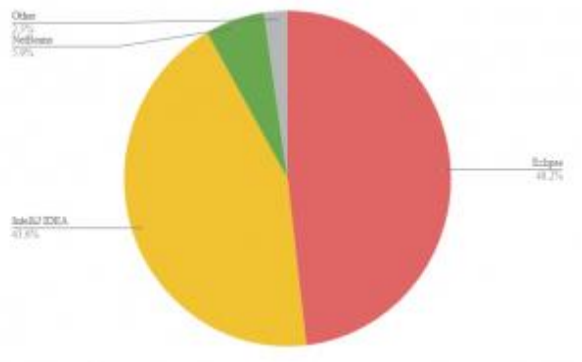
# Herramientas de Desarrollo

## Entornos de Desarrollo Integrados (IDE)

Un entorno de desarrollo integrado es una aplicación software que ayuda en la tarea de desarrollo de software integrando una serie de herramientas enfocadas en dicho fin. La cantidad de herramientas incluidas es configurable dependiendo de las características de las aplicaciones que se quieran crear.



### Componentes de un IDE

Por lo general los entornos de desarrollo actuales ofrecen:

- **Editor de texto**: facilita la escritura organizando las instrucciones atendiendo a un formato, colores, permite la escritura automática, autocompletar, además de las clásicas funciones de buscar, reemplazar cortar, pegar, etc.
- **Compilador/Interprete**: normalmente los IDEs incluyen herramientas de compilación y ejecución propias, sin tener necesidad de un jdk.
- **Depurador**: Herramienta que permite la ejecución de una aplicación instrucción a instrucción, y nos ayuda a examinar las distintas situaciones y cambios que se produzcan en las variables del programa.
- **Asistente para GUI**: (GUI - Interfaz gráfica de usuario). Normalmente es una paleta de componentes que me permite arrastrar y soltar (drag and drop) componentes gráficos a un panel que corresponde a la ventana que vamos a crear.
- **Control de versiones**: permite almacenar cambios en nuestro código y compartirlo con otros desarrolladores.
- **Exportar programas**: permitirá exportar nuestra aplicación completa en algún formato de fichero.

### Frameworks

Una aplicación informática requiere que su información esté estructurada de un modo que entienda para poder gestionarla. Cuando los desarrolladores queremos crear una aplicación usamos y generamos distinto tipo de información: código fuente, ficheros de configuración, ficheros binarios de datos, librerías, ficheros ejecutables, etc. A veces nos centramos únicamente en el tipo de tecnología que vamos a usar, pero nos despreocupamos a la hora de crear infinidad de propios recursos, de ficheros y de código fuente para crear nuestra aplicación, pero sin ningún orden u organización.

Un **framework** es un esquema (una estructura, un patrón) para el desarrollo de una aplicación. Se puede entender como una estructura conceptual y tecnológica que nos ayuda a través de un conjunto de herramientas, librerías, módulos, para crear aplicaciones de un tipo concreto. No están ligados a un lenguaje concreto, pero cuando más detallado es el *framework*, más necesidad tendrá de vincularse a un lenguaje concreto.

Puede ser algo muy sencillo como el patrón MVC, que indica que separemos las clases de nuestro programa en las capas de presentación (vista), los datos(modelo) y las operaciones (controlador). Pero también tenemos otros *frameworks* que llegan a definir los nombres de los ficheros, su estructura, las convenciones de programación, etc. También es posible que el framework defina una estructura para una aplicación completa, o bien sólo se centre en un aspecto de ella.


Ejemplos de frameworks:

- JUnit es un *framework* para el desarrollo de pruebas unitarias de software. Está formado por una serie de librerias.
- Java Server Pages es un *framework* para crear aplicaciones web con Java, principalmente la interfaz de usuario.
- Xamarin es un *framework* para crear aplicaciones móviles para Android, IOS en el lenguaje C#.
- Hibernate es un *framework* para Java para facilitar el trabajo con bases de datos relacionales desde la programación orientada a objetos.
- .NET es un *framework* con soporte para múltiples lenguajes de programación y está enfocado en la creación de aplicaciones para cualquier sistema Windows.

## Evolución histórica

En las décadas de utilización de la tarjeta perforada como sistema de almacenamiento el concepto de Entorno de Desarrollo Integrado sencillamente no tenía sentido. Los programas estaban escritos con diagramas de flujo y entraban al sistema a través de las tarjetas perforadas. Posteriormente, eran compilados.

El primer lenguaje de programación que utiliza un IDE fue el BASIC (que fue el primero en abandonar también las tarjetas perforadas o las cintas de papel). Éste primer IDE estaba basado en consola de comandos exclusivamente (normal por otro lado, si tenemos en cuenta que hasta la década de los 90 no entran en el mercado los sistemas operativos con interfaz gráfica).

Sin embargo, el uso que hace de la gestión de archivos, compilación, depuración… es perfectamente compatible con los IDE actuales. A nivel popular, el primer IDE puede considerarse que fue el IDE llamado Maestro. Nació a principios de los 70 y fue instalado por unos 22000 programadores en todo el mundo. Lideró el campo durante los años 70 y 80. El uso de los entornos integrados de desarrollo se ratifica y afianza en los 90 y hoy en día contamos con infinidad de IDE, tanto de licencia libre como no.

| Tabla de los IDE más relevantes hoy en día | | |
|---|---|---|
| **Entorno de desarrollo** | **Lenguajes que soporta** | **Tipo de licencia** |
| **NetBeans** | C/C++, Java, JavaScript, PHP, Python. | De uso público. |
| **Eclipse** | Ada, C/C++, Java, JavaScript, PHP. | De uso público. |
| **Microsoft Visual Studio.** | Basic, C/C++, C#. | Propietario. |
| **C++ Builder.** | C/C++. | Propietario. |
| **JBuilder.** | Java. | Propietario. |

No hay unos entornos de desarrollo más importantes que otros. La elección del IDE más adecuado dependerá del lenguaje de programación que vayamos a utilizar para la codificación de las aplicaciones y el tipo de licencia con la que queramos trabajar.

**AUTOEVALUACIÓN:**

**Un entorno integrado de desarrollo está compuesto por:**

- Editor de código y traductor.
- Interfaz gráfica, editor de código y depurador.
- Editor de código, compilador, intérprete, depurador e interfaz gráfica.
- Interfaz gráfica, editor de código y depurador.

## Entornos integrados libres y propietarios

# Entornos Integrados Libres

Son aquellos con licencia de uso público.

No hay que pagar por ellos, y aunque los más conocidos y utilizados son Eclipse y NetBeans, hay bastantes más.

| IDE | Lenguajes que soporta | Sistema Operativo |
|---|---|---|
| **NetBeans.** | C/C++, Java, JavaScript, PHP, Python. | Windows, Linux, Mac OS X. |
| **Eclipse.** | Ada, C/C++, Java, JavaScript, PHP. | Windows, Linux, Mac OS X. |
| **Gambas.** | Basic. | Linux. |
| **Anjuta.** | C/C++, Python, Javascript. | Linux. |
| **Geany.** | C/C++, Java. | Windows, Linux, Mac OS X. |

# Entornos Integrados Propietarios

Son aquellos entornos integrados de desarrollo que necesitan licencia. No son free software, hay quepagar por ellos. El más conocido y utilizado es Microsoft Visual Studio, que usa el framework .NET y es desarrollado por Microsoft.

| IDE | Lenguajes que soporta | Sistema Operativo |
|---|---|---|
| **Microsoft Visual Studio.** | Basic, C/C++, C#. | Windows. |
| **FlashBuilder.** | ActionScript. | Windows, Mac OS X. |
| **C++ Builder.** | C/C++. | Windows. |
| **Turbo C++ profesional.** | C/C++. | Windows. |
| **JBuilder.** | Java. | Windows. |
| **JCreator.** | Java. | Windows, Linux, Mac OS X. |
| **Xcode.** | C/C++, Java. | Mac OS X. |

Si queremos saber más:

**https://en.wikipedia.org/wiki/Integrated_development_environment**

**AUTOEVALUACIÓN:**

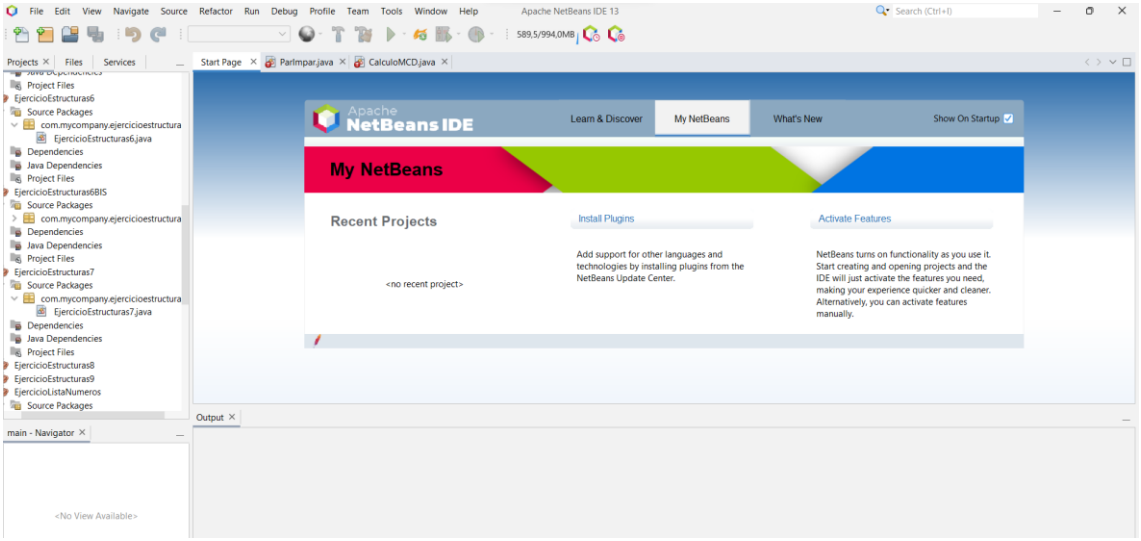
**Relaciona los siguientes entornos de desarrollo con sus características correspondientes:**

- Microsoft Visual Studio.
- NetBeans.
- C++ Builder.

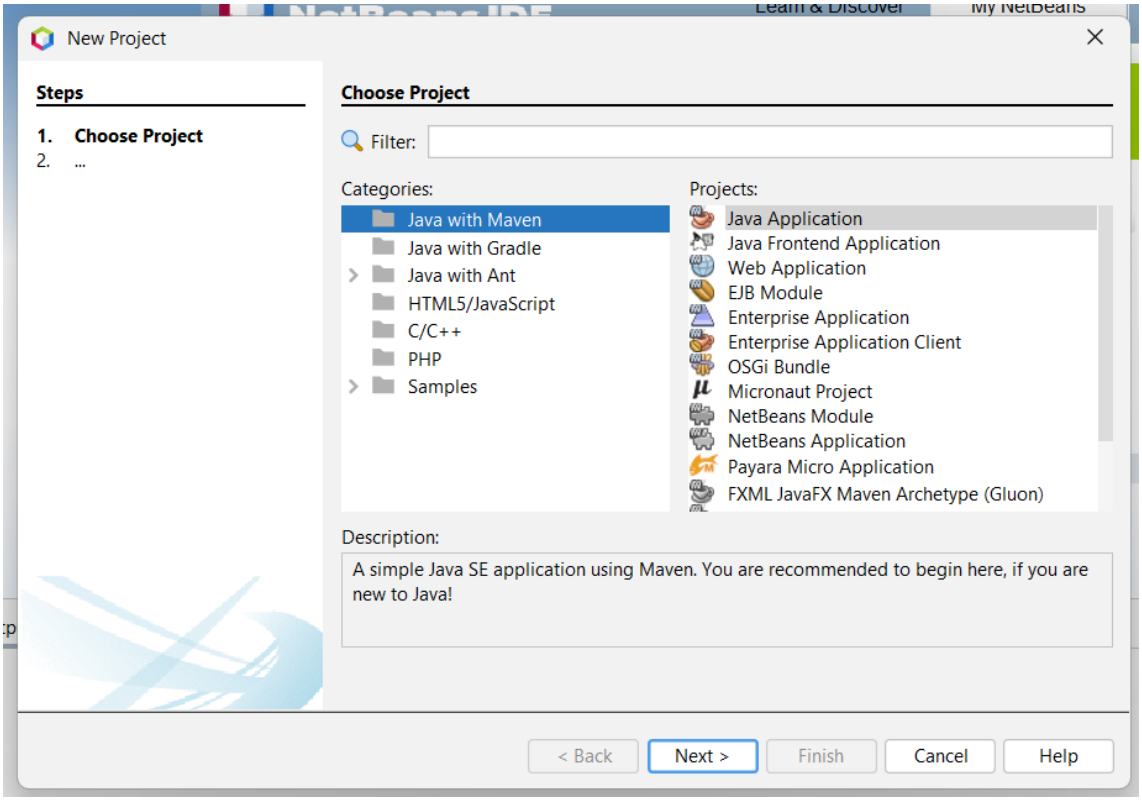
1. Libre. Soporta C/C++, Java, PHP, Javascript, Python
2. Propietario. Soporta Basic, C/C++, C#
3. Propietario. Soporta C/C++

# Instalación NetBeans

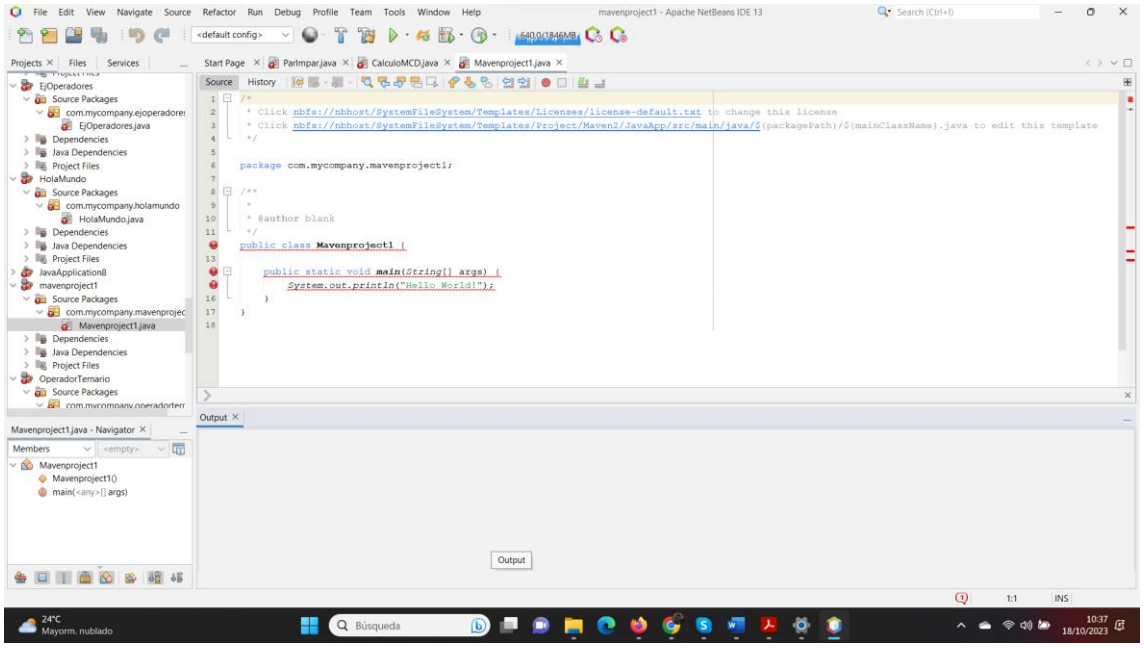
Una vez tengamos instalado NetBeans ya podemos comenzar a utilizarlo.



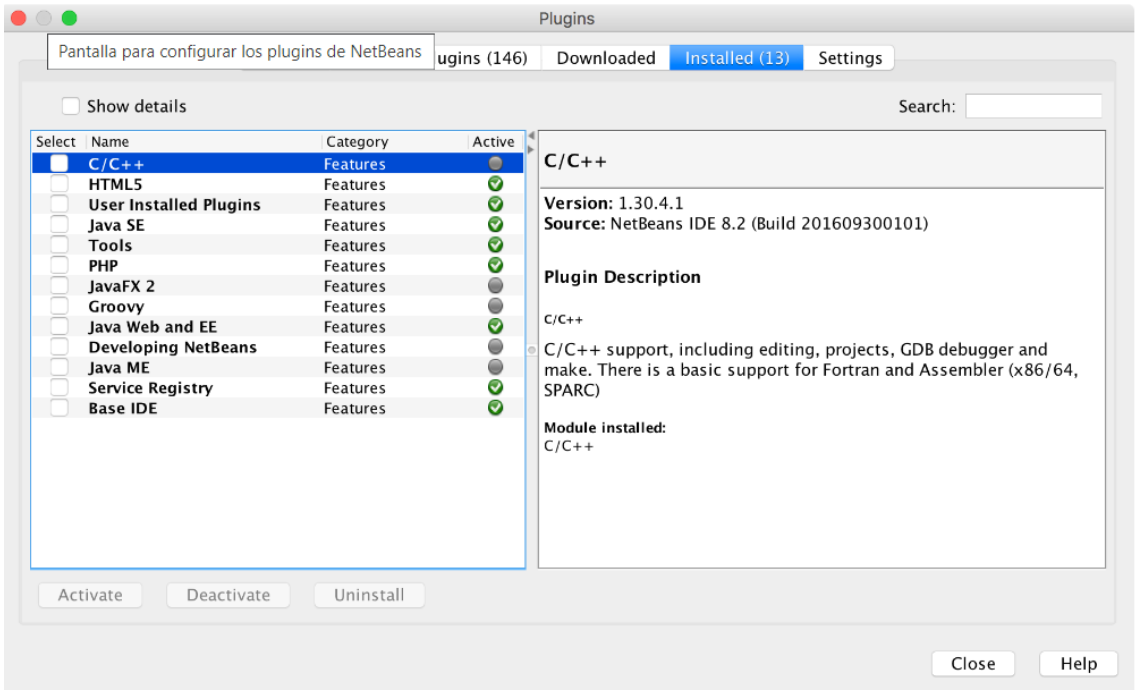

Una vez hemos arrancado Netbeans por primera vez, podemos crear un proyecto accediendo al menú File->New Proyect y nos aparecerán los distintos tipos de proyectos que podemos crear.



Al seleccionar el proyecto de tipo Java, NetBeans nos creará la estructura básica del mismo y ya estaremos preparados para comenzar a programar nuestro proyecto.



Una vez tenemos instalado NetBeans, podemos instalarle algún plugins desde la sección "Plugins" colocada en la zona superior del menú principal (Tools-> Plugins).



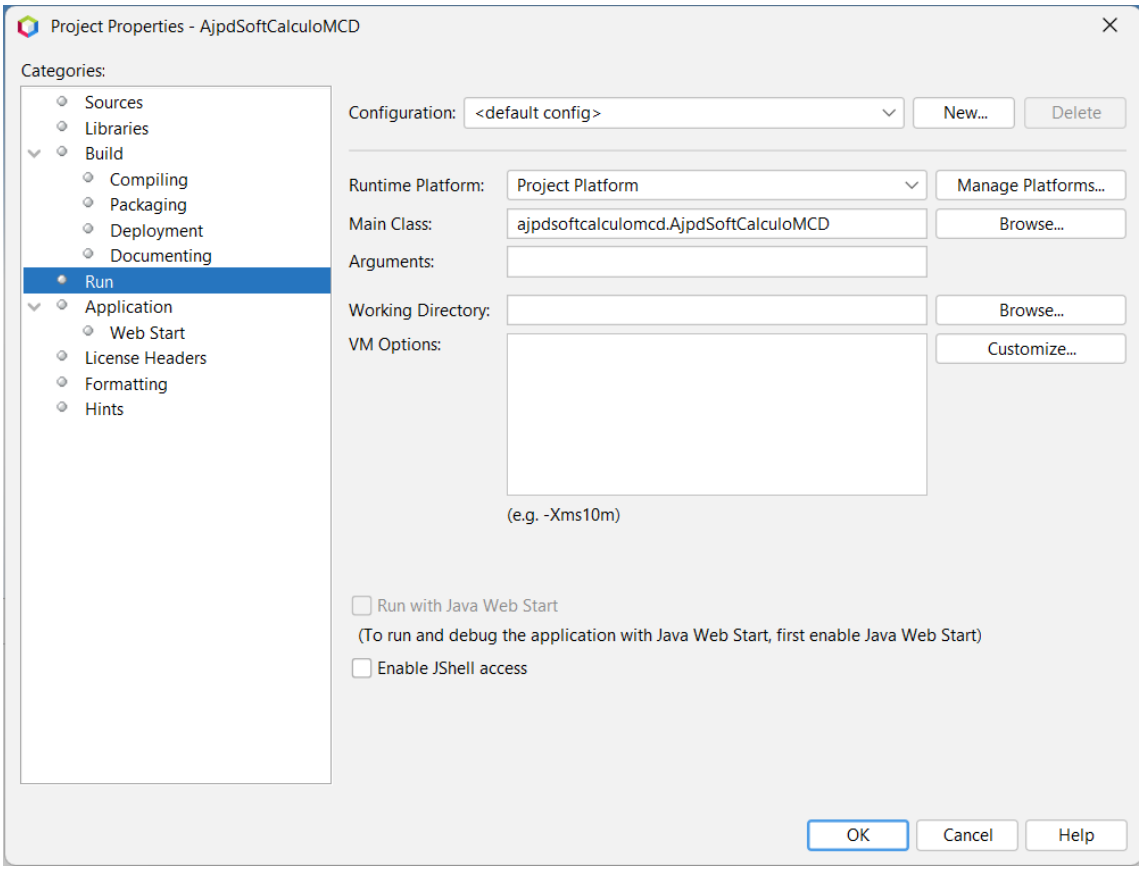
Una vez tenemos instalado nuestro entorno de desarrollo podemos acceder a personalizar su configuración.

Al abrir un proyecto existente, o bien crear un nuevo proyecto, seleccionaremos un desplegable con el nombre “configuración” desde donde podremos personalizar distintas opciones del proyecto.

Podemos personalizar la configuración del entorno sólo para el proyecto actual, o bien para todos los proyectos presentes y futuros.

Parámetros configurables del entorno:

- ✓ Carpeta/s donde se alojarán todos los archivos de los proyectos (es importante la determinaciónde este parámetro, para tener una estructura de archivos ordenada).
- ✓ Carpetas de almacenamiento de paquetes fuente y paquetes prueba.
- ✓ Administración de la plataforma del entorno de desarrollo.
- ✓ **Opciones de la compilación de los programas**: compilar al grabar, generar información de depuración...
- ✓ **Opciones de empaquetado de la aplicación**: nombre del archivo empaquetado (con extensión .jar, que es la extensión característica de este tipo de archivos empaquetados) y momento del empaquetado.
- ✓ Opciones de generación de documentación asociada al proyecto.
- ✓ Descripción de los proyectos, para una mejor localización de los mismos.
- ✓ Opciones globales de formato del editor: número de espaciados en las sangrías, color de erroresde sintaxis, color de etiquetas, opción de autocompletado de código, propuestas de insertar automáticamente código...
- ✓ Opciones de combinación de teclas en teclado.




**LIBRERIAS o LIBRARIES:**

**Desde esta ventana podemos elegir la plataforma de la aplicación.**

**Toma por defecto el JDK, pero se puede cambiar si se quiere, siempre y cuando sea compatible con la versión de NetBeans utilizada.**

**También en esta ventana se puede configurar el paquete de pruebas que se realizará al proyecto.**

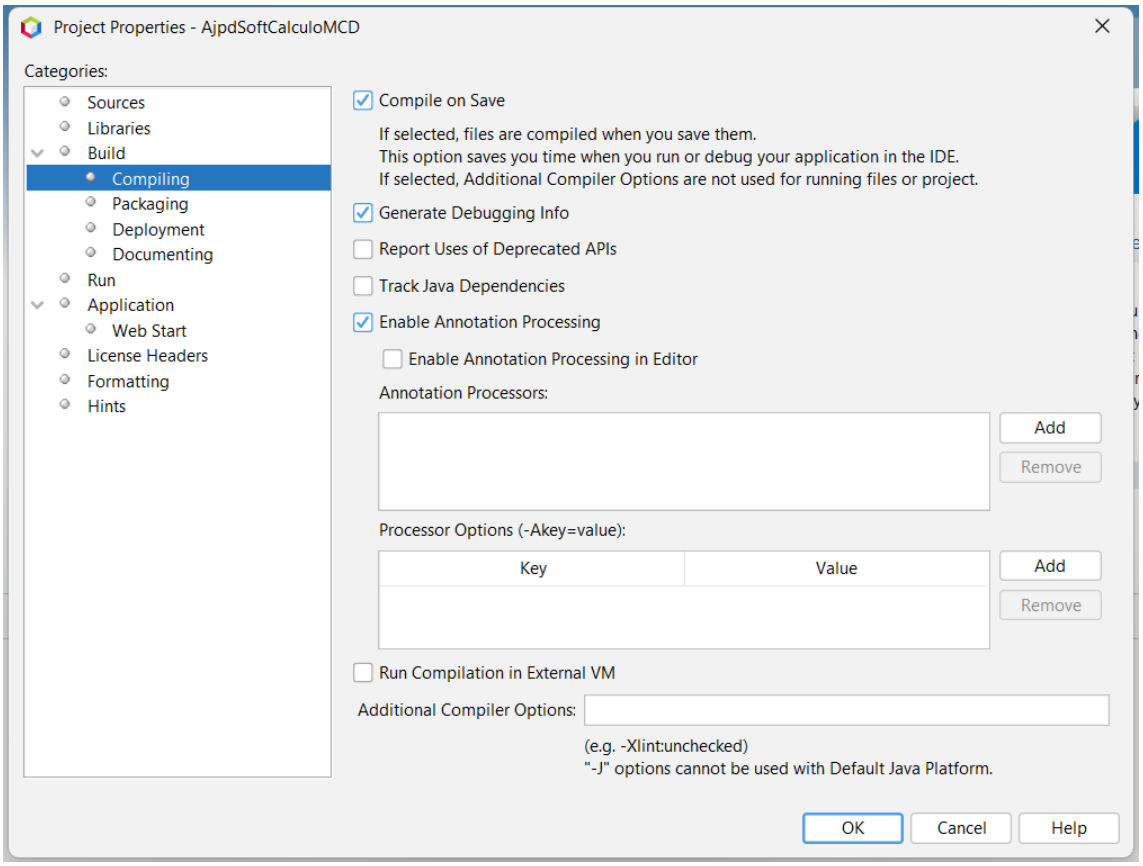
**GENERACIÓN DE CÓDIGO – COMPILANDO**

**Las opciones que nos permite modificar en cuanto a la compilación del programa son:**

- ✓ **Compilar al grabar: al guardar un archivo se compilará automáticamente.**
- ✓ **Generar información de depuración: para obtener la documentación asociada.**

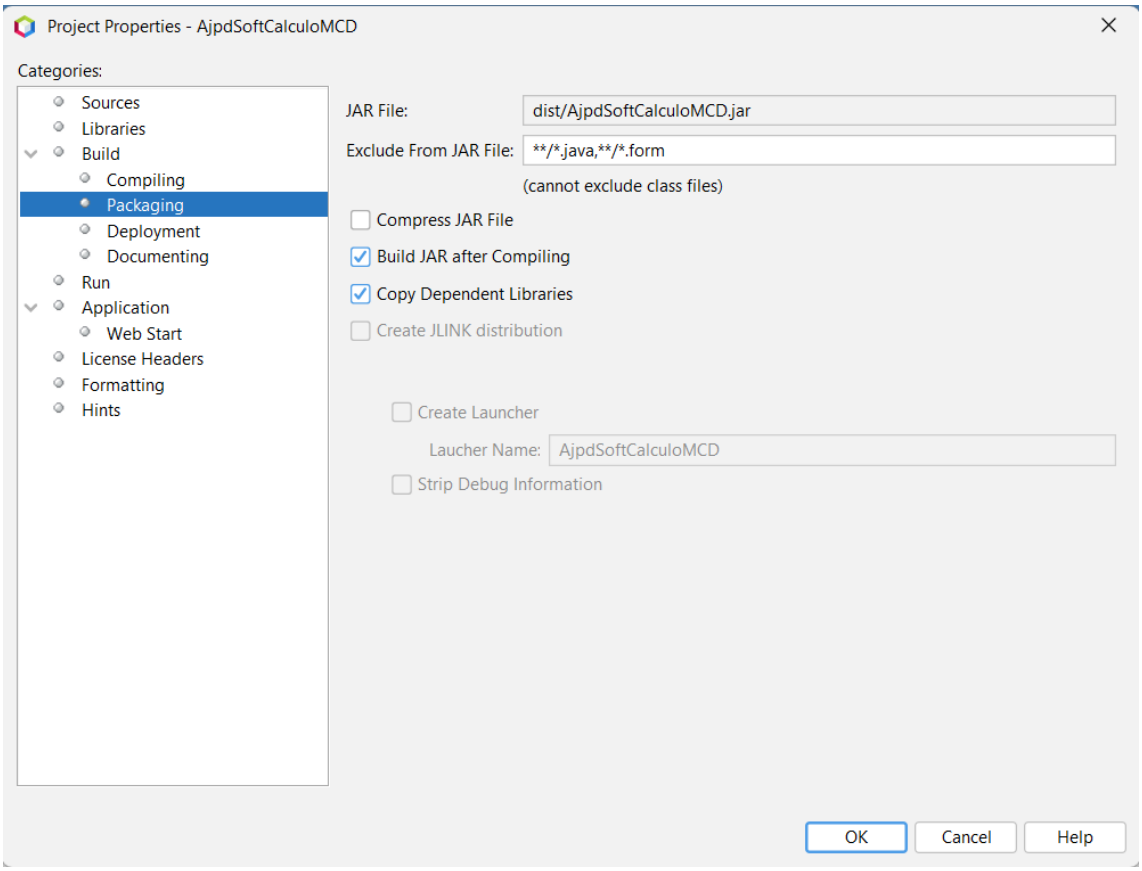
**Enable annotation processsing: permitir anotaciones durante el proceso.**

**También podemos agregar anotaciones concretas para el proceso de compilación y añadir opciones de proceso que, según las características del proyecto, puedan ser de interés para nosotros.**




## GENERACIÓN DE CÓDIGO - EMPAQUETANDO

**Las aplicaciones resultado de la compilación del código deben ser empaquetadas antes de su distribución, con objeto de tener un único archivo, generalmente comprimido, que contenga en su interior todos los archivos de instalación y configuración necesarios para que la aplicación pueda ser instalada y desarrollada con éxito por el usuario cliente.**

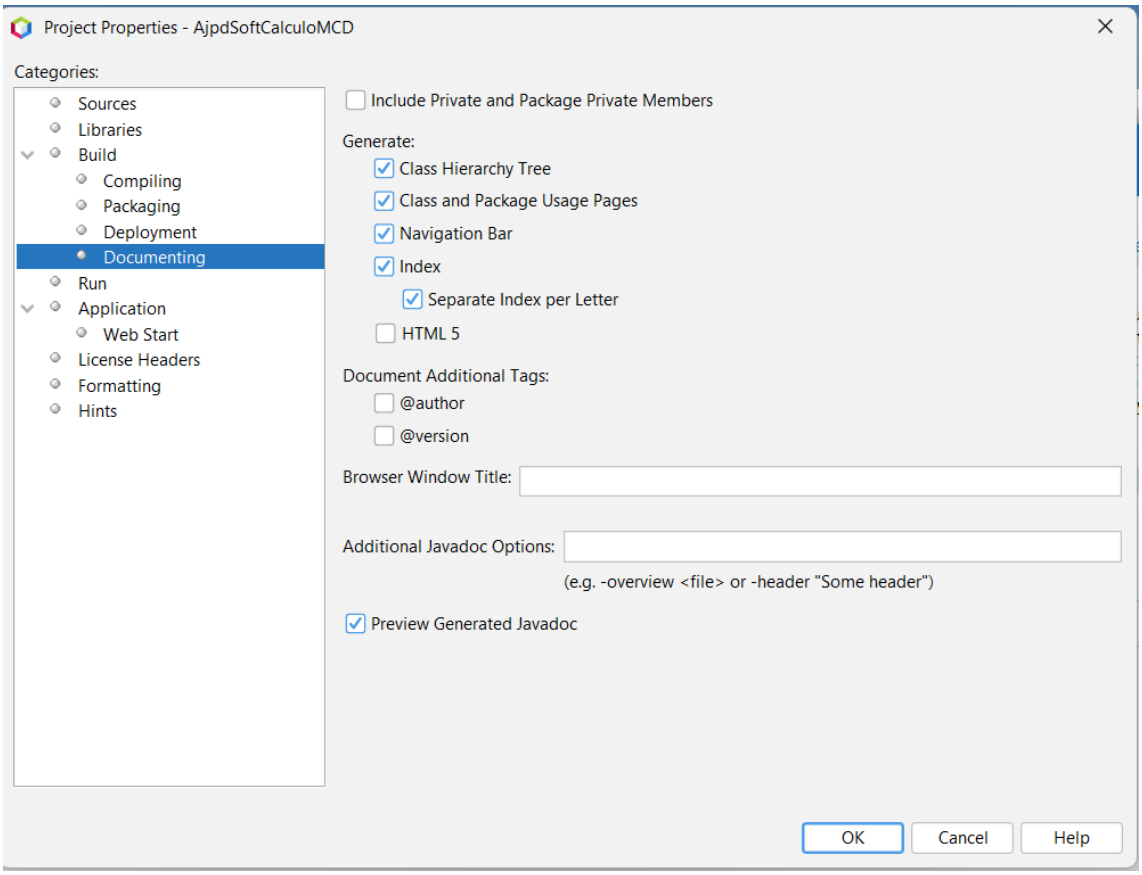



**Como vemos en la imagen, en esta opción podemos modificar el lugar donde se generará el archivo resultante del empaquetado, así como si deseamos comprimirlo. También podemos elegir que el archivo empaquetado se construya tras la compilación, que es lo habitual (por eso esta opción aparece como predeterminada).**

## GENERACIÓN DE CÓDIGO – DOCUMENTANDO

**Como ya vimos en la unidad anterior, la documentación de aplicaciones es un aspecto clave que no debemos descuidar nunca. NetBeans nos ofrece una ventaja muy considerable al permitirnos obtener documentación de la fase de codificación de los programas de forma automática.**

**Dentro del documento que se va a generar podemos elegir que se incluyan todas las opciones anteriores. Esto es lo más recomendable, por eso aparecen todas marcadas de forma predeterminada y lo mejor es dejarlo como está.**

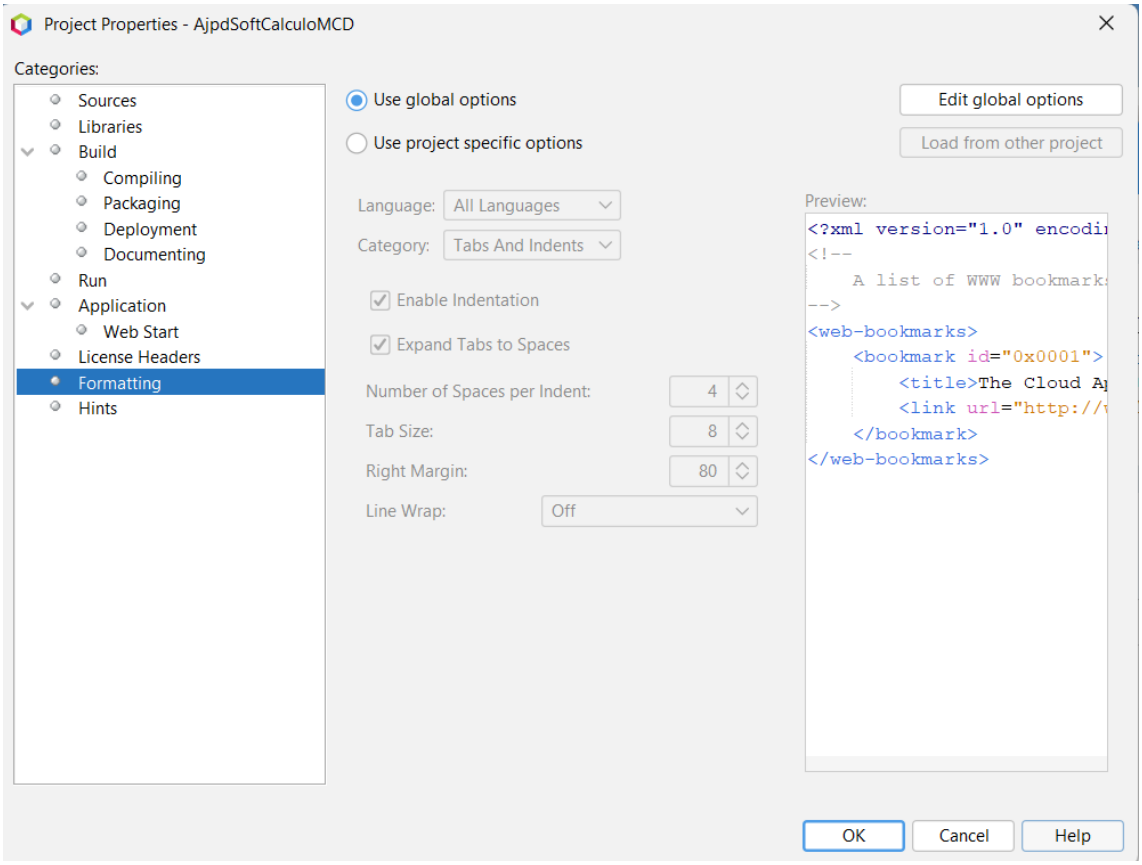



**EJECUTANDO CÓDIGO**

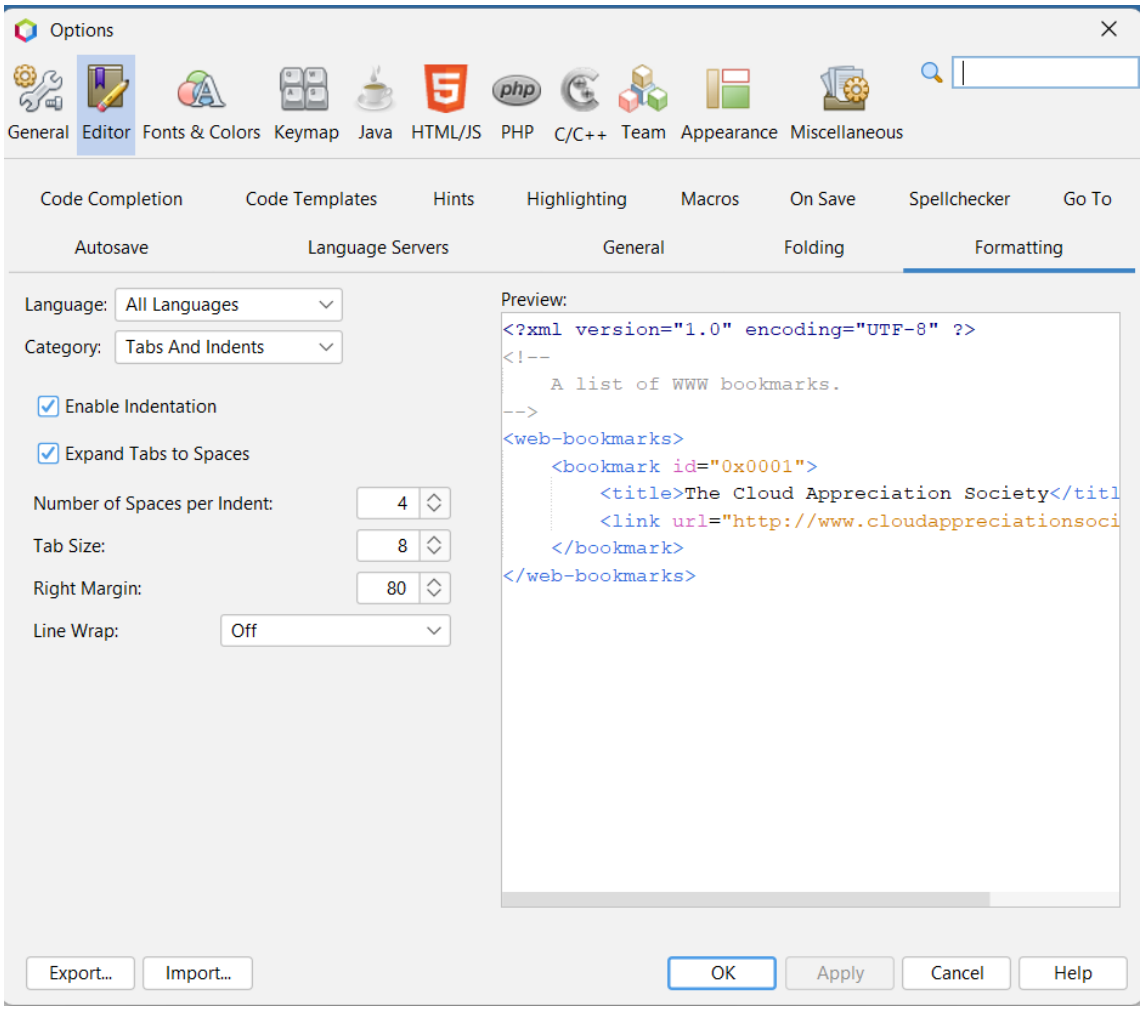
**Esta opción nos permite definir una nueva configuración de ejecución de código, elegir la clase principal, las carpetas de trabajo del proyecto y opciones de la máquina virtual.**

**FORMATO**

**Aquí podemos personalizar aspectos globales del formato del código fuente en la aplicación.**




**Si seleccionamos Editar opciones globales nos encontramos con la siguiente ventana, que tiene una barra superior de pestañas para configurar cada apartado del formato de forma independiente:**




## Gestión de módulos

Con la plataforma dada por un entorno de desarrollo como NetBeans podemos hacer uso de módulos y plugins para desarrollar aplicaciones.

En la página oficial de NetBeans encontramos una relación de módulos y plugins, divididos encategorías.

Seleccionando la categoría Lenguajes de Programación, encontraremos aquellos módulos y plugins que nos permitan añadir nuevos lenguajes soportados por nuestro IDE.

Un módulo es un componente software que contiene clases de Java que pueden interactuar con las APIs del entorno de desarrollo y el manifest file, que es un archivo especial que lo identifica como módulo.

Los módulos se pueden construir y desarrollar de forma independiente. Esto posibilita su reutilización y que las aplicaciones puedan ser construidas a través de la inserción de módulos con finalidades concretas. Por esta misma razón, una aplicación puede ser extendida mediante la adición de módulos nuevos que aumenten su funcionalidad.

Existen en la actualidad multitud de módulos y plugins disponibles para todas las versiones de los entornos de desarrollo más utilizados. En las secciones siguientes veremos dónde encontrar plugins y módulos para NetBeans 6.9.1 que sean de algún interés para nosotros

y las distintas formas de instalarlos en nuestro entorno.

También aprenderemos a desinstalar o desactivar módulos y plugins cuando preveamos que no los vamos a utilizar más y cómo podemos estar totalmente actualizados sin salir del espacio de nuestro entorno.

Veremos las categorías de plugins disponibles, su funcionalidad, sus actualizaciones…

Añadir un módulo va a provocar dotar de mayor funcionalidad a nuestros proyectos desarrollados en NetBeans. Para añadir un nuevo módulo tenemos varias opciones:

a) Añadir algún módulo de los que NetBeans instala por defecto.
b) Descargar un módulo desde algún sitio web permitido y añadirlo.
c) Instalarlo on-line en el entorno.

Por supuesto, una cuarta posibilidad es crear el módulo nosotros mismos (aunque eso no lo veremos ).

Sin embargo, lo más usual es añadir los módulos o plugins que realmente nos interesan desde la weboficial de NetBeans. El plugin se descarga en formato .nbm que es el propio de los módulos en NetBeans. Posteriormente, desde nuestro IDE, cargaremos e instalaremos esos plugins. A esta manera de añadir módulos se le conoce como adición off-line.

También es habitual instalarlos on-line, sin salir del IDE.

La adición on-line requiere tener instalado el plugin Portal Update Center en NetBeans 6.9.1 y consiste en instalar complementos desde nuestro mismo IDE, sin tener que descargarlos previamente.

A modo de ejemplo, a continuación, se explican los pasos para añadir un módulo o plugin, de forma off-line (descargando el archivo e instalándolo posteriormente) y de forma on-line.

**Hay dos formas de añadir módulos y plugins en NetBeans:**
***Off-line*: Buscar y descargar plugins desde la página web oficial de la plataforma:**
**http://plugins.netbeans.org/**
**Ejemplo:**
**Vamos a buscar un plugin para añadir posteriormente a nuestro IDE.**

**Entramos en la zona de descargas de plugins para NetBeans y en la zona del catálogo descargamos uno (el que queramos para probar)**

Cuando consideramos que algún módulo o plugin de los instalados no nos aporta ninguna utilidad, o bien que el objetivo para el cual se añadió ya ha finalizado, el módulo deja de tener sentido en nuestro entorno. Es entonces cuando nos planteamos eliminarlo.

Eliminar un módulo es una tarea trivial que requiere seguir los siguientes pasos:

- Encontrar el módulo o plugin dentro de la lista de complementos instalados en el entorno.
- A la hora de eliminarlo tenemos dos opciones:

    a) Desactivarlo: El módulo o plugin sigue instalado, pero en estado inactivo (no aparece en el entorno).
    b) Desinstalarlo: El módulo o plugin se elimina físicamente del entorno de forma permanente.

Los módulos y plugins disponibles para los entornos de desarrollo, en sus distintas versiones, tienen muchas y muy variadas funciones.
Podemos clasificar las distintas categorías de funcionalidades de módulos y plugins en los siguientes grupos:

1. Construcción de código: facilitan la labor de programación.
2. Bases de datos: ofrecen nuevas funcionalidades para el mantenimiento de las aplicaciones.
3. Depuradores: hacen más eficiente la depuración de programas.
4. Aplicaciones: añaden nuevas aplicaciones que nos pueden ser útiles.
5. Edición: hacen que los editores sean más precisos y más cómodos para el programador.
6. Documentación de aplicaciones: para generar documentación de los proyectos en la manera deseada.
7. Interfaz gráfica de usuario: para mejorar la forma de presentación de diversos aspectos delentorno al usuario.
8. Lenguajes de programación y bibliotecas: para poder programar bajo un Lenguaje de Programación que, en principio, no soporte la plataforma.
9. Refactorización: hacer pequeños cambios en el código para aumentar su legibilidad, sin alterar su función.
10. Aplicaciones web: para introducir aplicaciones web integradas en el entorno.
11. Prueba: para incorporar utilidades de pruebas al software.

Herramientas concretas

- ***Importador de Proyectos de NetBeans***: permite trabajar en lenguajes como JBuilder.
- ***Servidor de aplicaciones GlassFish***: Proporciona una plataforma completa para aplicaciones de tipo empresarial.
- ***Soporte para Java Enterprise Edition***: Cumplimiento de estándares, facilidad de uso y la mejora de rendimiento hacen de NetBeans la mejor herramienta para crear aplicaciones de tipo empresarial de forma ágil y rápida.
- ***Facilidad de uso a lo largo de todas las etapas del ciclo de vida del software***.
- ***NetBeans Swing GUI builder***: simplifica mucho la creación de interfaces gráficos de usuarios en aplicaciones cliente y permite al usuario manejar diferentes aplicaciones sin salir del IDE.
- ***NetBeans Profiler***: Permite ver de forma inmediata ver cómo de eficiente trabajará un trozo desoftware para los usuarios finales.
- ***El editor WSDL*** facilita a los programadores trabajar en servicios Web basados en XML.
- ***El editor XML Schema Editor*** permite refinar aspectos de los documentos XML de la mismamanera que el editor WSDL revisa los servicios Web.
- Aseguramiento de la seguridad de los datos mediante el ***Sun Java System Acces Manager***.
- ***Soporte beta de UML*** que cubre actividades como las clases, el comportamiento, la interacción ylas secuencias.
- ***Soporte bidireccional***, que permite sincronizar con rapidez los modelos de desarrollo con los cambios en el código conforme avanzamos por las etapas del ciclo de vida de la aplicación.

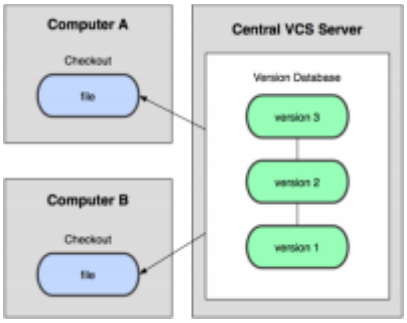
# Sistemas de Control de Versiones

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Se usan para registrar cualquier tipo de archivos, y comparar cambios a lo largo del tiempo, revertir a un estado anterior del proyecto, ver en qué momento se introdujo un error, etc.

Tenemos dos arquitecturas principales a través de la que podemos clasificar distintos SCVs:

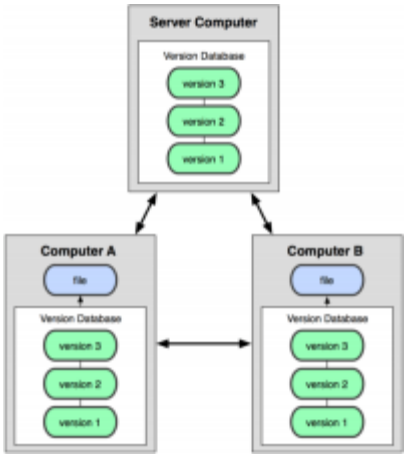
**Arquitectura Centralizada**




Los desarrolladores usan un repositorio central al que acceden mediante un cliente en su máquina. Tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Durante muchos años éste ha sido el estándar para el control de versiones. La desventaja más obvia es el servidor centralizado como único punto de fallo. Si ese servidor se cae o pierde los datos, se pierde el trabajo.

- Subversion SVN (Código Abierto)
- Visual SourceSafe (Propietario)

**Arquitectura Distribuida**




En un SCV distribuido los clientes no sólo descargan la última instantánea de los archivos: suben completamente el repositorio. Cada vez que se descarga una instantánea (versión del proyecto), en realidad se hace una copia de seguridad completa de todos los datos. Así, si un servidor muere, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo.

- GIT (Codigo Abierto)
- Mercurial (Codigo Abierto)

# GIT

Es un SCV distribuido creado como necesidad durante el desarrollo de núcleo de Linux. El creador es Linus Torvalds, creador también del núcleo de Linux.

## Servidores de Repositorios

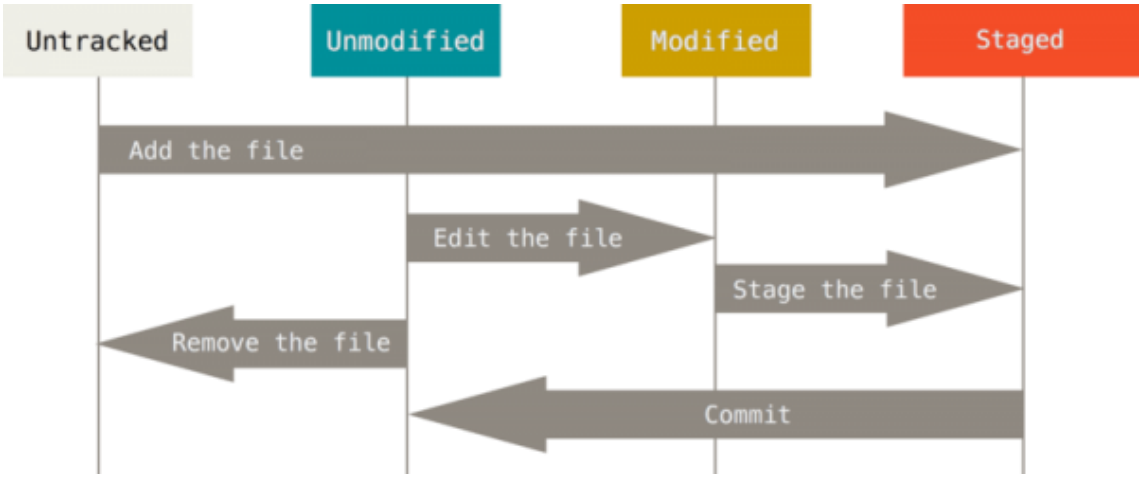
Para almacenar las versiones de nuestro código necesitamos un servidor de control de versiones basado en Git. Actualmente hay distintas opciones que ofrecen servicios comerciales y también servicios gratuitos, con algunas limitaciones, pero perfectamente válidos para trabajar en nuestros pequeños proyectos. Las versiones gratuitas de los siguientes servicios ofrecen:

- GitHub: repositorios públicos ilimitados.
- BitBucket: repositorios públicos y privados ilimitados.
- GitLab: repositorios públicos y privados ilimitados.

En nuestro caso usamos *BitBucket* porque permite la creación de repositorios privados, además de públicos como GitHub. Todos los servicios funcionan de forma similar y nos permite crear una cuenta de usuario gratuita.

## Funcionamiento de Git

Git guarda una copia completa del estado de los ficheros de nuestro proyecto en cada versión (*commit*). Esta información se almacena en la base de datos local (repositorio local), y posteriormente se puede sincronizar con otros repositorios remotos.




En Git, los ficheros tienen 2 estados principales:

1. Ficheros fuera de seguimiento (*untracked*)
2. Ficheros bajo seguimiento (*tracked*)

- **Ficheros Untracked**. Cuando creo o añado ficheros nuevos a mi repositorio local estos están fuera de seguimiento. También están fuera de seguimiento los que no estaban bajo seguimiento en la última versión; último *commit* (confirmación).

Los ficheros que están bajo seguimiento pueden tener 3 estados (unmodified, staged, modified)

- **Ficheros Staged (Preparados para confirmación)**: Cuando añado ficheros *untracked* mediante el comando (git add) para que formen parte de la siguiente confirmación, estos pasan a estar pendientes de confirmación, preparados, (staged). Estos cambios no estarán confirmados hasta que haga *commit*.
- **Ficheros unmodified (Sin modificar)**: Justo despues de hacer un ***commit*** (crear una versión), todos los ficheros que estaban pendientes de confirmación (staged), se almacenan en la base de datos local de Git. En ese momento todos los ficheros están sin ninguna modificación, que quiere decir que no hay modificaciones desde la última versión.
- **Ficheros modified (Modificados)**: Si modifico un fichero *unmodified*, este pasará a estado ***modified*** indicando que debo de volver a añadirlo (git add) parar prepararlo (staged) para la siguiente confirmación (commit).

Todo este trabajo se realiza en el repositorio local y se almacena en la base de datos local de Git. Independientemente de esto, puedo sincronizar el estado de mi repositorio local (directorio en mi equipo), con un repositorio remoto (fuera de mi equipo).

## Instalación y configuración de Git

Al configurar GIT por **primera vez**, como mínimo debemos configurar nuestra identidad para realizar *commits* (Confirmar una versión). Esto se debe hacer una sola vez, después de instalar Git. Para ello abrimos un terminal *Bash* y ejecutamos:

```
git config --global user.name "Blanca Calderón"

git config --global user.email b.calderon@email.com



git config --list  #Nos permite ver todas las propiedades de
configuracion de Git.
```

Si no indico la opción - - *global* la configuración es solo para el repositorio local (Debo haber creado primero el repositorio local)

En el siguiente video-tutorial podemos ver como instalar y configurar GIT.

## Manejo de Git

Para entender completamente el funcionamiento de Git debemos leer los capitulos 2 y 3 del libro Pro Git Book. Podemos descargarlo desde su web oficial en inglés o leerlo online en español.

En los siguientes dos videos se muestra el uso de las principales operaciones que nos ofrece Git junto con el servidor de repositorio BitBucket.

**IMPORTANTE**: Al crear un repositorio remoto en bitbucket, indicar no crear el fichero README.md.

Git ofrece tanto integración en un terminal de Windows (cmd), como en un terminal de Linux (Bash). Los siguientes comandos están centrados en el manejo del terminal de Linux (bash):

| Comandos Bash | Función |
|---|---|
| ls | Muestra el contenido del directorio actual |
| ls -a | Muestra el contenido del directorio actual con los fichero ocultos |
| pwd | Muestra la ruta del directorio en el que estoy |

| Comandos Bash | Función |
|---|---|
| cd [directorio] | Me permita cambiar de directorio |
| directorios . y .. | Representan el directorio actual y su padre, respectivamente |
| rm [nombre_archivo] | Elimina un archivo |
| rm -r [nombre directorio] | Elimina un directorio con su contenido |
| touch [nombre_fichero] | Crea un fichero de texto plano con dicho nombre] |
| mkdir [nombre_directorio] | Crea un directorio vacío con dicho nombre] |
| clear | Limpia el terminal de comandos |

**Trabajo en repositorio local**

- Crear un repositorio local. Se ejecuta solo una vez, y con ello indico que en dicho directorio se llevará un control de versiones (se convierte en repositorio). Para ello crea un directorio oculto (.git) que contiene la información que Git necesita para registrar las versiones.

```
git init
```

- Estado actual del repositorio. Indica los cambios que ha habido en el directorio (repositorio) desde la última versión realizada. Muestra los nuevos ficheros añadidos al repositorio o fuera de seguimiento, los modificados y los eliminados, desde el último *comit*.

```
git status
```

- Incluir archivos o directorios para la siguiente versión. Permite añadir archivos que están fuera de seguimiento (untracked) o que han sido modificados (modified) y prepararlos para el siguiente commit.

```
git add [nombre_fichero_o_directorio]

# Ó si quiero añadir todo el contenido de mi repositorio local:

git add.
```

Mientras que no se confirmen (commit) las modificaciones, se puede restaurar el estado inicial con el comando *git reset*.

- Confirmar las modificaciones. Crea una nueva versión Con los archivos que están bajo seguimiento dentro de mi repositorio. Debemos indicar siempre un mensaje descriptivo.

```
git commit -m "Descripción del commit"
```

**Sincronización repositorio local - remoto**

- Sincronizar con un repositorio remoto. Añade las modificaciones que hay en mi repositorio local a un repositorio remoto, con el fin de tener los dos repositorios sincronizados (idénticos). Debemos indicar la dirección de nuestro repositorio remoto (url) o su alias, si le hemos asignado uno.

```
git push [dirección_repositorio_remoto] master



#Ejemplo

git push https://bitbucket.org/fvaldeon/repopruebadam1 master
```

- Actualizar nuestro repositorio local con los cambios más recientes que tiene uno remoto. Cuando nuestro repositorio remoto está más actualizado que nuestro mismo repositorio local, debido a que hemos sincronizado desde un repositorio local en otro equipo, debemos descargar los últimos cambios para incluirlos en nuestro repositorio local.

```
git pull [repositorio_remoto] master



#Ejemplo

git pull https://bitbucket.org/fvaldeon/repopruebadam1 master
```

Cuando hacemos *git pull* se descargan esos nuevos ficheros y se intentan unir con los ficheros de nuestro repositorio local. Desde el terminal de Git se abre el editor de texto de linux (Vi). Para salir de él pulsamos las teclas (:wq!).

- Obtener una copia de mi repositorio remoto. Usaré este comando cuando en mi ordenador no tengo una copia de mi repositorio. Uso la url de mi repositorio remoto para traerme una copia en local, y poder seguir trabajando. Esto genera un repositorio local idéntico al remoto.

```
git clone [repositorio_remoto]



#Ejemplo

git clone https://bitbucket.org/fvaldeon/repopruebadam1
```

Una vez que hayamos clonado un repositorio desde una url, dicha url se almacena como el alias **`origin`**. Desde ese momento puedo omitir la url, y usar el alias `origin` para hacer *push o pull*.

### Gestión de repositorios remotos

Dentro de cada repositorio local de Git, puedo asociar la dirección de mi repositorio remoto a un alias. Así no necesito recordar la url, sino el alias.

- Añadir un repositorio remoto la gestión de Git de mi respositorio local.

```
git remote add [alias] [dirección_remoto]
```

- Como norma general en Git, el repositorio remoto con el que tengo pensado trabajar principalmente se llama ***origin***:

```
git remote add origin [dirección_remoto]
```

- Mostrar los repositorios remotos vinculados a mi repositorio local.

```
git remote
```

- Además, puedo mostrar mis repositorios remotos y mostrar su dirección.

```
git remote -v
```

### Obtener versiones anteriores

Cuando queremos obtener una versión anterior de nuestro proyecto podemos consultar el log de nuestro repositorio local. Cada confirmación (commit) crea una versión. Al restaurar a una versión anterior, el estado (ficheros y directorios) de mi repositorio local se cambia por lo que había en el momento de crear dicha versión.

```
git log #Muestra un historial con las versiones

git log --oneline #Muestra un historial resumido
```

Obtenemos la siguiente salida, que representa todos los *commits* realizados en mi repositorio local y el mensaje que se incluyó.

```
git log --oneline

b8b6a2d (HEAD -> master, origin/master, origin/HEAD) Readme y gitignore
actualizados

e8ad0df LoginYFichConf y JListSuprimir subidos

0a46d10 Actualizado README

256d909 Proyecto VehiculosMVC terminado
```

```
ffa9848 Correccion en la organizacion de packages

3dee7c5 Proyecto terminado. No guarda ni carga en fichero.
```

Si nos fijamos en el código generado en cada confirmación (columna de la izquierda), podemos emplearlos para restaurar una versión anterior:

```
git checkout 3dee7c5
```

También podemos restaurar únicamente un archivo a una versión anterior:

```
git checkout 3dee7c5 prueba.txt
```

### Ignorar archivos en las versiones

Es bastante frecuente que en los proyectos que tengo en mi repositorio local tengo archivos o directorios que no quiero tener en seguimiento(versiones). Es el caso del directorio */bin* o de los ficheros *.class* de Java.

El fichero *.gitignore* es un fichero de texto que podemos crear en nuestro repositorio local y que indica qué ficheros se excluirán del seguimiento y sincronización con nuestro repositorio remoto.

.gitignore

```
# Ficheros generados Java

*.class



# Carpetas que contienen ficheros generados

bin/

out/
```

El fichero ***.gitignore*** debería ser sincronizado con el resto de ficheros del repositorio remoto.

### Borrar archivos de las versiones

- Borrar archivos ya confirmados. Si tengo actualmente ficheros en mis versiones que no quiero que sigan en mi proyecto, debo borrarlos de **git** y posteriormente

volver a confirmar. Debo tener en cuenta que dichos archivos también desaparecerán de mi repositorio local.

```
git rm [fichero]

git rm -r [directorio con archivos]



git commit -m "Archivos borrados"
```

- Eliminar archivos de la versión. Se usa cuando lo que quiero es únicamente eliminar algunos archivos de las versiones de git, pero quiero que permanezcan en mi repositorio local. Es normalmente útil cuando se me olvida incluir algo en mi fichero *.gitignore*

```
git rm -r --cached [fichero o directorio]
```

**Trabajar con ramas**

- Crear una nueva rama

```
git branch develop
```

- Cambiar de rama

```
git checkout develop
```

- Fusionar ramas

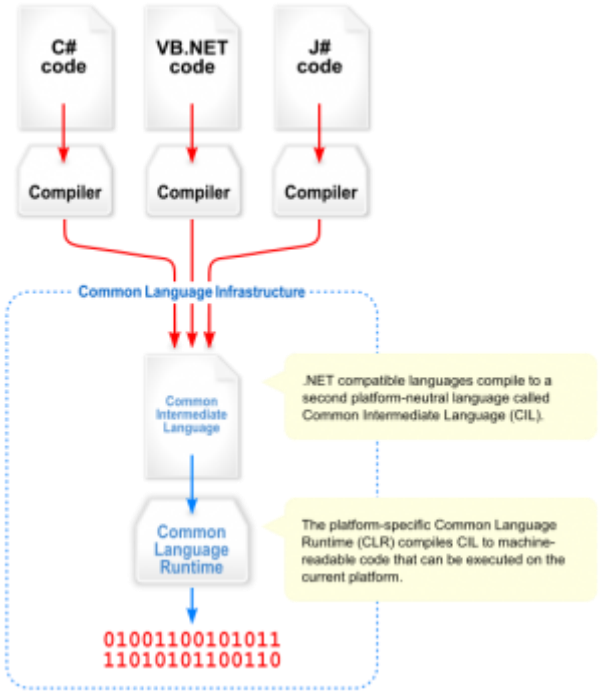
```
git checkout master

git merge develop
```

Con el siguiente tutorial interactivo podemos practicas todos los conceptos vistos aquí y también el manejo con ramas: https://learngitbranching.js.org/

# Visual Studio IDE

Es un IDE creado por Microsoft y usado en plataformas Windows. Tiene licencia propietaria y soporta distintos lenguajes como C#, C++, Visual Basic .NET, Python, etc. Este IDE está preparado para trabajar con todas las tecnologías y lenguajes de la plataforma .NET de Microsoft.

## Plataforma .NET




.NET es un framework creado por Microsoft principalmente para sistemas Windows. De forma parecida a la *plataforma Java*, .NET ofrece librerías de clases y un entorno de ejecución para diferentes lenguajes. A esta plataforma se le conoce como *CLI* (Common Language Infraestructure). Está formada de 2 partes:

- Todos los programas escritos en el *framework* .NET se compilan a un lenguaje intermedio conocido como *CIL* (Common Intermediate Language)
- Posteriormente se ejecutan en un entorno de ejecución llamado *CLR* (Common Language Runtime), parecido a la máquina virtual de Java.

Esto hace que las aplicaciones .NET puedan ser independientes de la máquina, a través de un enfoque similar al que usa Java.

## Lenguaje C#

Es un lenguaje de programación de propósito general, orientado a objetos y multiparadigma. Aunque ha sido muy desarrollado por Microsoft para su plataforma .NET, empleado como un Lenguaje de Infraestructura Común (CLI), C# es un lenguaje independiente que trabaja para otras plataformas.

La sintaxis del lenguaje C# está basada en la sintaxis de C/C++, de la misma forma que Java, por lo que la mayor parte de sus instrucciones se escriben y organizan de la misma forma. .NET también actúa como API para el lenguaje C#. Podemos acceder a la documentación de la biblioteca de clases de .NET

Programa `Hola Mundo` en lenguaje C#:

```
class Saludo {

  static void Main() {

    System.Console.WriteLine("Hola Mundo!");



    System.Console.WriteLine("Pulsa una tecla para terminar");

    System.Console.ReadKey();

  }

}
```

Las instrucciones principales que necesitaremos para crear programas de consola son:

- Mostrar mensajes por consola

```
System.Console.Write("Mensaje"); //Escribe por consola, sin salto de
linea

System.Console.WriteLine("Mensaje"); //Escribe por consola, con salto de
linea
```

- Leer texto por consola

En C# se leen los datos siempre como String y posteriormente se convierten al tipo necesario. **C# no permite leer int directamente.**

```
System.Console.ReadLine(); //Lee la siguiente linea (String)

System.Console.ReadKey(); //Lee una tecla pulsada en código unicode.
```

- Convertir de String a otro tipo y viceversa

```
int numero = int.Parse("-466");  //Convierto de String a int

double numeroReal = double.Parse("32.345"); //Convierto de String a
double
```

```
String cadena1 = numero.ToString(); //Convierto de int a String

String cadena2 = numeroReal.ToString(); //Convierto de int a String
```

- Espacio de nombres y sentencia *using*

Las clases en C# se organizan en espacios de nombres (namespaces). Un conjunto de clases que tengan relación entre sí, se agruparán bajo un mismo espacio de nombres. La idea es similar a los paquetes (packages) en Java. Ejemplos de espacios de nombres:

- System → contiene clases como Console, String, y otros tipos de datos muy comunes.
- System.IO → Contiene clases para entrada y salida, manejo de ficheros, etc.
- System.Windows.Forms → Contiene clases para crear interfaces gráficas de usuarios.

```
namespace Aplicacion  //espacio de nombres 'Aplicacion'

{

    public class MiClase{   //Puedo acceder a esta clase indicando
'Aplicacion.miClase'



    }

}
```

Por otra parte, podemos usar la sentencia *using* al comienzo de una clase para indicar a nuestro compilador los espacios de nombres de las clases que vamos a usar (p.e. clase Console).

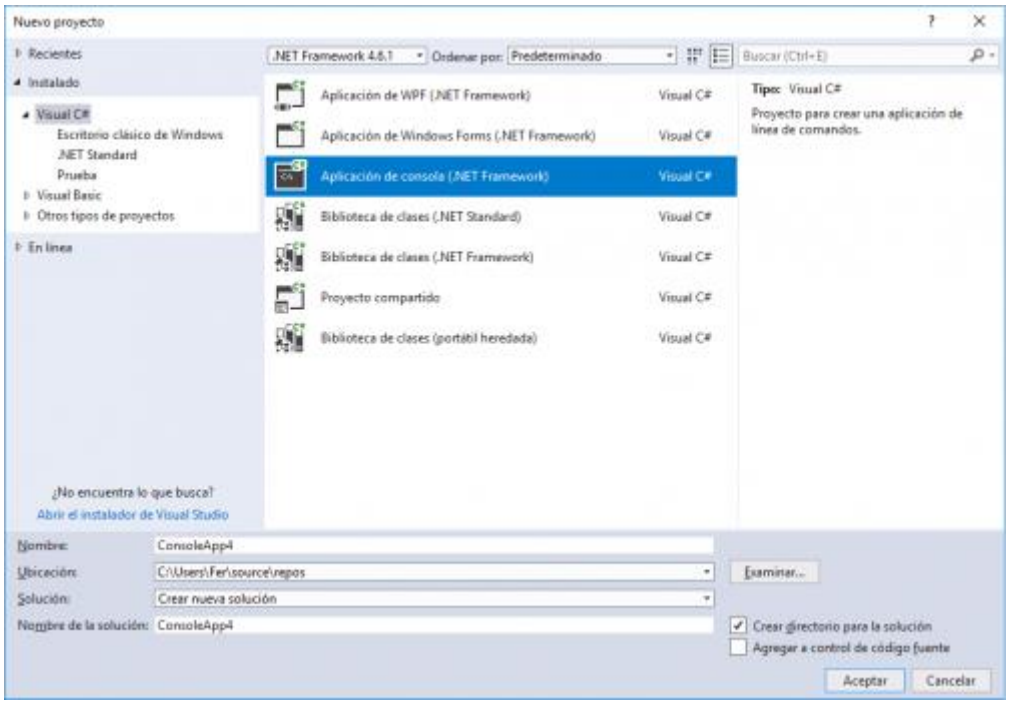
De esta forma podemos acceder a los métodos de la clase *Console* directamente sin necesidad de indicar el espacio de nombres (System):

```
using System;    //espacios de nombres que estoy usando

using System.IO;

namespace Aplicacion.parte1 //espacio de nombres de 'miClase'

{

    public class miClase{

       static void main(string[] args)
```

```
        {

            Console.WriteLine("No necesito escribir System para usar la
clase Console");

            Console.WriteLine("Pulse una tecla para terminar...");

            Console.ReadKey();

        }

    }

}
```

**Crear una aplicación de Consola**

Debemos crear un proyecto en C# seleccionando *Aplicación de Consola (.Net Framework)*. Para desarrollar en lenguaje C# necesitamos añadir el componente **Desarrollo de escritorio .NET**.




## Depurador

Para utilizar la herramienta de depuración debo crear antes algún *breakpoint* pulsando doble clic sobre la barra vertical a la izquierda del número de línea. La perspectiva del modo depuración se iniciará en el momento en que mi programa alcance algún breakpoint.

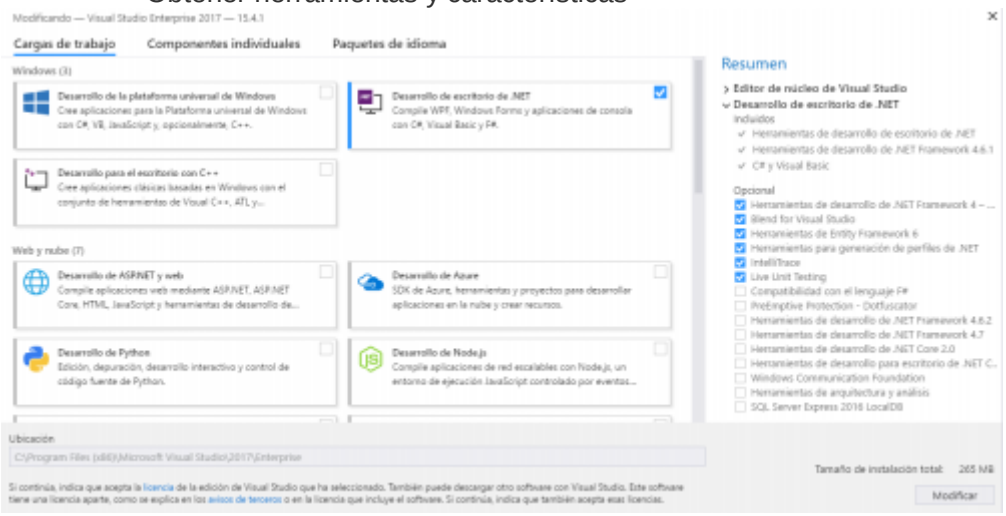
Para iniciar el modo depuración debo ir al menú *Depurar* y pulsar sobre *Iniciar Depuración* o también pulsando la tecla F5.

| Comando | Atajo de Teclado | Descripción |
|---|---|---|
| **Paso a paso por instrucciones** | **F11** | Si la línea contiene una llamada a un método, entra en el método y se detiene en la primera línea de código incluida en él. |
| **Paso a paso por procedimientos** | **F10** | Si la línea contiene la llamada a un método, no entra en el método y continua en la siguiente instrucción. |
| **Paso a paso pasa salir** | **Shift+F11** | Si estamos depurando un método, sale del método y continua en la siguiente instrucción. |

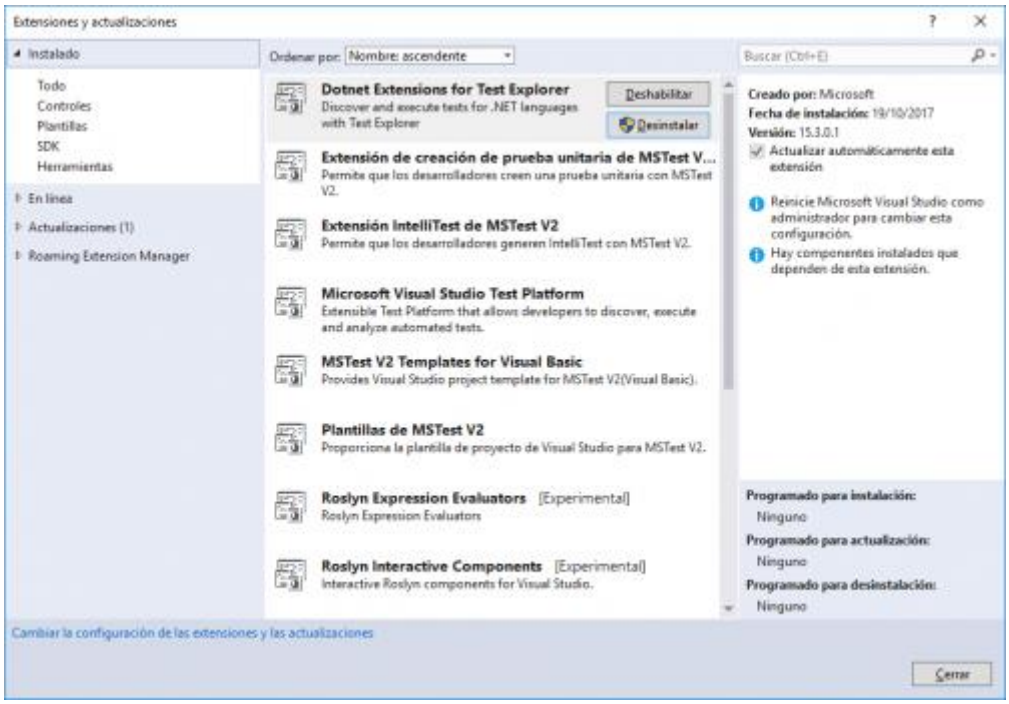
## Añadir componentes

Puedo acceder a la herramienta para incorporar extensiones o módulos, desde el menú *Herramientas* con las opciones *Obtener herramientas y características* o con *Extensiones y actualizaciones*.

- Obtener herramientas y características



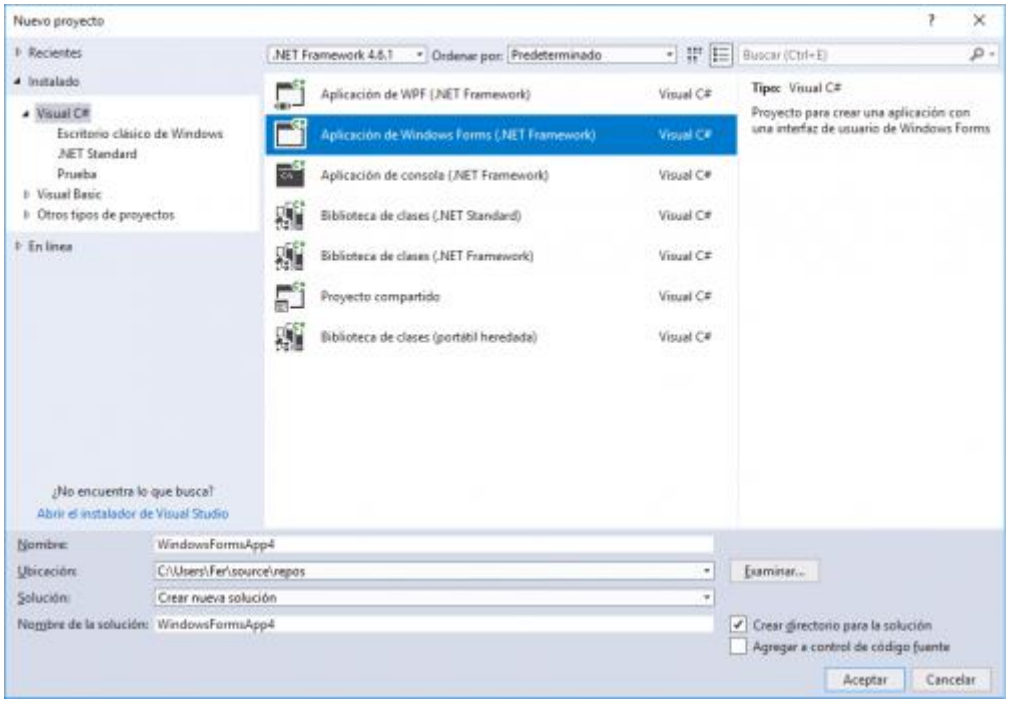

- Extensiones y actualizaciones



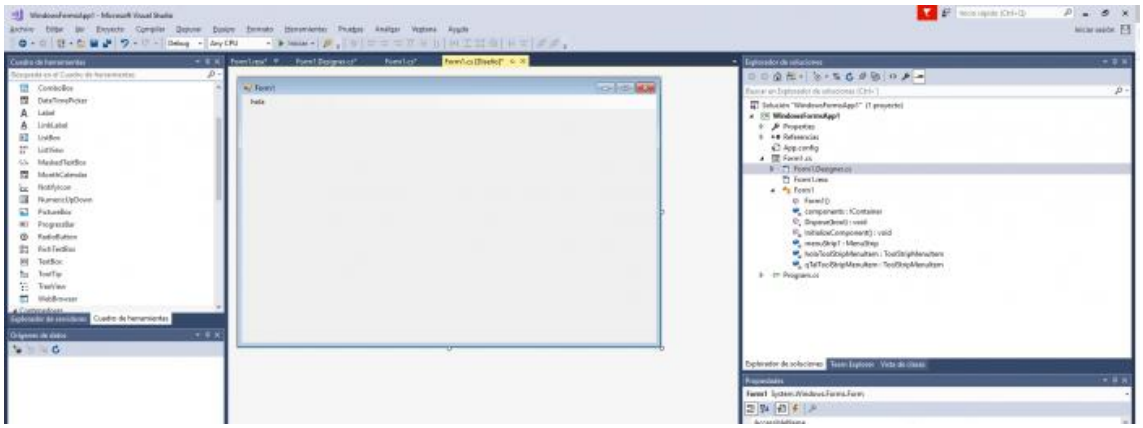

## Interfaz gráfica de usuario

Lo primero que necesitamos es añadir el componente de “Desarrollo de Escritorio .NET” desde la sección de componentes y extensiones.

Para crear una aplicación que ofrezca una interfaz gráfica debemos crear un proyecto de **Aplicación Windows Forms**.

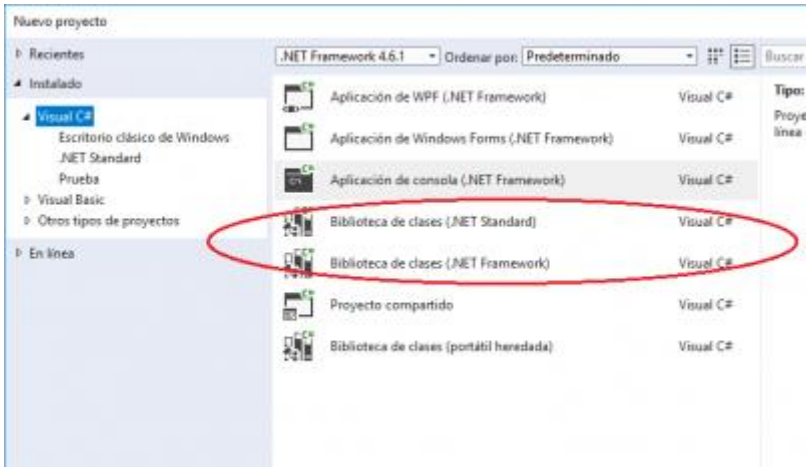


Esto nos creará una clase en C# vinculada a un formulario que nos permite diseñar una ventana. Nos ofrece una paleta con componentes gráficos que podemos arrastrar a un formulario para diseñar nuestra interfaz de usuario (*GUI*).



## Librerías de clases

En la plataforma .NET uno de los tipos de ficheros que contienen librerías de clases se conocen como **ficheros .dll** (librerías de enlace dinámico). Para poder crear un proyecto de librerías de clases debo seleccionar un proyecto de Librerías de Clases.




Después en él crearemos las clases que necesitemos con los métodos que queramos. Y finalmente compilaremos la solución. De este modo en el diretorio ***bin/Debug*** de la carpeta del proyecto tendré el fichero *.dll* con las librerías que he creado.

Para incluirlas en otro proyecto, en el *Explorador de Soluciones* debo pulsar con el botón derecho sobre el proyecto en el que las quiero incluir. Después **Agregar** → **Referencia** y ahí buscaré la *.dll* dentro de mi equipo y lo incorporaré al proyecto.

**Todas las clases y los métodos definidos en una librería deben ser públicos (public) si quiero tener acceso a ellos.**

# Eclipse


Es un entorno de desarrollo integrado de código abierto desarrollado en el lenguaje Java. Ofrece soporte para múltiples lenguajes.

## Workspace / Proyecto

- Un **workspace** es un tipo de directorio que usa Eclipse para almacenar proyectos. Un workspace no es un proyecto. Podemos organizar los distintos proyectos en diferentes workspaces. Para cambiar de workspace iré a la pestaña **File → Switch workspace**. Por defecto, Eclipse abre el último workspace utilizado.

Es importante entender que todos los proyectos que tengo en un workspace no tienen por qué aparecer en el *Explorador de Paquetes* de Eclipse. Pueden estar en el workspace y no haberlos abierto o importado en Eclipse.

- Un **proyecto** es un directorio que contiene subdirectorios para organizar todos los elementos de un programa. Principalmente tenemos la carpeta **src** donde guardamos los ficheros fuente .java, la carpeta **bin** donde se guardan los ficheros .class generados de la compilación de los fuentes. Para abrir un proyecto puedo hacerlo desde la pestaña **File → Open Projects…** o también desde **File → import → General → Existing Projects into workspace**.

Es importante entender que para abrir un proyecto no es necesario tenerlo en ningún workspace. Puede estar en cualquier lugar de nuestro sistema, y ahí seguirá. Sin embargo, es útil almacenarlo en algún workspace.

## Plantillas (Templates)

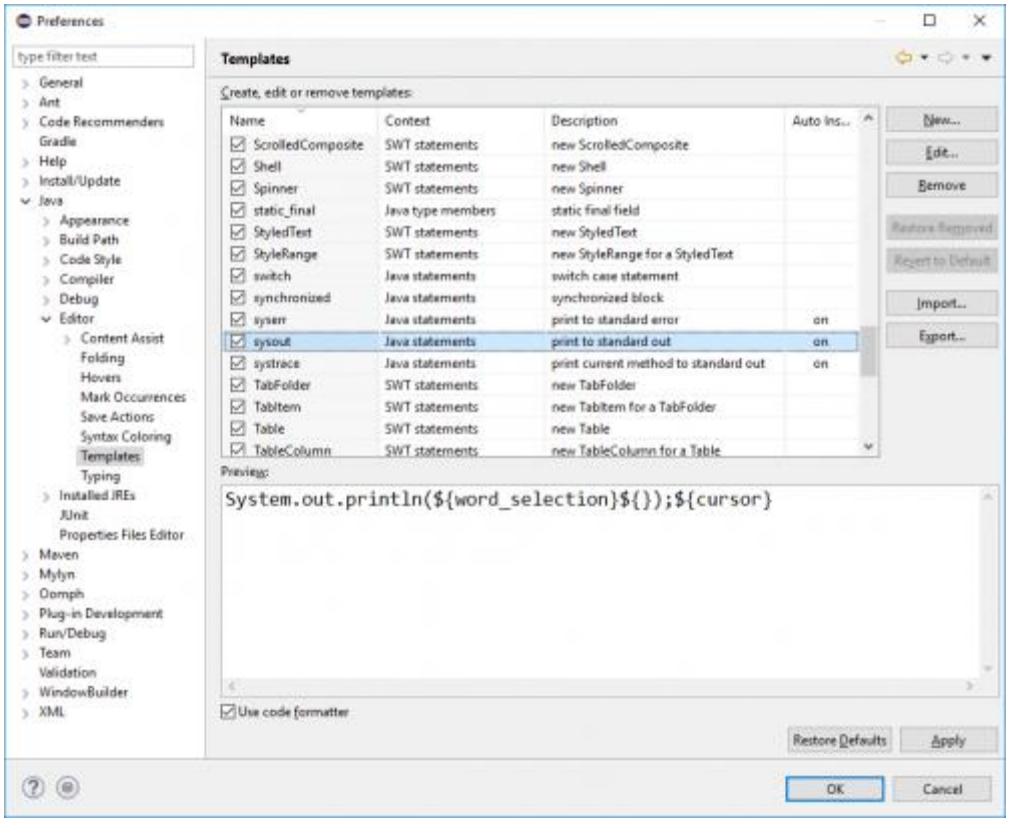
Las plantillas son un tipo de herramientas mediante las cuales se puede autogenerar (autocompletar) código a partir de una palabra. Se despliegan en un menú al pulsar la combinación de teclas *ctrl + barra espaciadora*.

```
syso -> System.out.println()
```

Para ver las plantillas que tenemos predefinidas para un lenguaje iremos a Window → Preferences.

Ahí accederemos al lenguaje del cual queramos mostrar las plantillas o gestionarlas.

Para Java: Java → Editor → Templates




Desde la opción ***New*** podremos crear nuestra propia plantilla. Esta se compone de:

- Nombre: nombre que aparecerá en mi editor cuando la invoque
- Descripción: Para qué se usa
- Contexto: Podemos indicar el lenguaje para el que se usa
- Patrón: código de la plantilla. Puedo además usar variables de Eclipse

Podemos consultar algunos aspectos de la creación de plantillas desde la documentación de eclipse, en concreto sobre las variables que podemos usar al crear plantillas.

Ejemplo de plantilla para Scanner:

## Depurador

Para establecer un punto de ruptura en mi código (breakpoint) tan solo debo hacer doble clic en mi editor de código de Eclipse, en la pequeña columna vertical que tenemos a la izquierda del número de línea. Cuando creamos un breakpoint queda marcado un pequeño punto azul. Se eliminan de la misma forma. Para acceder al modo depuración: Menu *Run → Debug*, o mediante la tecla *F11*. Mi programa se ejecutará hasta que encuentre un *breakpoint*, momento en el que parará.

- El primer botón (F8), botón “play”, hace que continúe la ejecución de nuestro programa hasta el siguiente Breakpoint.
- El siguiente botón, parecido al botón de pausa, suspende la ejecución del programa que está corriendo. Es una forma de parar “al vuelo” el programa, esté donde esté en ese momento, haya o no breakpoint.
- El cuadrado rojo es el botón de stop, aborta la ejecución de nuestro programa.

**Botones de avance paso a paso:**

- El siguiente botón (F5), una flecha amarilla que se “mete” entre dos puntos, hace que el debugger vaya a la siguiente instrucción y si encuentra un método, se meterá dentro del método y se parará en la primera línea del mismo, de forma que podemos depurar dentro del método.
- El siguiente botón (F6), una flecha amarilla que salta sobre un punto negro, avanza un paso la ejecución del programa, y si se encuentra un método no entra dentro de él, sino que salta a la siguiente instrucción.
- El último botón (F7) es una flecha saliendo de entre dos puntos negros, cuando nos encontramos dentro de un método, salta hasta la instrucción en la que ha sido llamado el método.

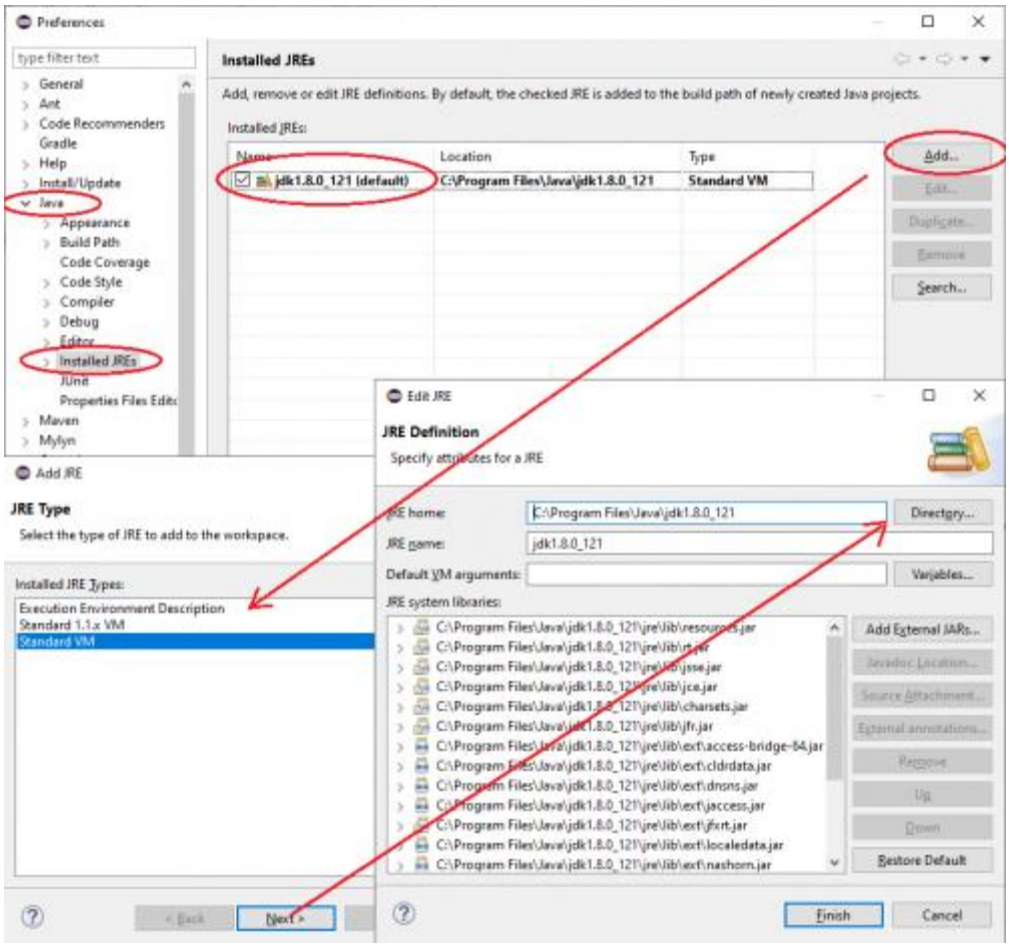
Además, podemos ver el valor que devuelve una instrucción, seleccionado la expresión (método) y ejecutando el atajo del teclado **Ctrl + Shift + i**. Por ejemplo, si no sé qué puede devolver la instrucción *cadena.charAt(i)*, selecciono dicha instrucción y pulso ctrl+shift+i.

**Depurar en las clases internas de Java**

Una posibilidad muy interesante del depurador es Eclipse, es poder acceder a los métodos de las clases nativas de Java. Cuando depuramos nuestro código fuente podremos ver el avance de la ejecución de nuestro programa, pero no podremos ver el avance dentro de los métodos de las clases propias de Java (Clase String, Scanner, etc).

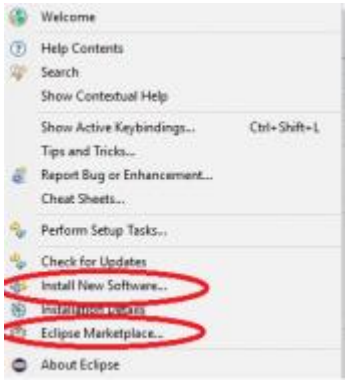
Para que nuestro IDE tenga acceso al código fuente de estas clases necesito indicarle dónde está mi paquete JDK de Java: **Window → Preferences → Java → Installed JREs**

Por defecto tendré el paquete **JRE** (Java runtime Environment) en lugar del paquete de desarrollo **JDK**. Debo seleccionar el paquete JRE y pulsar sobre *Remove* para eliminarlo. A continuación, añadiré el JDK, tal y como se muestra en la siguiente imagen:




Una vez que accedo a la ventana *JRE Definition*, debo buscar la carpeta de mi JDK de Java en mi equipo y seleccionarla. Normalmente estará en C:/Archivos de Programa/Java/Jdk… Acepto los cambios y podré depurar con esta funcionalidad.

## Añadir componentes




Desde la pestaña *Help* en Eclipse podemos incorporar elementos a nuestro IDE. Podemos descargarlo desde un servidor repositorio externo desde la opción *Install New Software* o a través del *Eclipse MarketPlace*. Desde este último simplemente tenemos que buscar las etensiones que queremos incluir y pulsar sobre el botón instalar.

## Interfaz gráfica de usuario

El plugin WindowBuilder nos permite crear ventanas a través de editores de ventanas *drag and drop*. Desde el menu *Nuevo* puedo seleccionar otros tipos de aplicaciones, → WindowBuilder → SwingDesigner → Jframe, por ejemplo. Esto crea una nueva perspectiva en la que puedo diseñar una ventana con un editor, y ver al mismo tiempo como se crea el código Java referente a mi ventana.

## Librerías de clases

### Ficheros JAR

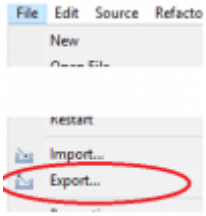
Jar son las siglas de **J**ava **AR**chive, y es un tipo de fichero en el que podemos empaquetar clases de Java. Son archivos comprimidos en formato **.zip** y cambiada su extensión a *.jar*. En el directorio *bin* del JDK de Java tenemos el programa *jar* para crear estos ficheros, aunque también podemos crearlos desde el IDE con el que trabajemos.

En Java el fichero estandar para empaquetar librerías de clases es el fichero ***Jar***. Podemos crear nuestras propias clases con métodos para usarlas en otro proyecto, empaquetándolas en un fichero jar. Así no tenemos que llevar nuestras carpetas y archivos sueltos de un lugar a otro.

Dentro de un fichero JAR podemos empaquetar todos los recursos que tiene nuestro proyecto: ficheros .class, otras librerías, imágenes, y todo lo que esté en los directorios de nuestro proyecto.

### Fichero Jar con Clases (Librería de clases)

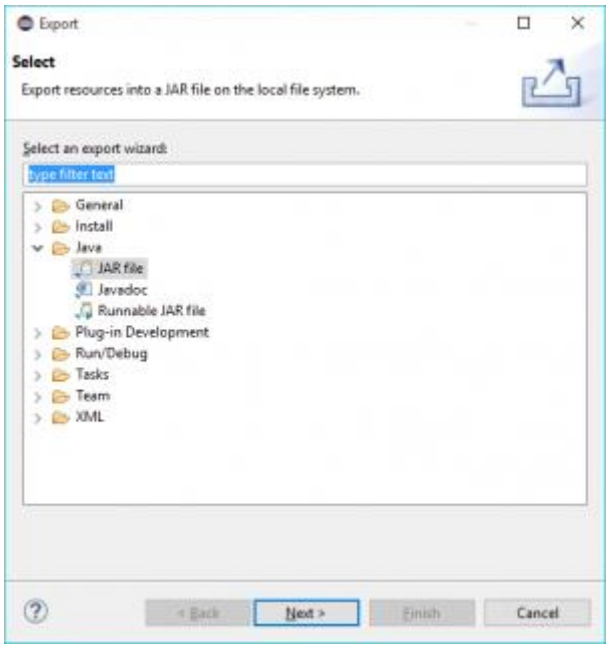
*Todas las clases y los métodos que guardemos en una librería deben ser definidos como públicos* ***(public)***

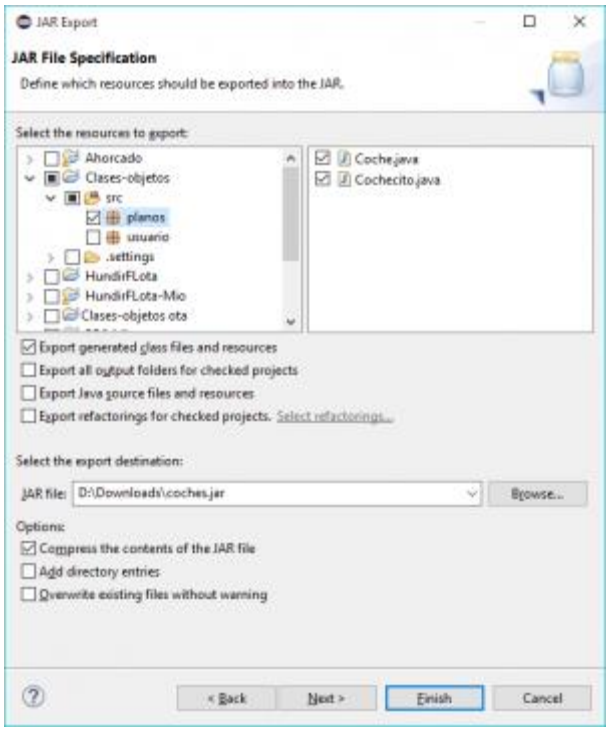



Para guardar en un fichero *jar* las clases que necesitamos usar en otros proyectos, debemos seleccionarlas para empaquetarlas. En Eclipse desde la pestaña *File*, seleccionaremos la opción *Export*.

Una vez en esta nueva ventana abriremos la sección *Java*, y pulsamos sobre *JAR File*.

Después de pulsar sobre *JAR file*, se abrirá otra ventana en la que podemos seleccionar las clases que queremos guardar en nuestro JAR. Podemos seleccionar clases de distintos proyectos que queramos usar en otro.



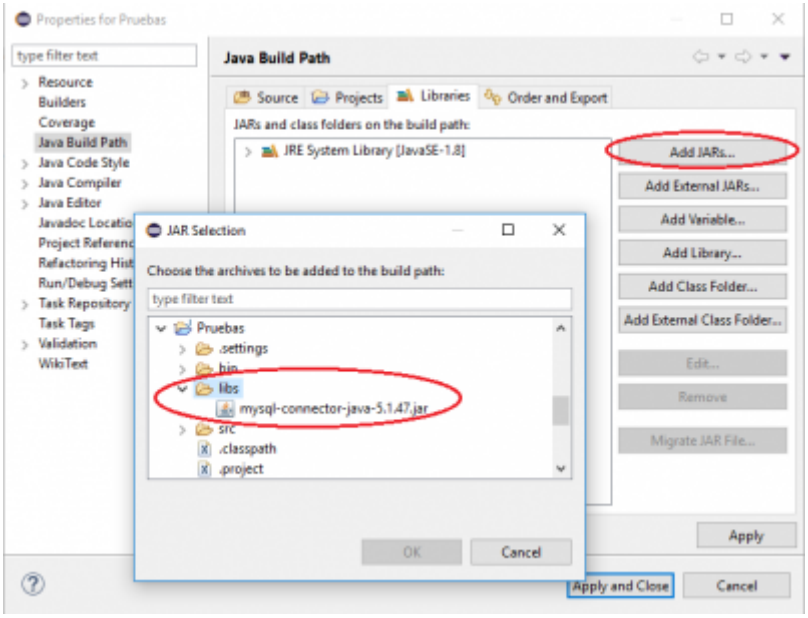

---

Finalmente seleccionaremos un destino para almacenar nuestro archivo JAR. Podemos usar una carpeta nueva (libs, por ejemplo) en el workspace para almacenarlos.

**Añadir librerias JAR a otros proyectos**: Para incluir ficheros JAR con clases y poderlas usar en otros proyectos debemos incluirlas en el PATH del proyecto:

1. Creamos un directorio en nuestro proyecto: por ejemplo, *libs*

2. Añadimos a dicho directorio nuestra libreria *.jar* (Copiamos el fichero Jar de la ubicación y lo pegamos en ese directorio)
3. Pulsamos con el botón derecho sobre nuestro proyecto en el explorador de paquetes, y seleccionamos *Build Path → Configure Build Path*
4. En la nueva ventana iremos a la pestaña *Libraries → Add Jars…* y seleccionaremos la librería Jar.




Ahora podemos usar las clases que tenemos en el paquete JAR.

# Generación de Ejecutables

## Ficheros Exe

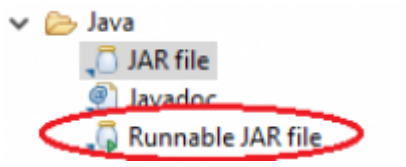
Desde **Visual Studio** a través de la *Infraestructura de Lenguaje Común* los ejecutables que obtenemos después de construir el proyecto tiene extensión *.exe* y permiten ser ejecutados en cualquier máquina que tenga el framework .NET instalado.

Los podemos encontrar en la carpeta `bin` dentro del directorio del proyecto de Visual Studio.

## Ficheros Jar Ejecutables

Son contenedores de clases, en los que, a la hora de crearlos, indicamos la clase que contiene el método *main*

En Eclipse: **File → Export → Java → Runnable JAR File**



- **Fichero Jar Ejecutable con programa consola**: Si lo que quiero es obtener mi programa ejecutable en un fichero JAR, debo crear un *Runnable JAR file*, indicando el proyecto y su clase principal (clase con método Main). Guardaremos el fichero Jar en nuestro PC.

Posteriormente para ejecutar ese programa JAR arrancaré el terminal de Windows (cmd), iré al directorio donde lo tengo almacenado y ejecutaré: *java -jar archivoJar.jar* (donde *archivoJar.jar* es mi archivo). También puedo ejecutarlo desde cualquier lugar, dando la ruta de mi fichero jar.

- **Fichero Jar Ejecutable con Interfaz gráfica**: Para guardar mi aplicación con interfaz de usuario en un JAR ejecutable, después de pulsar sobre exportar para crear un fichero JAR, debo indicarle que el tipo de JAR será *Runnable JAR file*. Este tipo de ejecutables, pueden lanzarse directamente, sin un terminal de comandos.

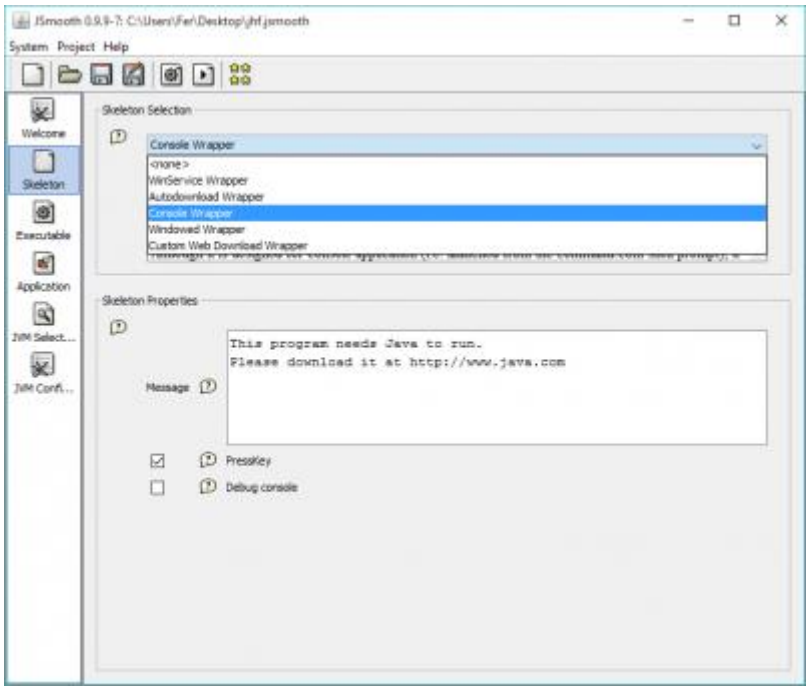
## Exe Java con JSmooth

Cuando tenemos un programa en Java, podemos crear un ejecutable (Exe) en consola a partir de él. Tan solo necesitaremos haber exportado nuestro programa como un archivo Jar.

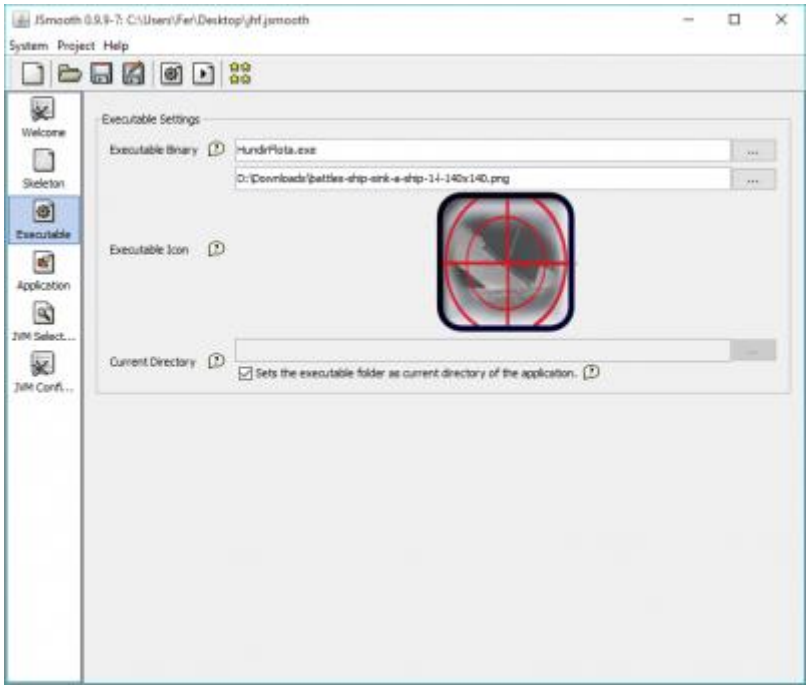
**JSmooth** es un programa muy sencillo de usar y con poca configuración, aunque solo está disponible para Windows. Actualmente el proyecto ya no tiene continuidad, pero el programa sigue funcionando, es gratuito, y descargable desde su página web.

***Problema al iniciar JSmooth - JSmooth no reconoce Java***: Al instalar Java hay algunas variables de entorno que no se crean de forma adecuada. Para usar JSmooth necesitamos crear una nueva variable de sistema llamada *JAVA_HOME* (Equipo → Propiedades → Configuración Avanzada → Opciones Avanzadas → Variables de entorno) y darle el valor de la ruta del JDK que estamos utilizando (p.e. → C:\Program Files\Java\jdk1.X.X.X).

- **Sección Skeleton**: Aquí indicamos simplemente de qué tipo de aplicación se trata, si de consola (Console Wrapper) o de Ventana (Windowed Wrapper). En esta sección configuramos qué mensaje aparecerá en caso de que el usuario no tenga instalada la máquina virtual de Java. La opción **presskey** es útil en aplicaciones de consola, para que no se cierre la ventana del terminal automáticamente.

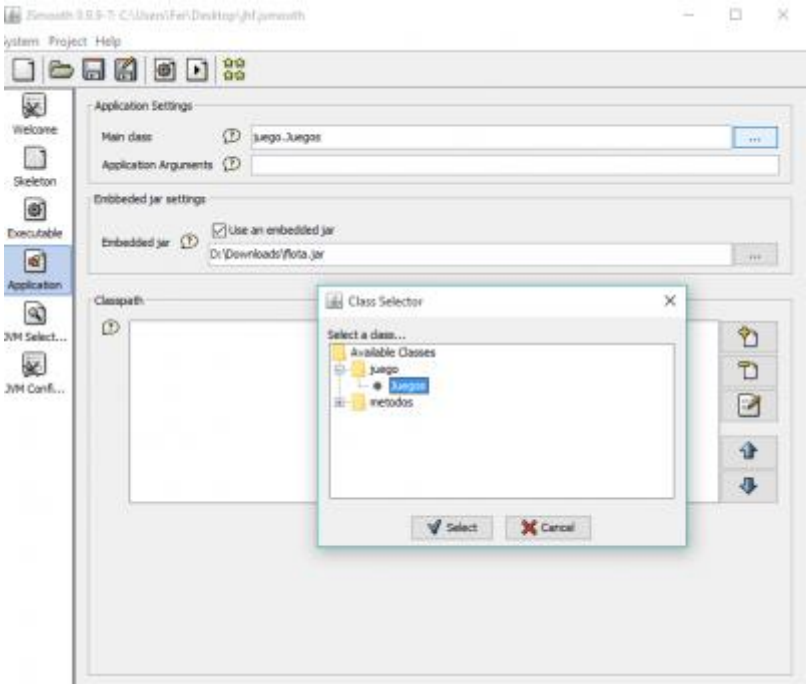



- **Sección Executable**: Aquí indicaremos simplemente el nombre y el lugar donde creamos el fichero ejecutable (Executable Binary). También podemos seleccionar una imagen, para el icono de nuestro ejecutable. Debe ser una imagen (p.e. PNG) pequeña, 120×120 o similar. En "Current Directory" indicamos simplemente el directorio por defecto que usará la aplicación cuando la ejecutemos. Podemos marcar el Checkbox ya que en principio no nos importa demasiado.




- **Sección Application**: En esta sección debemos indicar dónde está nuestro archivo Jar en nuestro ordenador (Embbeded Jar Settings) marcando el checkbox. Después en "Application Settings" pulsaremos sobre Main Class para

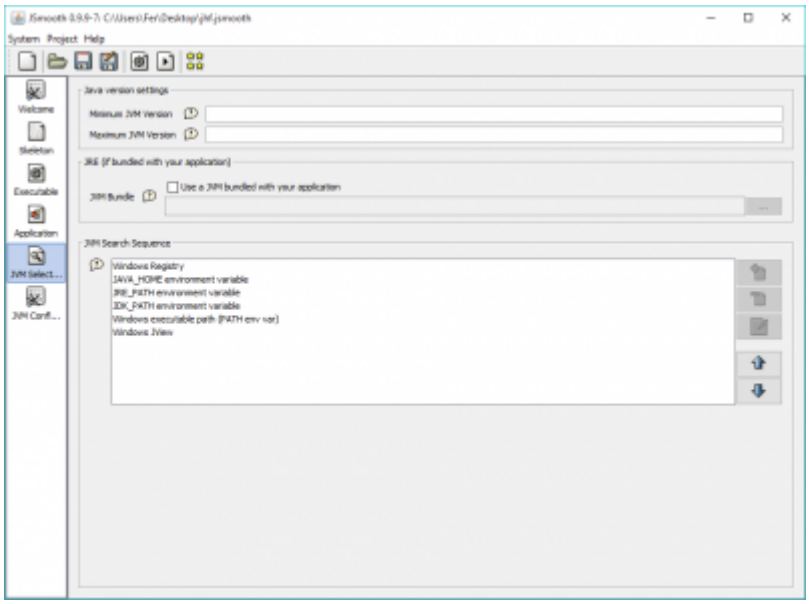
indicarle cuál es la clase de nuestro JAR que contiene el método ***main***, desde la que se iniciará el programa. Y está!!




- **Crear nuestro ejecutable**: Pulsaremos sobre la pestaña “Project” y después sobre “Compile”. Nos pedirá de nuevo una ruta y un nombre para guardar un archivo de proyecto de JSmooth (no es el exe). También nos creará un fichero .exe en la ruta que le indicamos anteriormente. Ya podemos ejecutar nuestro programa.

Además, también podemos configurar nuestra aplicación a partir de estas secciones:

- **Sección JVM Selection**: Podemos indicarle la versión de Java mínima que necesitamos para ejecutar la aplicación. Esto se debe a que hay instrucciones de Java que han salido en una versión concreta. En principio no necesitamos indicar nada, a no ser que sepamos que nuestro programa no funciona en todas las versiones de Java. (Por ejemplo, Scanner está en java desde 1.5v, o “Java 5”). Actualmente usamos Java 8 (1.8v)



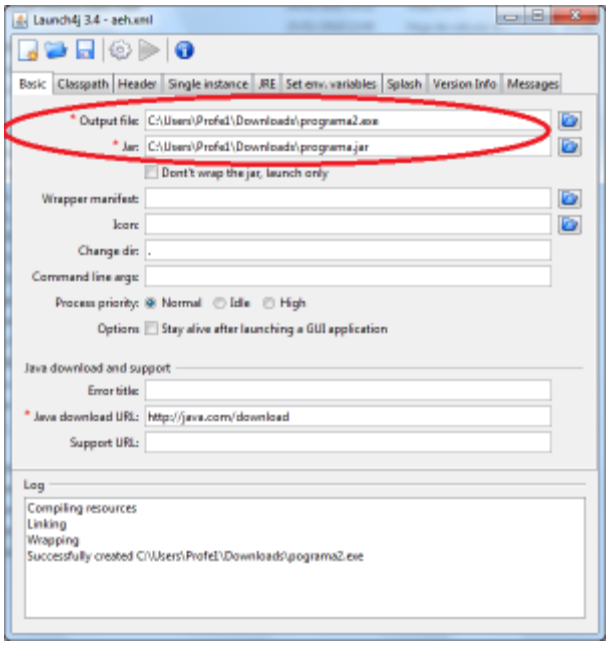
- **Sección JVM Configuration**: Aquí podemos indicar la cantidad de memoria que usará nuestra aplicación de Java o la versión mínima que necesitamos.

## Exe Java con Launch4J

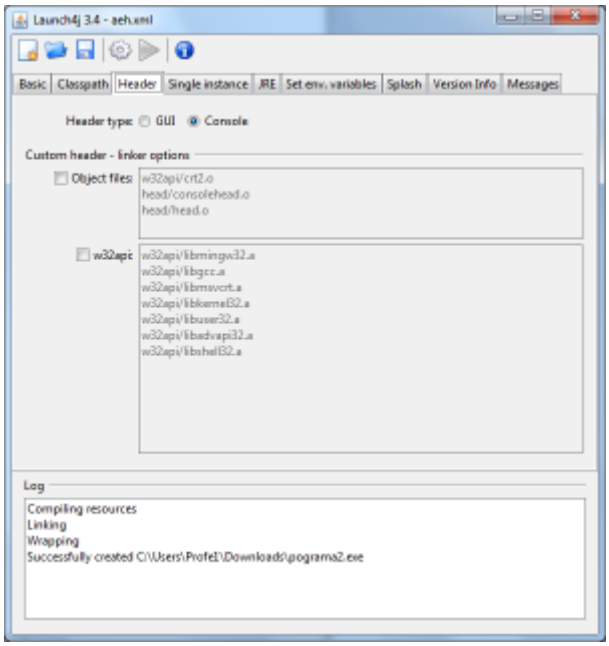
**Launch4J** es un programa similar a JSmooth, aunque permite más detalle en la configuración. El proyecto de desarrollo de Launch4J sigue en activo y es mutiplataforma (Linux, Windows, Mac). Lo podemos descargar desde su página web.

Para crear un .exe a partir de un JAR con Launch4J necesito indicar algunos parámetros en las siguientes secciones de la aplicación:

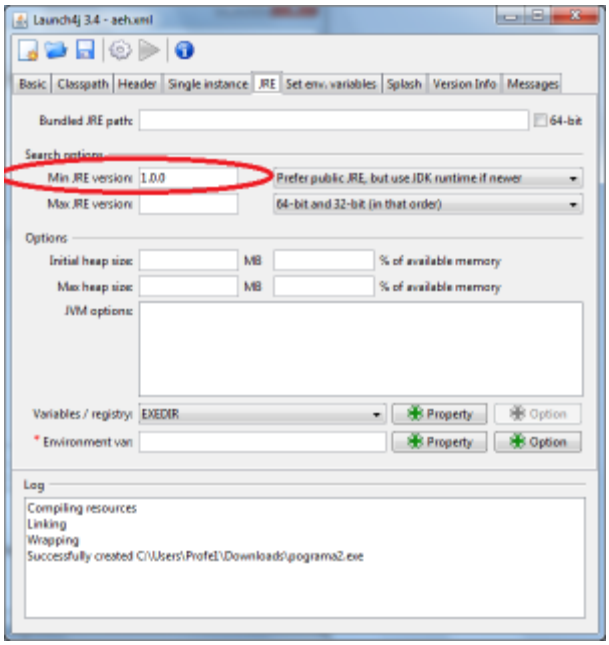
- **Sección Basic**: Debemos indicar el nombre y la ruta del fichero .exe que vamos a crear en el campo *OutPut File*. También indicaremos donde se encuentra el fichero Jar que vamos a utilizar para crear el exe, en el campo *Jar*. Si queremos que la aplicación tenga un icono lo indicaremos en el campo *Icon*.




- **Sección Header**: Debemos indicar si la aplicación es de Ventana o de Consola:




- **Sección JRE**: Debemos indicar al menos una versión mínima de Java necesaria para ejecutar el fichero. También podemos configurar los parámetros de Java con lo que quiero ejecutar mi aplicación.




- **Crear el ejecutable .exe**: Una vez configuradas al menos las partes anteriores, pulsamos sobre el botón del engranaje en la barra de herramientas. Nos preguntará donde queremos guardar el archivo del proyecto de Launch4J (es distinto al fichero .exe que queremos generar)

Además, también podemos configurar los siguientes aspectos de la aplicación:

- **Splash**: puedo indicar una imagen de inicio antes de arrancar la aplicación.
- **Version Info**: información sobre el fichero ejecutable.
- **Messages**: Configuro los mensajes que mostraría mi ejecutable ante posibles errores.

---