



CIPFP Mislata

Centre Integrat Públic
Formació Professional Superior

Entornos desarrollo

Autor: Joan Puigcerver Ibáñez

Correo electrónico: j.puigcerveribanez@edu.gva.es

Curso: 2022/2023

Este material es una obra derivada a partir del material de: **Sergio Badal**

Licencia: BY-NC-SA

(Reconocimiento - No Comercial - Compartir Igual)



1. Entornos desarrollo

Un **Entorno de Desarrollo Integrado o Integrated Development Environment (IDE)**:

es una aplicación informática cuyo objetivo es asistir al programador en la tarea de diseñar y codificar un software mediante la integración de múltiples herramientas destinadas a esta tarea" (Casat, 2012:46).

En definitiva, es una aplicación informática que facilita la labor de codificación y desarrollo de software.

Los elementos básicos que contiene son:

- Un **editor de texto**: Para escribir y modificar el código fuente.

Los IDEs modernos tienen herramientas que asisten a esta edición:

- Resaltar la sintaxis (syntax highlight)
- Autocompletado de código (autocompletion)
- Macros (En IntelliJ, *Live Templates*).

- Un **compilador** o **intérprete**: Para traducir y ejecutar el código fuente y generar ejecutables.
- Un **depurador (debugger)**: Herramienta que ayuda a encontrar errores lógicos en el código.

Sus principales características son:

- Pueden ser para uno o varios lenguajes de programación.
- Ayudan a la visualización del código fuente.
- Permite moverse rápidamente entre los archivos de la aplicación.

Usar IDE, por tanto, supone una ayuda a la programación. Sin embargo, generan dependencia, ya que el desarrollador debe acostumbrarse a su funcionamiento. Además, consumen más recursos y algunos son de pago.

Hoy en día existen editores de texto que realizan estas funciones básicas (Sublime Text, Visual Studio Code, Notepad++, Texpad, KLite...). Todos ellos identifican y resaltan la sintaxis y algunos permiten la instalación de extensiones para incorporar herramientas y que parecen IDEs.

2.¿Por qué utilizar un IDE?

No obstante, las funcionalidades básicas, los modernos IDEs ofrecen más funcionalidades.

Una de las más importantes es, el **autocompletado de código**: cada vez que empezamos a escribir una palabra reservada aparece una ayuda contextual que indica las opciones más comunes para completarlo automáticamente.

También permite crear estructuras de clases o instrucciones de bucles automáticamente mediante **macros** o palabras clave.

Además, ofrecen herramientas para refactorizar, ejecutar depurador y más opciones que ayudan a la programación y que hacen más corto el ciclo de vida del software.

Los IDEs son *altamente configurables*. Cada desarrollador puede configurarlo cuanto más le guste: con más o menos barras de herramientas, con atajos de teclado, con órdenes personalizadas, con diferentes ubicaciones físicas de los distintos paneles, etc. Pero también permite configurar la depuración y compilación de proyectos, convirtiéndose así en una herramienta cómoda y útil, que facilita y agiliza el desarrollo de software.

Por último, los IDEs se integran fácilmente con **herramientas de control de versiones**, que permite realizar el desarrollo de una manera descentralizada, distribuida y colaborativa. De esta forma, varios desarrolladores pueden trabajar sobre el mismo proyecto, como ocurre en el software libre. El control de versiones se realiza creando repositorios donde cada desarrollador puede publicar los

cambios que va realizando sobre el código, y se pueden crear diferentes ramas para realizar diferentes tareas a la vez.

A la hora de elegir un IDE debemos tener en cuenta varios factores:

- **Sistema operativo:** Hoy en día la gran mayoría de IDEs son multiplataforma, pero hay algunos que sólo se pueden instalar en alguna plataforma en concreto.
- **Lenguaje de programación:** un IDE puede soportar uno o varios lenguajes de programación, pero puede ser se comporta mejor en uno que con otros.
- **Framework:** El IDE que queramos utilizar debería soportar el framework que estamos gastando.
- **Herramientas:** también se elige un IDE por la oferta de herramientas que tiene. No todos los IDE tienen las mismas funciones.
- **Licencia:** No todos los IDEs son gratuitos y no todos los gratuitos permiten desarrollar software en cualquier condición.

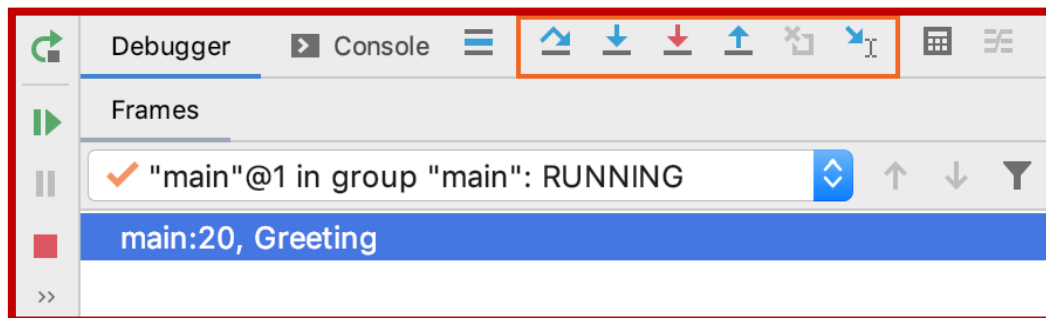
3.Debugger

El **depurador o debugger** es una herramienta que permite visualizar el flujo de ejecución de un programa para comprobar el correcto funcionamiento del mismo y detectar posibles errores lógicos (aquellos errores que son sintácticamente correctos, pero el código no hace lo que se espera de él). Permite observar las variables definidas y el valor que toman en cada momento.

Para iniciar el depurador, es necesario ejecutar el programa utilizando la opción **Debug**.

Cada IDE tiene una interfaz diferente para realizar la depuración del código, pero existen unos elementos básicos que todos los depuradores tienen:

- **Breakpoint:** Punto de parada. El debugger para la ejecución del programa en la línea seleccionada y nos permite ver el valor de las variables definidas en ese momento concreto.
- **Resume execution:** Continúa la ejecución del programa normalmente, hasta que encuentra otro breakpoint o termina.
- **Step over:** Ejecuta la siguiente instrucción. Si es un método, ejecuta todo el contenido del mismo y continúa.
- **Step into:** Ejecuta la siguiente instrucción. Si es un método, continúa la ejecución desde dentro del método.
- **Step out:** Ejecuta la siguiente instrucción. Si es un método, continúa la ejecución desde fuera del método que se está ejecutando actualmente.



Para una explicación más detallada, es necesario consultar la documentación oficial del IDE. Puede encontrar el de IntelliJ Idea [aquí](#).

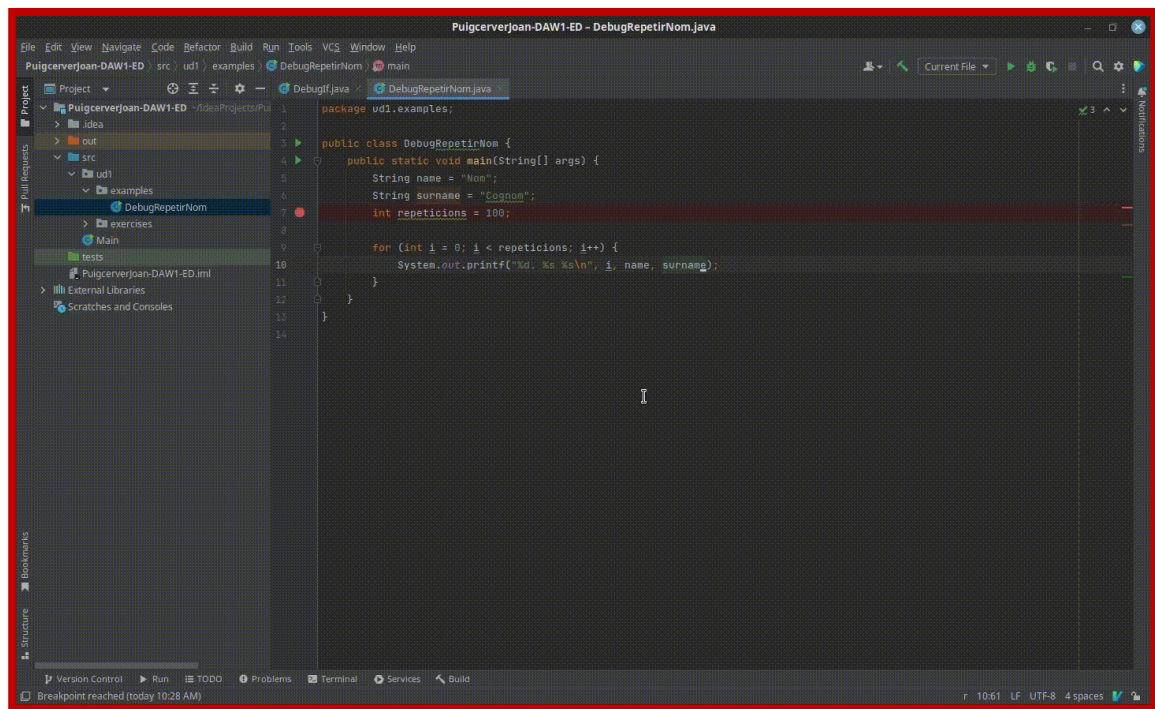
Ejemplo

- [DebugRepetirNom.java](#)







```
package ud1.examples;

public class DebugRepetirNom {
    public static void main(String[] args) {
        String name = "Nom";
        String surname = "Cognom";
        int repetitions = 100;

        for (int i = 0; i < repetitions; i++) {
            System.out.printf("%d. %s %s\n", i, name, surname);
        }
    }
}
```



El objetivo de este ejemplo es mostrar el funcionamiento básico del depurador.

1. Ante todo, se ha configurado un  **Breakpoint** en la línea 7. Esto indica al depurador que debe parar la ejecución del programa en esta línea. Puede pulsar junto al número de la línea para marcar o desmarcar el breakpoint.
2. A continuación, se ha comenzado la ejecución del programa en modo de depuración mediante el botón  **Debug**. También se puede realizar mediante el menú contextual que aparece si le das al botón derecho en el código fuente.
3. El programa comienza a ejecutarse y se para en la línea 7. Podemos observar en la pestaña *Debugger* cómo se han definido las variables `name` y `surname` y han tomado los valores "Nombre" y "Apellido" respectivamente.
4. En este punto, el depurador nos permite controlar manualmente la ejecución del programa. Mediante el botón  **Step over** se ha indicado que deseamos ejecutar la siguiente instrucción. Puede observarse que se ha creado la variable `repetitions` con valor `100`.
5. A continuación, se muestra la ejecución de un bucle `for`, que repite el código tantas veces como el valor en `repetitions` (`100`).
 - En la pestaña *Console*, se puede consultar los mensajes que se imprimen desde el programa mediante la instrucción `System.out.print`.
 - En la pestaña *Debugger*, se puede observar que se ha definido la variable `i`, que se utiliza para controlar cuántas veces se repite el bucle `for`.
6. Por último, se muestra cómo continuar o parar la ejecución del programa.
 - El botón  **Resume execution** permite continuar la ejecución del programa hasta que encuentre otro  **Breakpoint**.
 - El botón  **Stop execution** permite detener la ejecución del programa.