- ... -->

</plugins>

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

- Se realizarán las siguientes sustituciones de archivos de configuración XML a clases Java:
 - web.xml --> Spring @Configuration
 - spring-mvc-demo-servlet.xml --> Spring Dispatcher Servlet Initializer

```
// AppConfig.java

package com.tienda.config;

import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

// anotaciones de configuración para Spring MVC
@Configuration
@EnableWebMvc
@ComponentScan(basePackages={"com.tienda.entidades", "com.tienda.aspect"})
public class AppConfig {

    // define un bean para resolver las vistas
    @Bean
    public ViewResolver viewResolver() {

        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

        // indica dónde se ubicarán las vistas y qué extensión tendrán
        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");

        return viewResolver;
    }

}
```

```
// SpringInitializer.java

package com.tienda.config;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletIniti
alizer;

// será necesario sobrescribir los métodos de la clase
AbstractAnnotationConfigDispatcherServletInitializer
public class SpringInitializer extends AbstractAnnotationConfigDispatcherServletInitializer
{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    // en este método habrá que indicar dónde se encuentra la clase previa de
    // configuración
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { AppConfig.class };
    }
}
```

```
}
```



```
@Override
protected String[] getServletMappings() {
    return new String[] { "/" };
}
```

- Los dos métodos anteriores *getServletConfigClasses* y *getServletMappings* representan la siguiente configuración de Spring.

```
<!-- spring-mvc-servlet.xml -->

<!-- ... -->

<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-mvc-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

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

- A continuación puede crearse un controlador de Spring.

```
// DemoControlador.java

package com.tienda.controlador;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class DemoControlador {

    @GetMapping("/")
    public String index() {
        return "index";
    }
}
```

```
}
```

```
<!-- src/main/webapp/WEB-INF/views/index.jsp -->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
Hola
</body>
</html>
```

- Y eliminar el archivo *web.xml* que es creado inicialmente, así como también el archivo *index.jsp*

```
<!-- pom.xml -->

<?xml version="1.0" encoding="UTF-8"?>

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

  <groupId>com.tienda</groupId>
  <artifactId>tienda-maven-noxml</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>tienda-maven Maven Webapp</name>
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.11</maven.compiler.source>
    <maven.compiler.target>1.11</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
```

```
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.11</version>
<scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aop -->
<dependency>
```

```
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>5.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet.jsp/javax.servlet.jsp-api -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.27.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.22</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.5</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.9.6</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
```

```

<version>1.9.6</version>
<scope>runtime</scope>
</dependency>

</dependencies>

<build>
  <finalName>tienda-maven-noxml</finalName>
  <pluginManagement>
    <plugins>
      <plugin>
        <!-- Add Maven coordinates (GAV) for: maven-war-plugin -->
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.0</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>

```

- También es conveniente asegurarse que el contexto web esté bien establecido desde las opciones del proyecto (Botón derecho en el proyecto - *Preferences - Web Project Settings*)
- Una buena solución para resolver posibles errores con Maven es eliminar manualmente todas las librerías almacenados en el repositorio local (`~/.m2/repositories`).

Problemas con la importación de proyectos

- En la importación de proyectos puede producirse errores con la versión del estándar Servlet. Para solucionarlo debe abrirse el archivo `.settings/org.eclipse.wst.common.project.facet.core.xml` del proyecto y modificar manualmente la versión de facet. Tras reiniciar Eclipse, aparecerá la versión correcta en las preferencias del proyecto.

```

<!-- .settings/org.eclipse.wst.common.project.facet.core.xml -->

<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>

  <!-- es necesario modificar la versión del facet jst.web -->
  <installed facet="jst.web" version="4.0"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
  <installed facet="java" version="11"/>
</faceted-project>

```

Renombrado de proyectos

- ◆ Cuando se renombra un proyecto hay que modificar otros valores de determinados archivos para evitar problemas.
 - pom.xml: artifactId
 - pom.xml: finalName
 - Preferences - Web Project Settings
 - AppConfig: en caso de que haya cambios en los nombres de paquete -->

```
@ComponentScan(basePackages="...")
```

Tu carrera digital ~



VISIÓN DIGITAL
PRIMERA DÉCADA
DE ASUNTOS ECONÓMICOS
Y TRANSPORTE Y DIGITALIZACIÓN



SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
Y INTELIGÉNCIA ARTIFICIAL

red.es



GARANTÍA
SOCIAL



Adecco

"El FSE invierte en tu futuro"
Fondo Social Europeo