EFA
MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

UT2 – INSTALACIÓN Y USO DE ENTORNOS DE DESARROLLO

```
 *
 * @author Piotr Gieldon
 *
 */
public class EpicEditorTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

ENTORNOS DE DESARROLLO

DANIEL GONZÁLEZ-CALERO JIMÉNEZ

eclipse

NetBeans

IntelliJIDEA

EFA
MORATALAZ

1º CFGS Desarrollo
Aplicaciones Web

ENTORNOS DE DESARROLLO

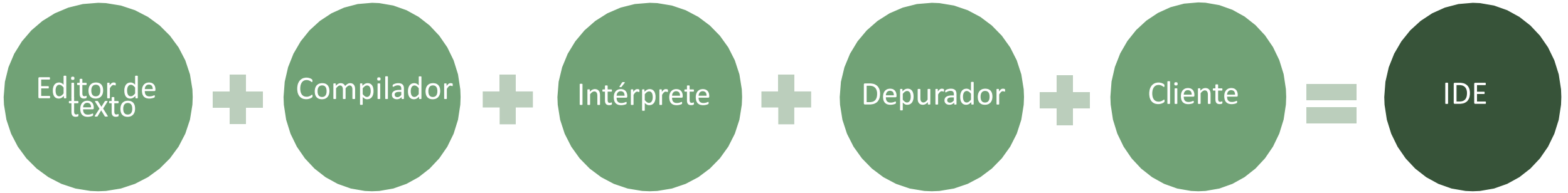
INDICE

UT2 – INSTALACIÓN Y USO DE ENTORNOS DE DESARROLLO

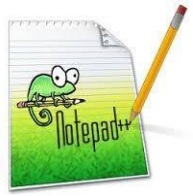
CARACTERÍSTICAS DE UN IDE

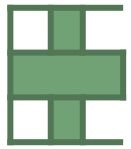
1

- Entorno de desarrollo integrado (*IDE*)
  - Es un programa informático que tiene el objetivo de asistir al programador en la tarea de diseñar y codificar un software mediante la inclusión de múltiples herramientas destinadas para dicha tarea
- Características comunes en un IDE



- Es posible programar sin IDE utilizando:
  - Editor de textos
  - Compilador





- Pero perdemos muchas facilidades y herramientas que nos facilita el IDE

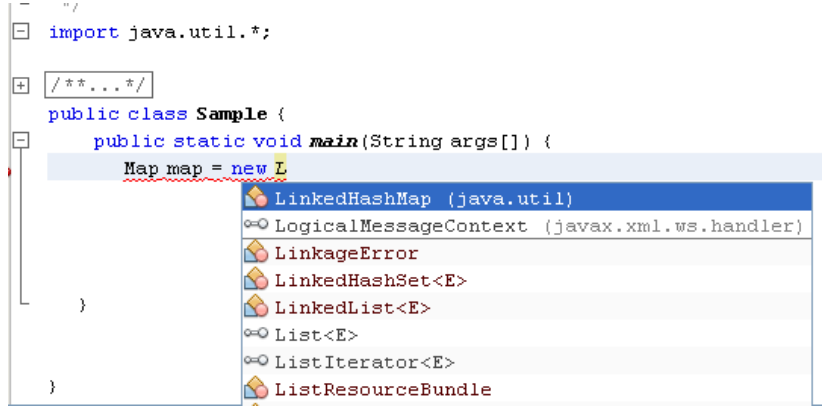
Con coloreado de sintaxis

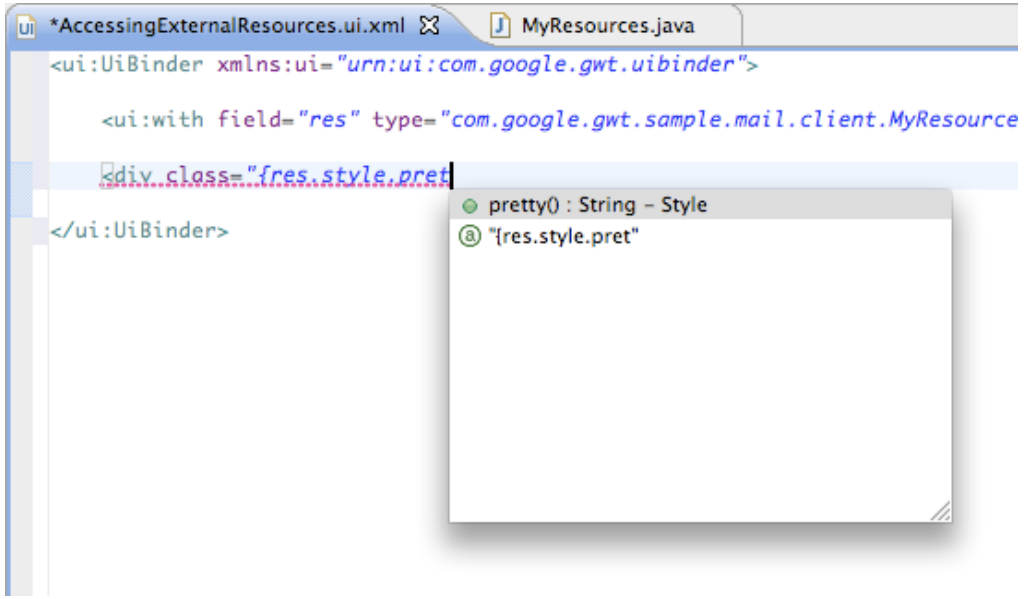
```java
public class Prueba{
    public static void main(String [] args){

        int a = 3;
        int b = 5;

        int c = a + b;

        System.out.println("a + b = " + c);
    }
}
```

Sin coloreado de sintaxis

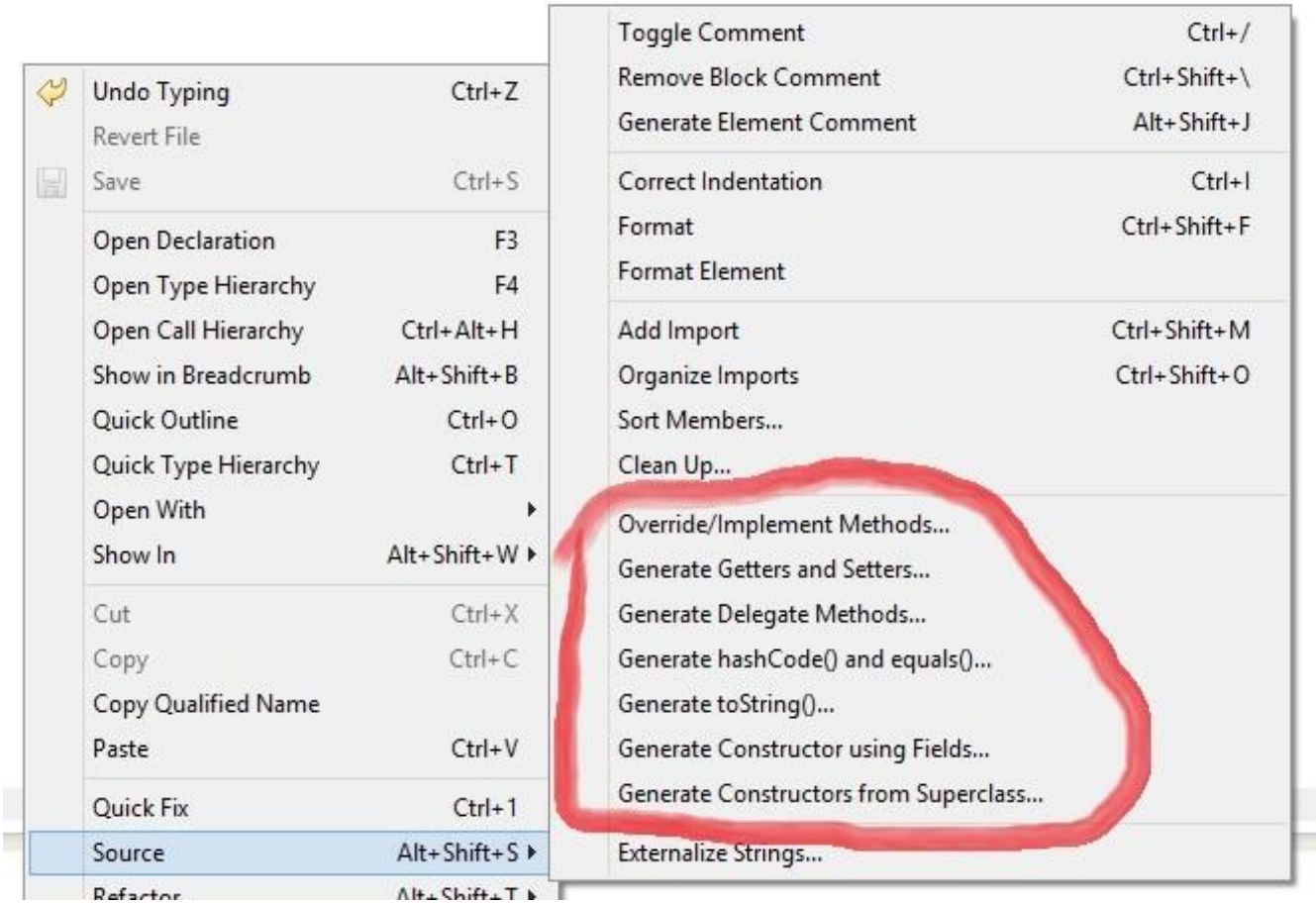
```
public class Prueba{
    public static void main(String [] args){

        int a = 3;
        int b = 5;

        int c = a + b;

        System.out.println("a + b = " + c);
    }
}
```

Autocompletado de código





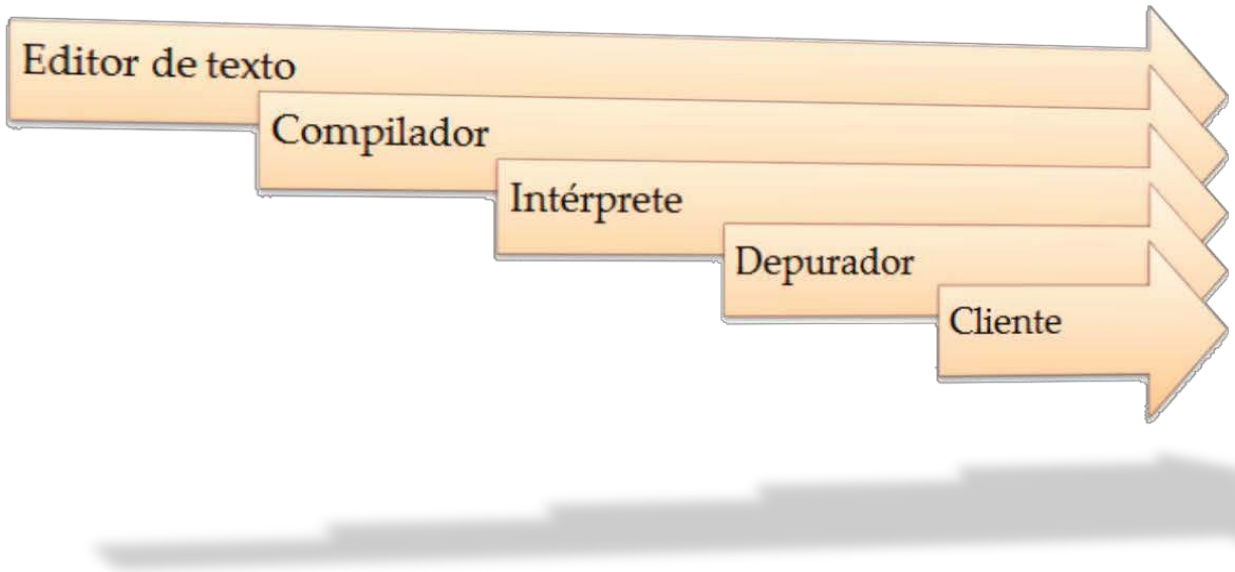
Creación automática de estructuras



PERSONALIZACIÓN Y CONFIGURACIÓN DE UN IDE

- Son altamente configurables
- Las necesidades de cada programador pueden ser diferentes
- Ofrecer una interfaz amigable al programador
- Posicionamiento de ventanas comunes, atajos de teclado, etc.
- Configuración de depuración de código
- Diferentes opciones de compilación

Un entorno de desarrollo integrado o **IDE** (*Integrated Development Environment*) es un programa informático que tiene el objetivo de asistir al programador en la tarea de diseñar y codificar un software mediante la inclusión de múltiples herramientas destinadas para dicha tarea.



Además de las herramientas comunes que todo **IDE** debe tener (Editor de texto, compilador, depurador…), un **IDE** aporta una serie de herramientas adicionales para cumplir con mayor eficacia el objetivo de facilitar el trabajo a los desarrolladores.

Una de las herramientas más útiles e importantes es el autocompletado de código y las inspecciones de las clases y objetos. En Visual Studio esa herramienta se llama ***IntelliSense***.

Los *snippets* nos permiten utilizar código reusable de una manera rápida y cómoda. Algunos entornos de desarrollo admiten la posibilidad de crear tus propios *snippets* para utilizarlos cuando quieras.

Como ya hemos comentado, la labor principal de un entorno de desarrollo es facilitar la vida al programador. Para cumplir éste propósito, los **IDEs** son altamente configurables y personalizables.

La configuración del IDE permite entre otras cosas añadir y modificar las barras de herramientas, pudiendo crear comandos personalizados y atajos de teclado para cada una de ellas. Estableciendo el posicionamiento de las ventanas y barras conjuntamente con los atajos de teclado podremos mejorar sumamente nuestro rendimiento y aprovechar con mayor comodidad todas las funciones del **IDE**.

La mayoría de los **IDEs** también permiten configurar interfaces diferentes dependiendo de la operación que se esté realizando, teniendo una configuración para la etapa de desarrollo y otra diferente para la etapa de depuración.

CRITERIOS DE ELECCIÓN DE UN IDE

SISTEMA OPERATIVO

Sin lugar a dudas, uno de los criterios más restrictivos a la hora de seleccionar nuestro entorno de trabajo es saber en qué sistema operativo vamos a trabajar y, más importante aún, para qué sistema operativo vamos a desarrollar nuestro software.

Siempre y cuando nuestro software no vaya a ser ejecutado mediante una **máquina virtual**, estaremos desarrollando para un **sistema operativo** concreto.
Esto realmente no se debe a una restricción inherente al **IDE**, sino al

compilador que tiene el IDE integrado.

Este problema es fácilmente salvable compilando nuestro código fuente en un compilador de otro sistema operativo (siempre y cuando exista), por lo que dependiendo de nuestras necesidades pudiera ser un problema menor.

LENGUAJE DE PROGRAMACIÓN Y FRAMEWORK

Un IDE puede soportar uno o varios lenguajes de programación, por lo que saber en qué lenguaje de programación vamos a codificar nuestro software y qué lenguajes nos ofrecen los distintos **IDE** es una información valiosa que hay que tener en cuenta.

Lo mismo ocurre con las plataformas de trabajo, también llamadas

***framework*.**

Este criterio va de la mano con el sistema operativo, ya que si quisiéramos desarrollar en **Visual Basic** bajo un sistema operativo **Linux** no sería **Visual Studio** nuestra opción, sino que tendríamos que utilizar **Gambas**.

HERRAMIENTAS Y DISPONIBILIDAD

Las diferentes herramientas de las que disponen los IDE son el último criterio de selección, seguramente nos encontremos con varios IDE que cumplen los requisitos de lenguaje y sistema operativo, pero no todos tienen las mismas funciones, por lo que saber cuáles son esas herramientas es un dato sumamente importante en nuestra decisión.

En ocasiones pueden ser restrictivos ya no solo por tus propias preferencias, sino por trabajar de manera colaborativa y el modo de utilizar e interpretar diferentes códigos entre diferentes **IDE**.

Los entornos de desarrollos generan código automáticamente siguiendo un patrón propio que suele ser único. Es por ello que realizar un proyecto de forma colaborativa donde se utilice el mismo lenguaje y el mismo *framework* puede resultar una experiencia diferente dependiendo del entorno de desarrollo que se utilice.

➢ HERRAMIENTAS Y DISPONIBILIDAD

Fuera del marco de trabajo colaborativo, también tendremos nuestras preferencias y necesidades, por lo que, si necesitamos tener una funcionalidad concreta, deberemos encontrar un IDE que nos la facilite.

Podríamos también ir más allá de las meras funcionalidades añadidas e irnos a un ámbito mucho más centrado en el propio software, como podría ser la interfaz de usuario, los IDE pueden incluir sus propios y específicos controles que mejoran la interfaz y aportan mayor funcionalidad y usabilidad a nuestros formularios y aplicaciones. También podríamos ver este criterio desde un punto de vista más destinado a la codificación y pensar en qué refactorizaciones automáticas nos podemos encontrar entre los IDE a la hora de elegirlo.

Una vez comprobados todos los criterios de selección mencionados tendríamos que comprobar si el IDE que cumple los requisitos está a nuestro alcance, ya sea por una cuestión de presupuesto o localización.

¿Cuál es el mejor vehículo?

¿Cuál es el mejor IDE?

¿Requisitos?

¿Características?

¿Necesidades?

Criterio de SSOO

- Saber en qué sistema operativo se va a trabajar con el IDE

- Saber para qué sistema operativo se va a desarrollar el software
  - Utilizar compilador acorde al SSOO
- Se puede utilizar máquina virtual para desarrollar sobre otro SSOO
  - No se aconseja

Criterio de Lenguaje de programación

- Tener en cuenta el lenguaje de programación que se necesita para el desarrollo del software
- Conocer los lenguajes de programación que soportan los IDEs
- El criterio lenguaje de programación – SSOO van relacionados
  - Visual Basic
    - Windows → Visual Studio
    - Linux → Gambas

Criterio de herramientas

- Conocer las herramientas de que disponen los IDEs
- Aunque un IDE cumpla con las características básicas de un lenguaje de programación, puede tener herramientas que faciliten el desarrollo del software frente a otros
  - Sistema de control de versiones
    - Team Foundation Server (TFS)
      - Solo se utiliza en Visual Studio
    - Subversion (SVN)
      - Netbeans
      - Eclipse
      - InteliJ DEA

Otros criterios…

- Preferencias y necesidades del programador
- Refactorización de software automática
- Uso de diagramas
- Configurable a base de plugins
  - Video 1

USO BÁSICO DE UN IDE

Parece obvio pensar que el uso básico de un IDE consiste en **desarrollar software**, y no sería un pensamiento equivocado. No obstante, esa misma la tarea la podríamos realizar con un editor de texto y un compilador, por lo que el uso de un IDE va más allá de la simple edición de código.

Muchas de las herramientas más habituales que solemos usar conjuntamente con los IDE también se encuentran disponibles fuera de ellos, como las herramientas de modelado o de pruebas unitarias. En esencia, todas esas aplicaciones pueden o podrían estar disponibles sin necesidad de un entorno de desarrollo, o mejor dicho, sin que fuese un entorno de desarrollo integrado pero nos veríamos forzados en ese caso a utilizar diferentes aplicaciones de manera simultánea, que además, no tendrían por que comunicarse de manera eficiente entre ellas.

Es por ello, que la verdadera usabilidad del entorno de desarrollo consiste una vez más en ofrecer una comodidad y una asistencia a un trabajo que podríamos hacer sin él, del mismo modo que podríamos subir al noveno piso de un edificio sin ascensor.

La necesidad básica que todo IDE debe cubrir es la creación o edición de programas y convertir ese código fuente en código ejecutable.

Un IDE realiza esa operación de manera conjunta y compacta. Gracias a un gestor de proyectos podemos ajustar las dependencias de cada sección de nuestro programa y todas las necesidades y opciones de compilación que queramos.

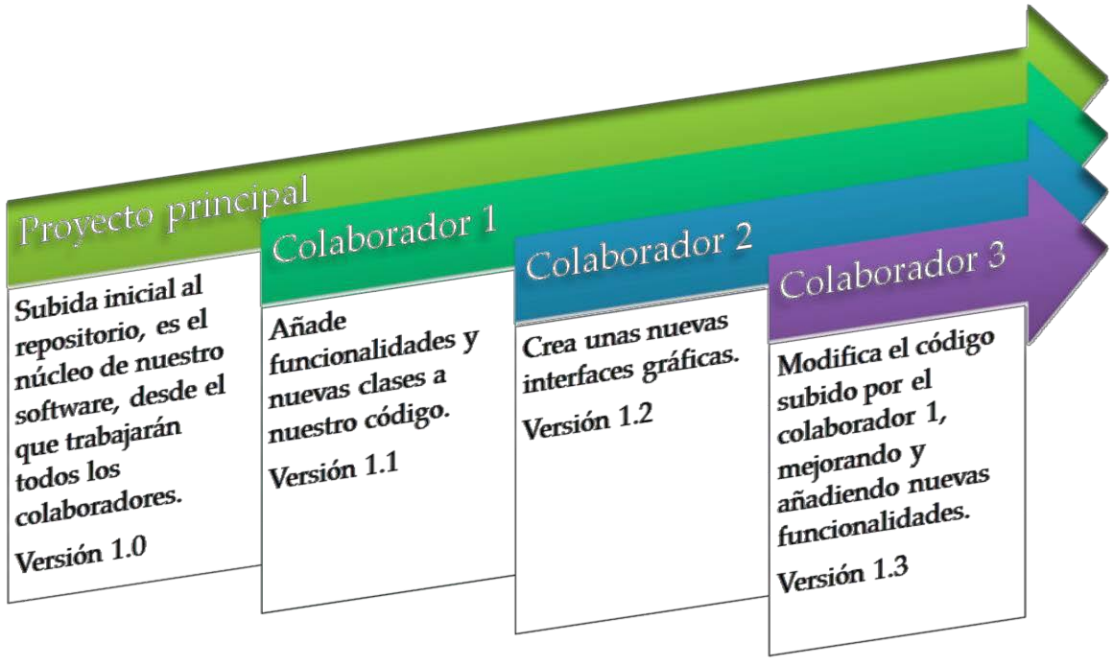
Los IDE, además, suelen ofrecer una funcionalidad añadida, ya que permiten ejecutar de manera virtual el programa que se está codificando en cualquier momento siempre y cuando no tenga errores de compilación.

Los proyectos sobre los que vamos a trabajar en un IDE pueden requerir de un grupo de trabajo, y ese grupo de trabajo puede requerir un proyecto de **desarrollo colaborativo**.

Para ello haremos uso de programas de **control de versiones** integradas en el entorno de desarrollo. Los programas de control de versiones son aplicaciones que constan de servidor y cliente, donde en la parte del servidor se crean repositorios para que los clientes puedan descargar y subir código. Son herramientas asíncronas que permiten controlar y gestionar las fuentes y versiones del código del repositorio.

Gracias a las herramientas del **IDE**, podemos hacer un uso mucho más rápido y avanzado de los controles de versiones, pudiendo elegir qué archivos actualizar en cualquier lado de nuestra conexión (servidor o cliente), omitir cambios para no pisar nuestro trabajo con el de otros, y viceversa, y una gran cantidad de operaciones de la misma índole.

La sincronización de un IDE con el repositorio del proyecto permite además saber qué archivos han cambiado y por ende tener un control de versiones más allá del servidor, tenerlo directamente sobre el código que estamos actualmente desarrollando.



ECLIPSE

Es un IDE de código abierto. Es una plataforma potente con un buen editor, depurador y compilador. El JDT (Java development toolkit) es de los mejores que existen en el mercado y tiene detrás a una gran comunidad de usuarios que van añadiendo mejoras al software.

Fue desarrollado por IBM como evolución de su VisualAge, pero ahora lo mantiene la fundación Eclipse, que es independiente y sin ánimo de lucro.

Tenía licencia CPL (common public license), pero luego la fundación cambió dicha licencia por una EPL (Eclipse public license).

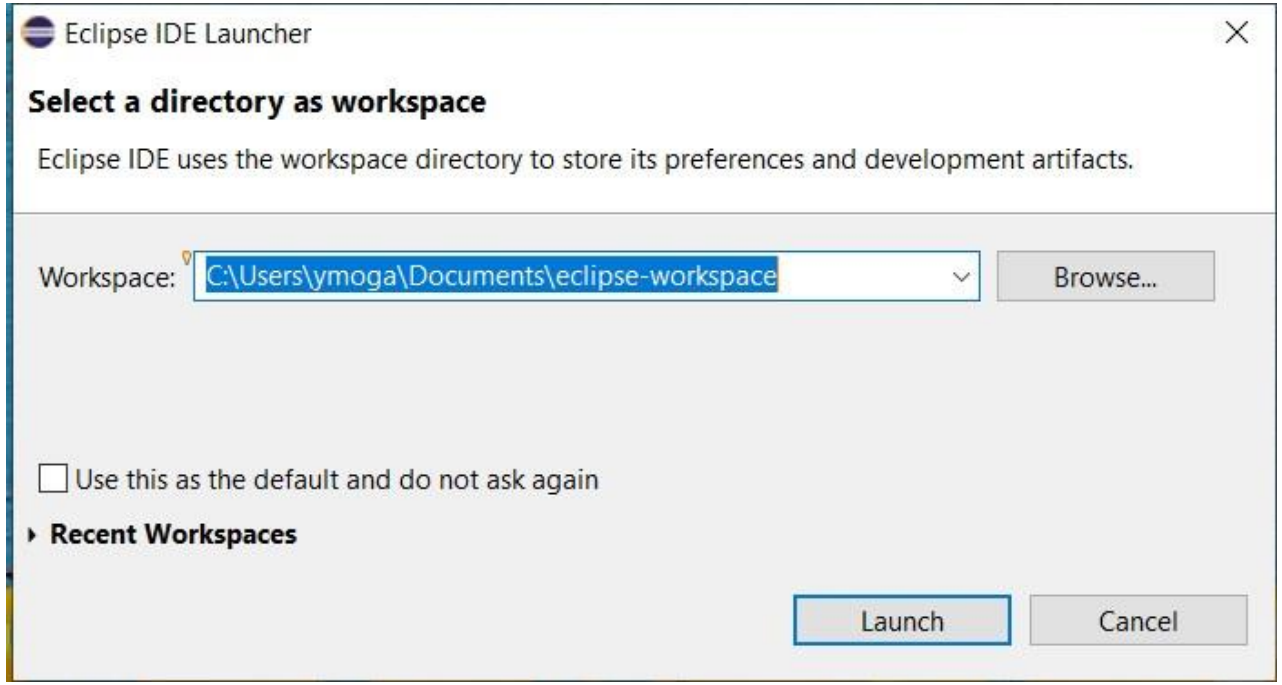
Antes de instalar Eclipse se debe tener instalado un JDK (Java development kit). Es el software utilizado para desarrollar, incluye:

- Compilador de Java (javac).
- JRE (Java Runtime Environment) es un conjunto de utilidades que permite la ejecución de programas Java.
- JVM (Java Virtual Machine) ejecuta instrucciones generadas por el compilador Java.

Una vez instalado el JDK instalamos Eclipse desde el enlace: https://www.eclipse.org/downloads/packages/

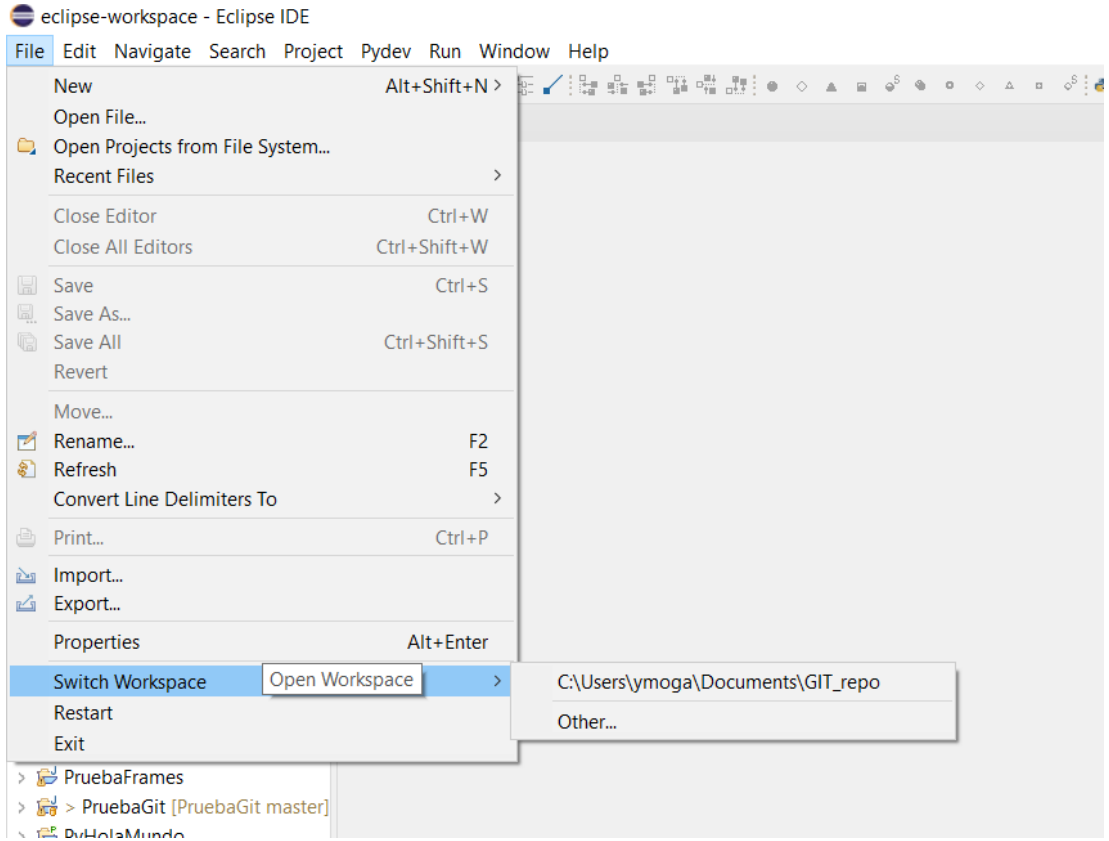
CONFIGURACIÓN WorkSpace

La primera vez que ejecutemos el **IDE**, nos preguntará donde queremos tener nuestro lugar del trabajo "**eclipse-workspace**". Le indicaremos la ruta donde queremos tener el espacio de trabajo donde se guardarán los proyectos que creeemos, por ejemplo: **C:\Users\usuario\eclipse-workspace**. Si queremos que no nos vuelva a preguntar donde ubicar el "**workspace**", marcamos el check que hay debajo para que siempre use por defecto la ruta indicada.



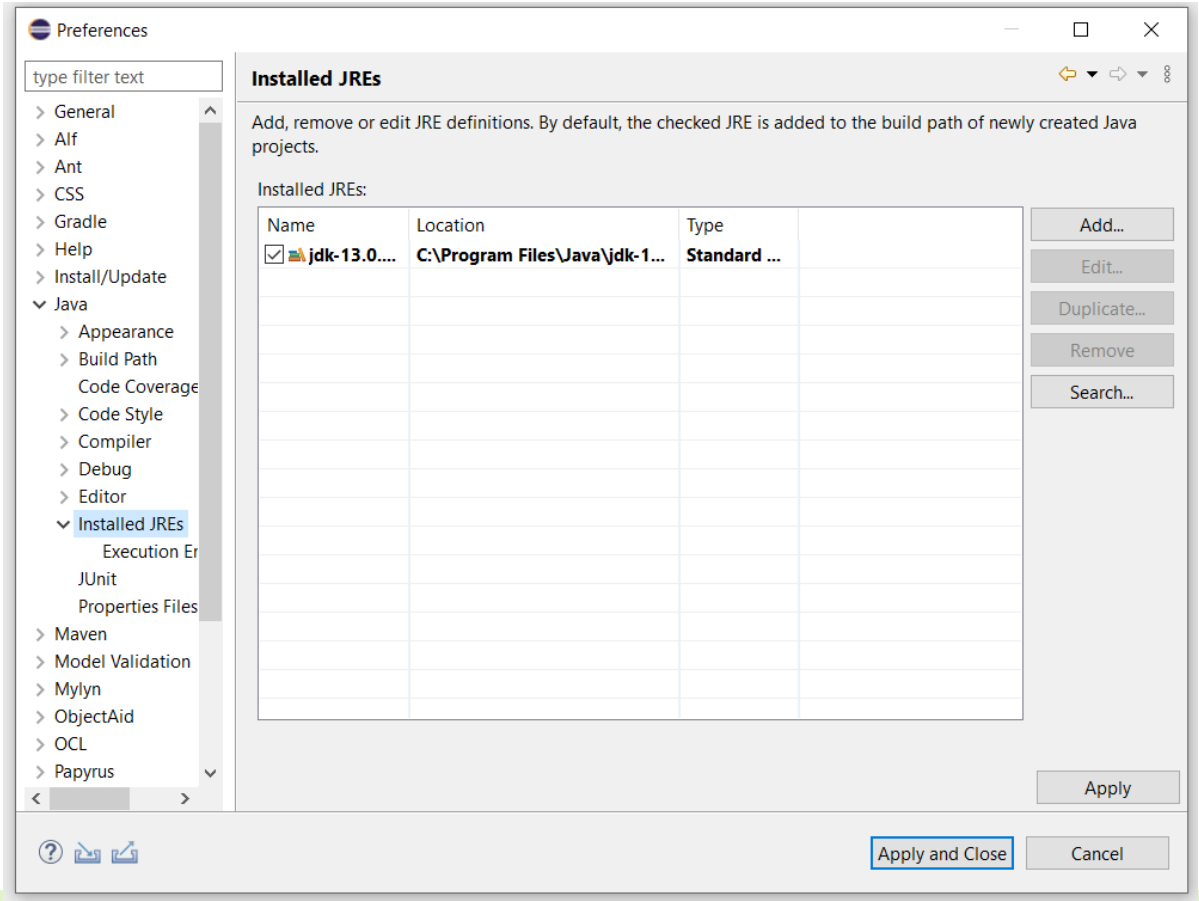
CONFIGURACIÓN WorkSpace

Una vez dentro de un espacio de trabajo, podemos cambiarnos o crear uno nuevo desde el menú File→Switch WorkSpace, podremos elegir uno que ya tengamos creado o crear nuevo espacio:



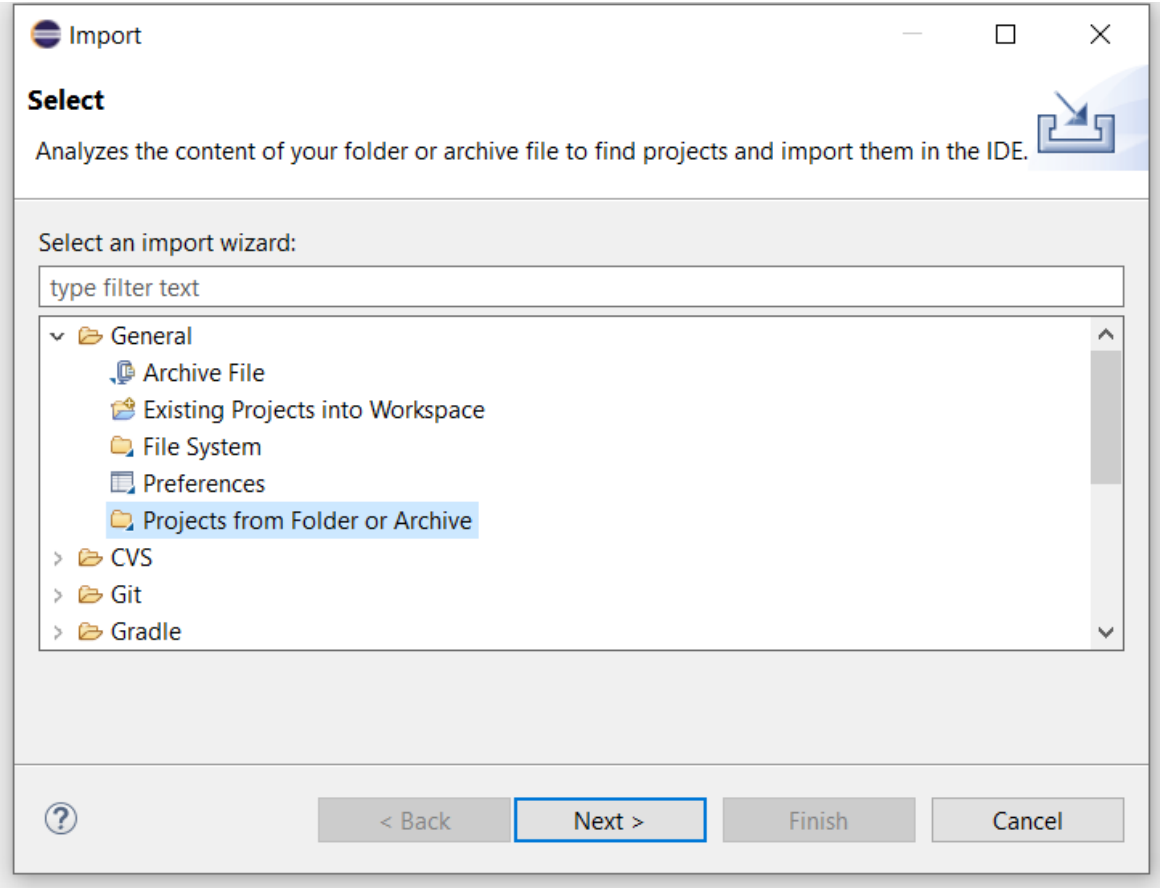
CONFIGURACIÓN JDK dentro de Eclipse

Si no lo tenemos configurado ya y necesitamos configurar el JDK instalado dentro de eclipse lo podemos hacer desde el menú Window→Preferences, en el apartado de Java→Installed JREs nos aparecerá el que estamos utilizando y podemos añadir otro en el botón "Add":



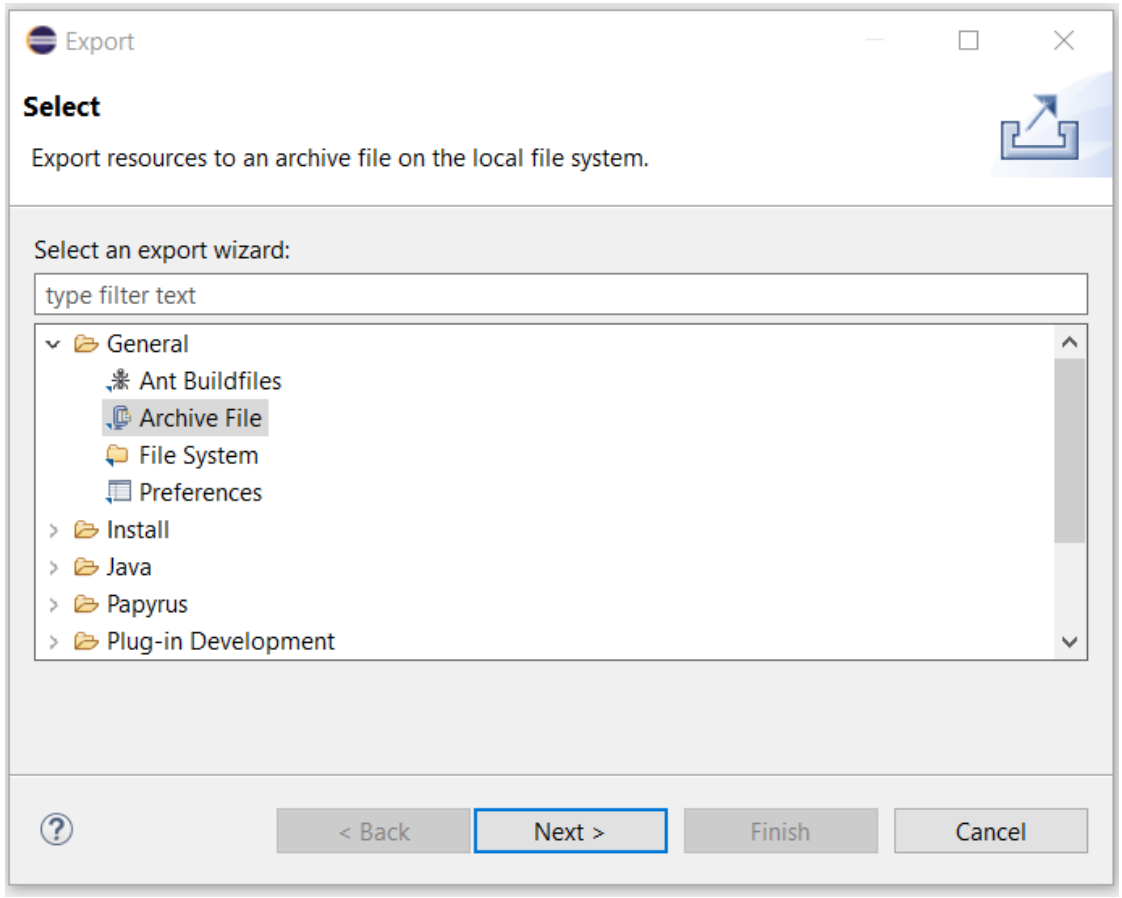
Importar y exportar proyectos

Para importar un proyecto Java a nuestro WorkSpace se hará desde el menú File→Import:



Importar y exportar proyectos

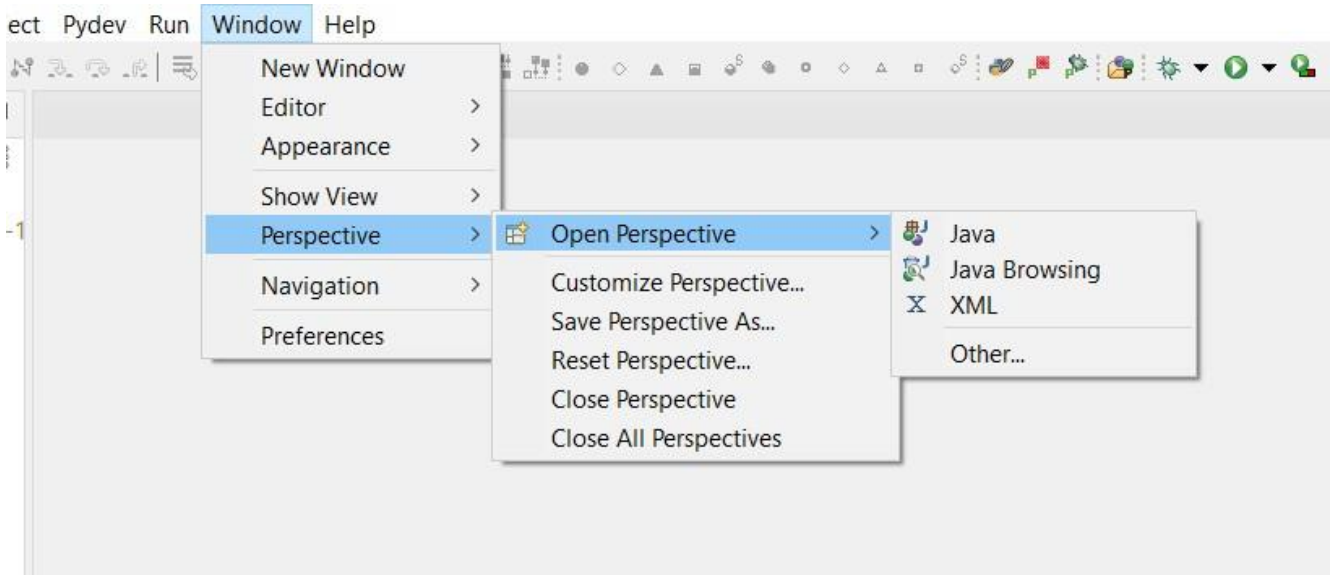
Para exportar un proyecto de Eclipse, una de las formas de hacerlo es desde el menú File→Export:



Vistas y perspectivas

Eclipse divide el espacio de trabajo en **vistas** y **perspectivas** que podemos configurar a nuestro gusto para trabajar más cómodos.

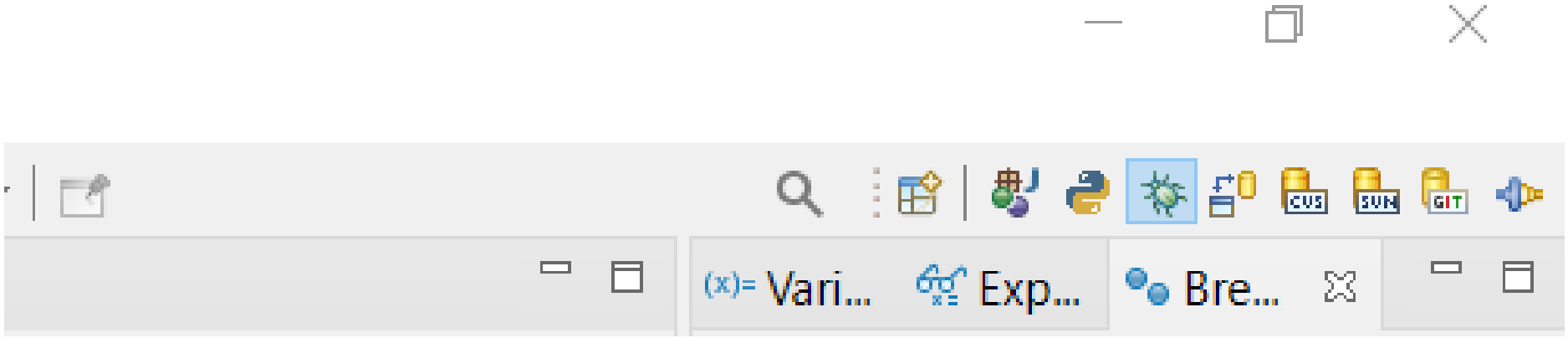
La **perspectivas** son una preconfiguracion de vistas relacionadas entre sí para trabajar de forma óptima. Destacan aquí la perspectiva de "**Java**" y "**Depurar**" (**Debug**).Para poder cambiar de **perspectiva**, cerrar o modificarla lo podemos hacer desde el menú Window→Perspective:



Si no aparece la perspectiva que queremos abrir, en la opción Other nos aparecerán todas las que tenemos disponibles.

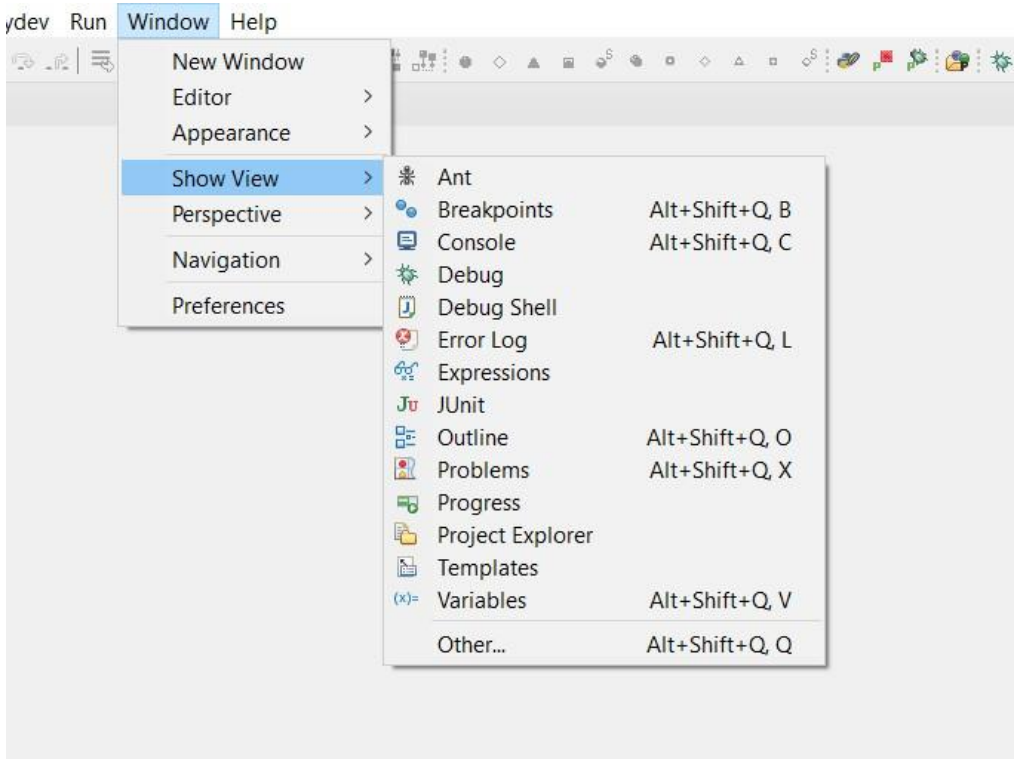
Vistas y perspectivas

También podemos cambiar de perspectiva desde la barra de acceso rápido:



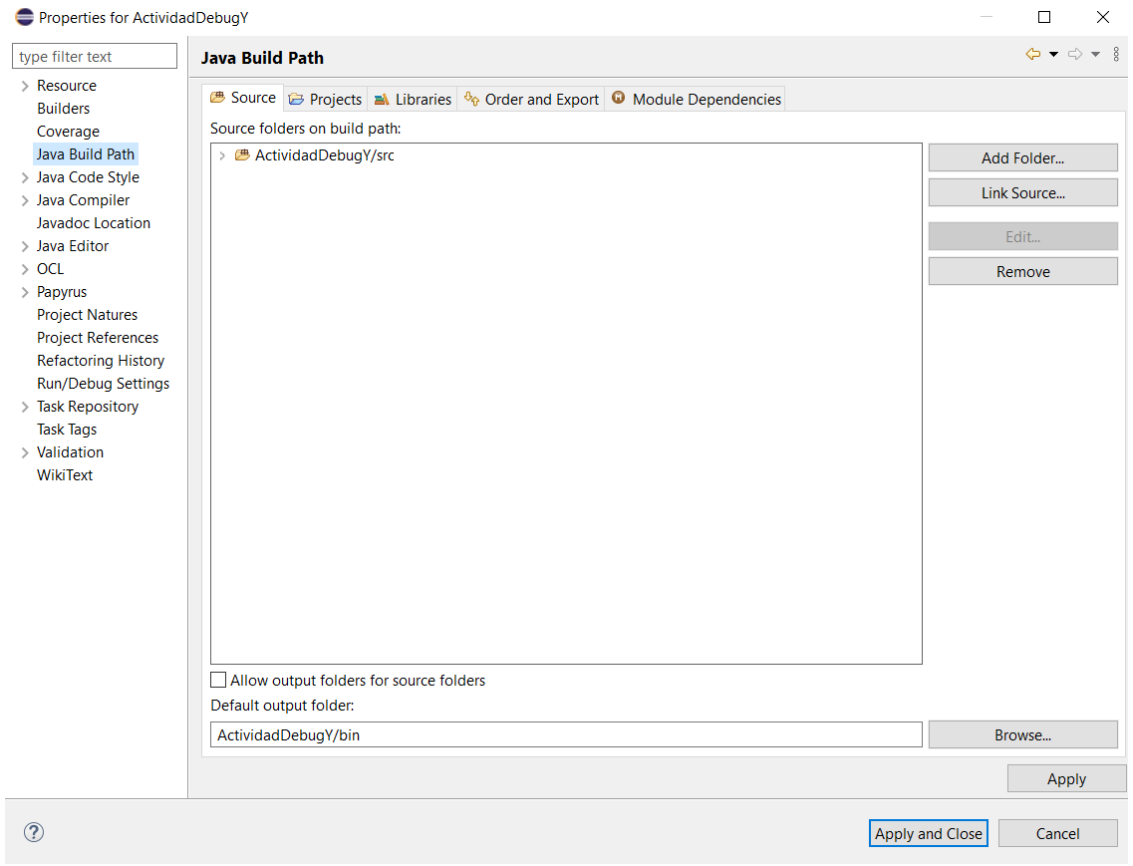
Vistas y perspectivas

Las **vistas** son los paneles en las que se divide el espacio de trabajo proporcionando funciones de apoyo. Podemos mostrarlas desde el menú Window→Show view:



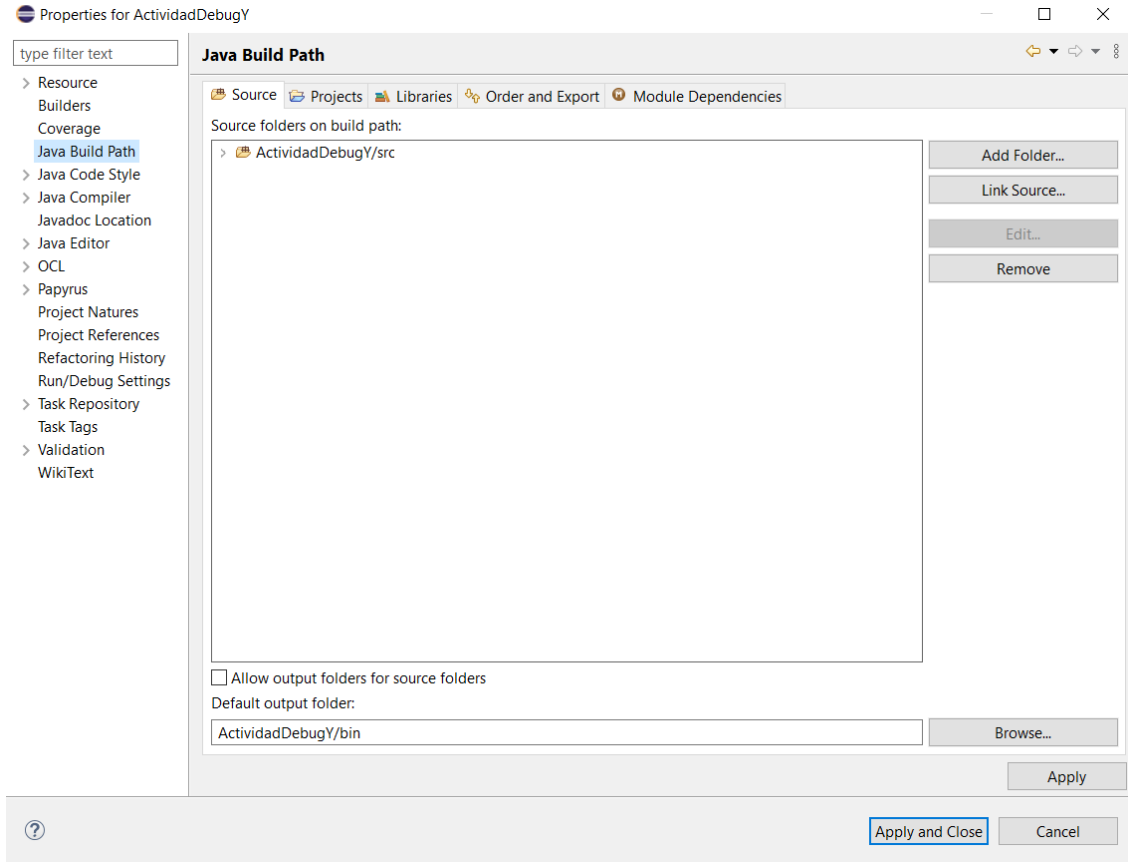
Propiedades del proyecto

En el menú Project→Properties se mostrarán todas las propiedades del proyecto que tenemos seleccionado. Una de las partes más importantes a conocer de esta ventana es el apartado “Java Build Path”:



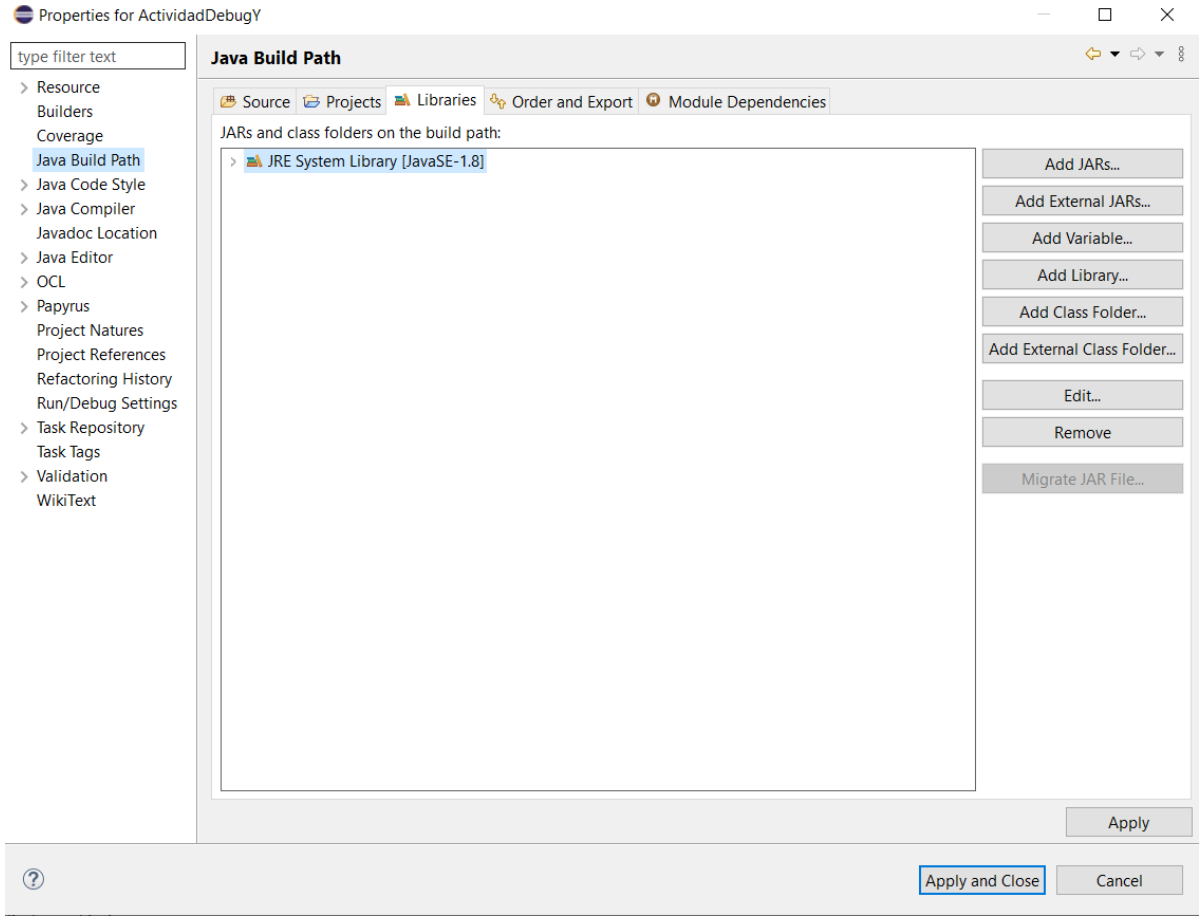
Propiedades del proyecto. Java Build Path.

La primera pestaña que nos encontramos es la de “Source”, aquí se puede configurar la ruta para los archivos .java (código fuente) y .class (bytecode que será interpretado por la máquina virtual de Java) de nuestro proyecto:



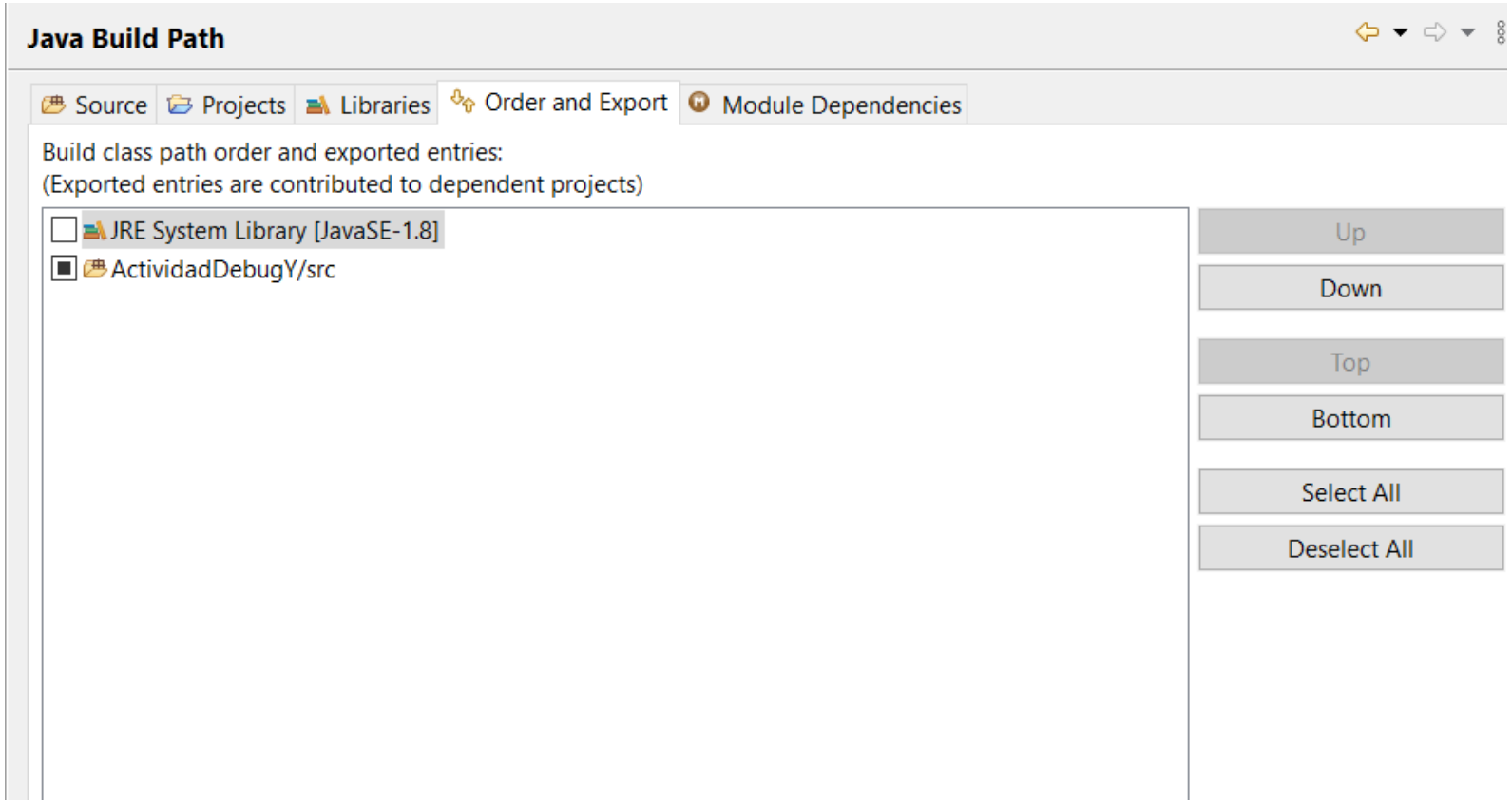
Propiedades del proyecto. Java Build Path.

Pestaña “Libraries”, en esta pestaña nos aparecerá la librería de Java que estamos utilizando (JRE). Si el proyecto necesita utilizar otras librerías las podemos añadir desde esta ventana:



Propiedades del proyecto. Java Build Path.

Pestaña “Order and Export”, determina el orden en que los proyectos y las bibliotecas aparecen en el classpath. Export determina que los proyectos y bibliotecas serán exportados y por lo tanto disponible en otros proyectos que dependen de éste.



HERRAMIENTAS DE DEPURACIÓN

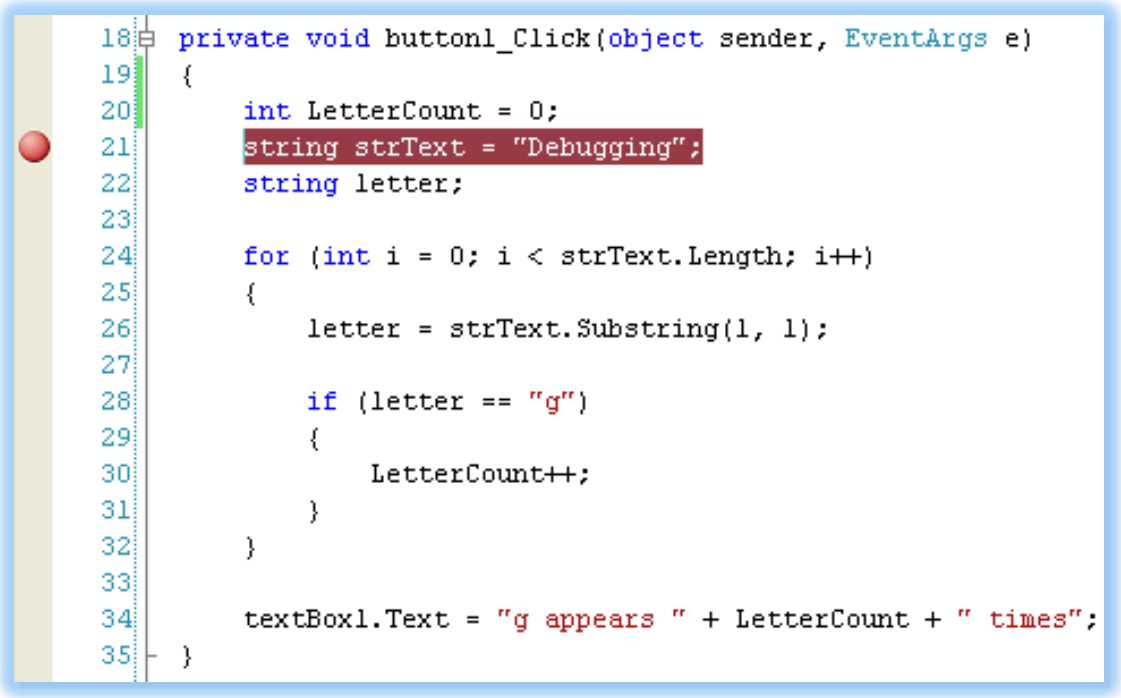
La **depuración** es uno de los procesos más importantes en el desarrollo software. Permite identificar errores mediante la ejecución controlada del software.

El depurador de código o ***debugger*** es una de las herramienta más importantes en un IDE.

Con el *debugger* se puede:

- Controlar completamente el flujo del programa.
- Ver el proceso de un programa paso a paso.
- Observar el valor de los métodos, variables y objetos.
- Establecer puntos de control para interrumpir la ejecución.
- Etc.

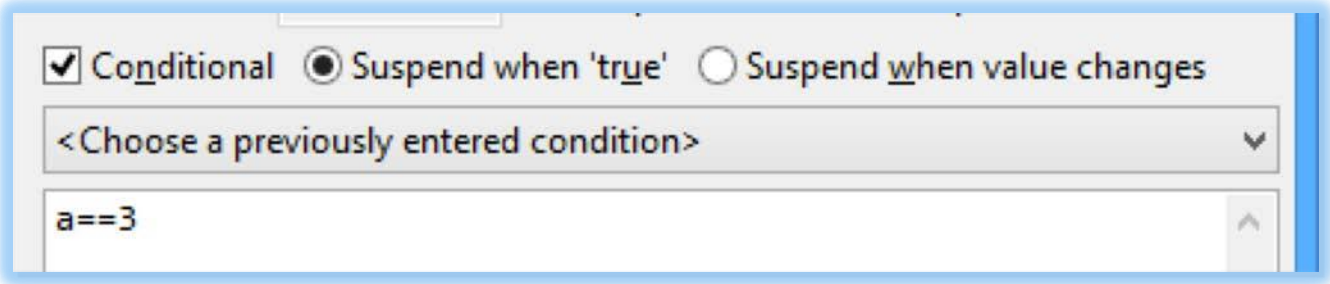
- Los puntos de ruptura, puntos de interrupción o breakpoints son puntos de control situados en líneas concretas de nuestro código fuente
- Cuando el depurador pasa por este punto detiene la ejecución del programa
- Se pueden colocar todos los puntos de interrupción que se desee



Los *breakpoints* se pueden personalizar añadiendo una condición o filtro.

Se establece la condición de parada que se tendrá que cumplir para que se detenga la ejecución en el *breakpoint.*

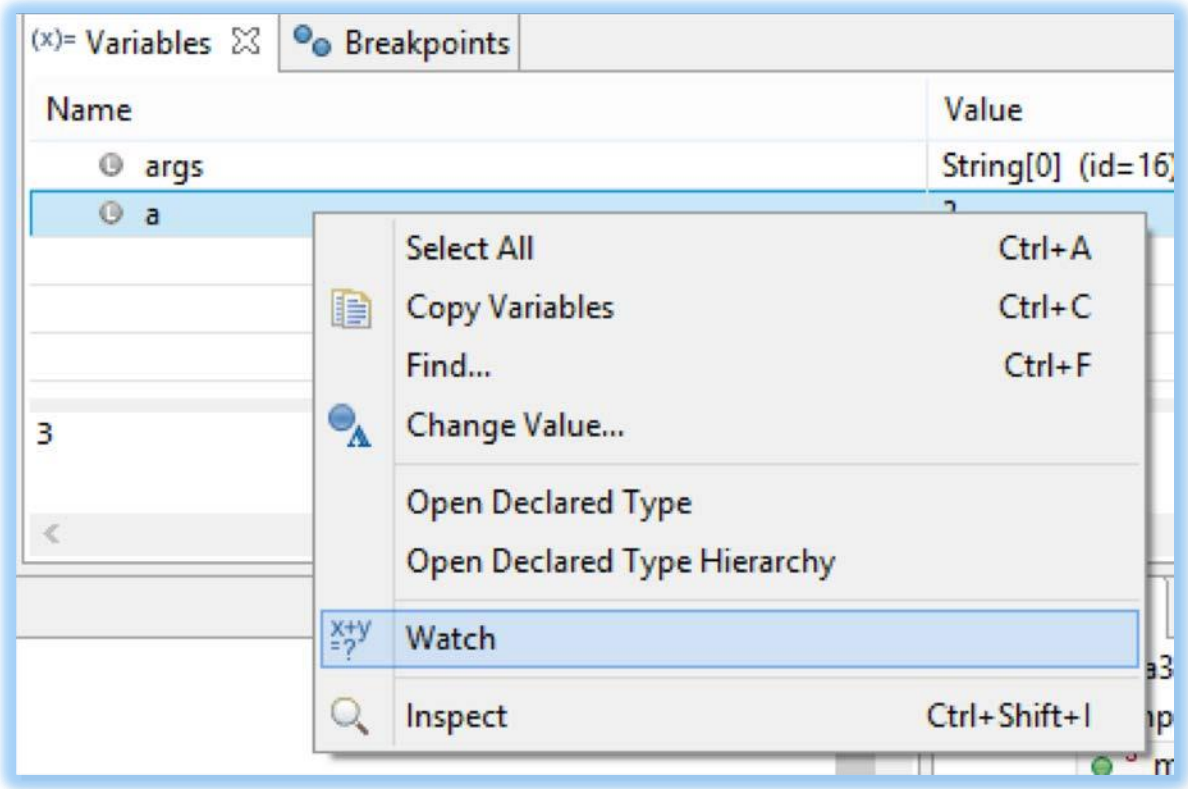
```
8          if(a==3) {
9              System.out.println("Se para solo si es igual a 3");
10         }
11
12         System.out.println("Si a es distinto de 3 se viene directamente a este breakpoint");
```

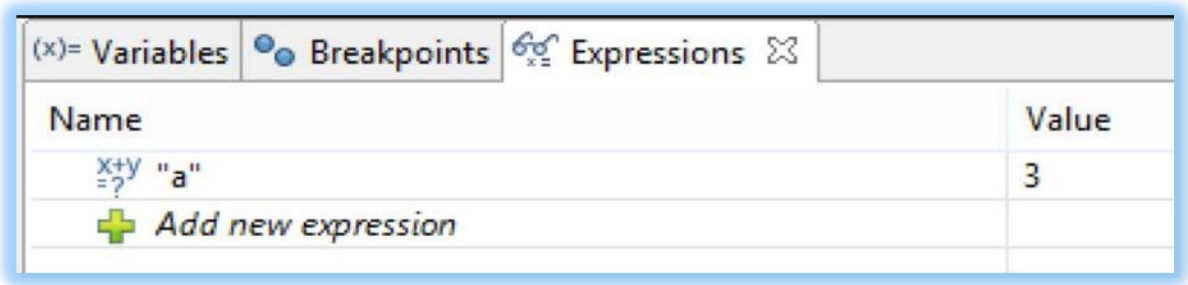


Durante la depuración de código es importante conocer:

- El valor que tienen las variables.
- Los atributos de los objetos.
- El resultado de un condicional.

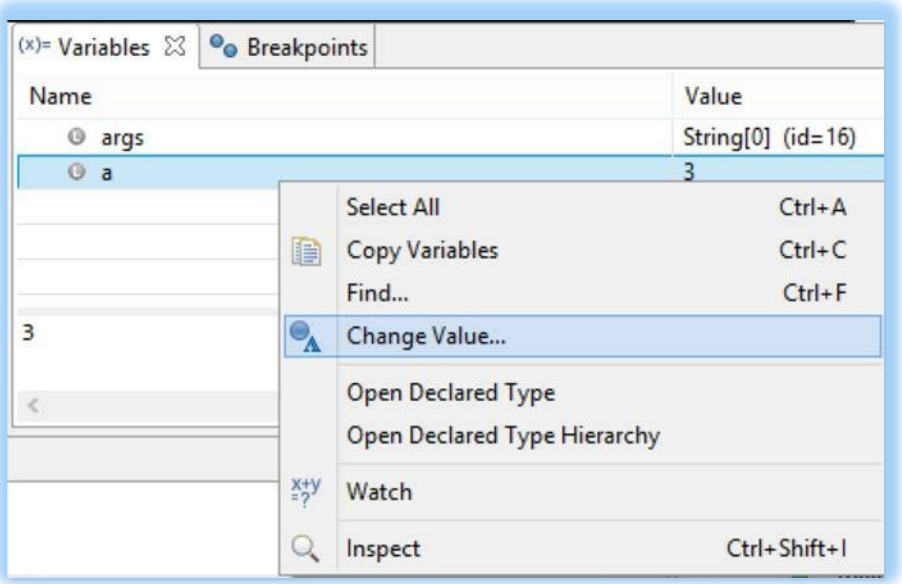
Para conocer siempre el valor de ciertos datos se genera una inspección de los que más interese.

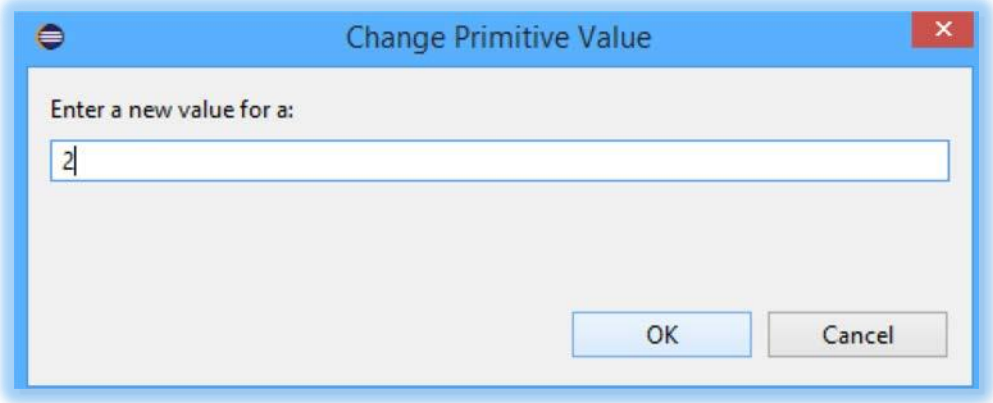




Puede ser interesante modificar el valor de las variables en tiempo de ejecución para forzar al programa a que pase por ciertos módulos que de forma habitual nunca pasaría.

```
8          if (a==3) {
9              System.out.println("Se para solo si es igual a 3");
10         } else {
11             System.out.println("Se ha cambiado el valor de a en tiempo de ejecución");
12         }
```





En la barra de botones principal, al lado de imprimir, vemos los botones para manipular la ejecución de la depuración de código.

Las funciones de estos botones, por orden, son:

1. Resume(F8); continúa con la ejecución (hasta el próximo breakpoint).
2. Suspend; podemos detener la ejecución aunque no alcancemos un breakpoint (muy útil cuando entramos en un ciclo infinito).
3. Stop; detiene la depuración.
4. Step Into (F5); se detiene en la primer línea del código del método que estamos ejecutando. Si no hay método, hace lo mismo que Step Over.
5. Step Over (F6); pasa a la siguiente línea que vemos en la vista de código.
6. Step Return (F7); vuelve a la línea siguiente del método que llamó al método que se está depurando actualmente. O lo que es lo mismo, sube un nivel en la pila de ejecución, que vemos en la vista Debug.

ANÁLISIS DE CÓDIGO

Para desarrollar software es necesario hacer un buen diseño para que:

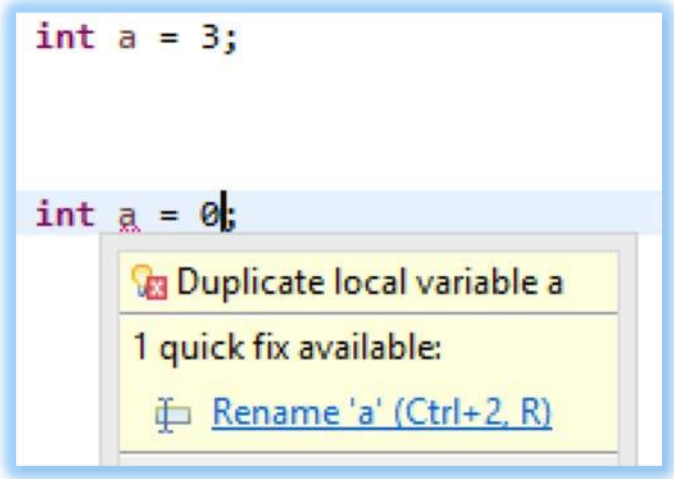
- El código funcione correctamente
- Se cometan el menor número de errores posible

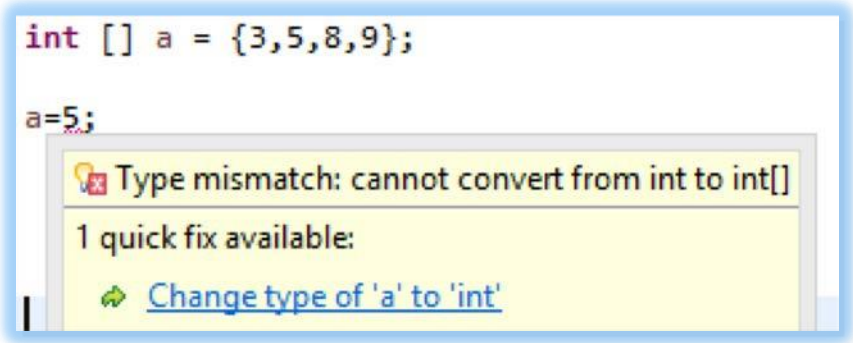
Aunque el ser humano siempre comete errores:

- De funcionalidad
- De coherencia
- De sintaxis

Los IDEs disponen de analizadores de código en tiempo real para ayudar a mitigar estos errores del programador


int a=3;
String b = "hola";
b = 10;
if
} e
}
Type mismatch: cannot convert from int to String
1 quick fix available:
Change type of 'b' to 'int'


int a = 3;
int a = 0;
Duplicate local variable a
1 quick fix available:
Rename 'a' (Ctrl+2, R)


int [] a = {3,5,8,9};
a=5;
Type mismatch: cannot convert from int to int[]
1 quick fix available:
Change type of 'a' to 'int'

- Es posible configurar el nivel de ayuda de los IDEs
- Se recomienda dejar los valores por defecto establecidos para cada lenguaje de programación