

# Codificación y depuración en Java



Curso 2023/24

# Java

## Codificación

# Consejos y/o conceptos iniciales

- Organización en carpetas y workspaces.
- Creación de proyectos.
- Hilos en ejecución.
- Paquetes con nombre.
- Diferencia entre argumento y parámetro.
- Qué es la cabecera de un método.
- Ámbito de una variable y de un atributo.
- Las clases se llaman igual que sus ficheros.
- El método main es el punto de entrada de todo programa.
- Jerga: tamaño  $n$ ,  $n$ -ésimo, posición  $i$ -ésima,  $j$ -ésima,  $k$ -ésima, etc.
- Hacer lo mismo que en el tutorial en vídeo con el fichero Tema 1, Calculadora.txt.

# Codificación

Una vez realizado el diseño, entonces es cuando se comienza el proceso de codificación. En esta etapa, el **programador recibe las especificaciones del diseño** y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación.

En cualquier proyecto de desarrollo que pretenda tener una cierta calidad, deben existir unas **normas** claras y concisas sobre la codificación y el estilo. Estas normas **facilitarán** las tareas de corrección y mantenimiento de los programas, sobre todo cuando son realizadas por personas que no los han desarrollado.

<https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html>

# Codificación en Java: consideraciones generales I

A continuación vamos a ver una gran cantidad de normas que cualquier desarrollador Java debería seguir.

- ❑ La extensión para los ficheros de código fuente es `.java` y para los ficheros compilados es `.class`.
- ❑ Es aconsejable que cada fichero contenga únicamente una clase java.
- ❑ Se debe declarar una sola variable por línea.
- ❑ Se deben inicializar las variables en el momento que se declaran.
- ❑ No se ponen espacios entre el nombre del método y el paréntesis de apertura `"("`.
- ❑ La llave de apertura `"{"` se coloca en la misma línea que el nombre del método.
- ❑ La llave de cierre `"}"` se coloca en una línea aparte pero a la misma altura que el inicio del bloque.

# Codificación en Java: consideraciones generales II

- ❑ Los métodos se separan por una línea en blanco.
- ❑ Lo primero que debe aparecer es información acerca del proyecto, seguido de la importación de paquetes.
- ❑ Los métodos deben ser verbos. Si tienen una sola palabra, todo en minúscula. Si tienen varias, la primera en minúscula y las demás en mayúscula.
- ❑ El nombramiento de variables, clases e interfaces sigue el mismo principio.
- ❑ Las constantes se escriben todo en mayúscula, separando las palabras con un guión bajo.
- ❑ Escribe los paréntesis en las condiciones aunque no sean necesarios.

```
if (a == b && c == d)           // Incorrecto
if ((a == b) && (c == d))       // Correcto
```

# Codificación en Java: consideraciones generales III

- ❑ Cuida la **tabulación** y asegúrate que las **llaves de apertura y cierre estén bien puestas** y en su nivel de tabulación correspondiente.
- ❑ Cuidado con los **espacios en blanco** al escribir código.

```
a=(b+c)*d           // Incorrecto  
a = (b + c) * d;    // Correcto
```

```
while(true){...}     // Incorrecto  
while (true) {...}   // Correcto
```

```
for(i=0;i<n;i++)      // Incorrecto  
for (i = 0; i < n; i++) // Correcto
```

# Codificación en Java: comentarios y Javadoc

Existen tres tipos de comentarios en Java:

- Comentarios de bloque

```
/*  
 * Esto es un comentario de bloque  
*/
```

- Comentarios de línea

```
/* Esto es un comentario de línea */
```

- Comentarios cortos

```
// Esto es un comentario corto
```



# Codificación en Java: comentarios y Javadoc

Esta herramienta se utiliza para **documentar** las clases de Java. Los IDE como Eclipse, IntelliJIDEA o NetBeans tienen una herramienta Javadoc incorporada para generar documentación HTML.

La documentación HTML generada por Javadoc es documentación de API. Analiza las declaraciones y genera un conjunto de **archivos fuente**. El archivo fuente describe campos, métodos, constructores y clases.

# Codificación en Java: comentarios y Javadoc

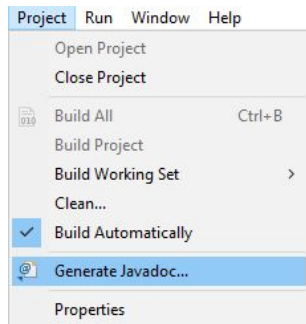
Los comentarios pueden incluir **etiquetas**, las cuales ayudan al programador a conocer el código de la aplicación. Algunas de las más comunes son:

Etiqueta	Descripción	Se aplica a
<code>@return</code>	Descripción del valor de retorno	Método
<code>@autor</code>	Indica el autor	Clase, interface o enum
<code>@version</code>	Versión del software	Clase, interface o enum
<code>@param</code>	Descripción del parámetro	Método
<code>@exception</code>	Descripción de la excepción que lanza el método	Método

# Codificación en Java: comentarios y Javadoc

Etiqueta	Descripción	Se aplica a
<code>@throws</code>	Igual que la etiqueta <code>@exception</code>	Método
<code>@ver</code>	Especifica referencias a otra documentación	Clase, interface, enum, campo o método
<code>@description</code>	Especifica si el método está desactualizado	Clase, interface, enum, campo o método

En Eclipse, el Javadoc se genera seleccionando el proyecto desde la raíz > Project > Generate Javadoc.



# Codificación en Java: comentarios y Javadoc

```
/**
 * Clase Jugador
 * @author J.C.
 */
public class Jugador {
    String nombre;
    Integer puntuacion;

    /**
     * Constructora
     * @param _nombre
     * @param _puntuacion
     */
    public Jugador(String _nombre, Integer _puntuacion) {
        this.nombre = _nombre;
        this.puntuacion = _puntuacion;
    }

    /**
     * Retorna el nombre de jugador
     * @return
     */
    public String getNombre() {
        return this.nombre;
    }
}
```

```
/**
 * Modifica el nombre del jugador
 * @param _nombre
 */
public void setNombre(String _nombre) {
    this.nombre = _nombre;
}

/**
 * Retorna la puntuacion del jugador
 * @return
 */
public Integer getPuntuacion() {
    return this.puntuacion;
}

/**
 * Modifica la puntuacion del jugador
 * @param _puntuacion
 */
public void setPuntuacion(Integer _puntuacion) {
    this.puntuacion = _puntuacion;
}
}
```

# Codificación en Java

Este conjunto de normas nos ayudan a **establecer un marco** a través del cual los programadores podemos escribir código al unísono. Seguir estas reglas de codificación sería el equivalente a no cometer faltas de ortografía en la escritura regular.

**Tarea:** Observa las clases Main, Jugador y Record. Identifica los fallos de codificación y corrígelos según las normas vistas anteriormente.

**Tarea:** Busca algún proyecto Java en Internet (o que ya tengas por ahí) de no más de tres clases y crea su Javadoc. Consulta la página HTML generada y compara la información que aparece con la que hay en el código.

# Lenguajes de programación

Clasificación y características

# Definición

Podríamos decir que un lenguaje de programación es un conjunto de caracteres, reglas para la combinación de esos caracteres y reglas que definen sus efectos cuando son ejecutados por un ordenador. Consta de los siguientes elementos:

- ❑ **Un alfabeto (léxico):** formado por el conjunto de símbolos permitidos.
- ❑ **Una sintaxis:** reglas que indican cómo realizar construcciones con los símbolos.
- ❑ **Una semántica:** reglas que determinan el significado de cualquier construcción en el lenguaje.

# Clasificación y características

Los lenguajes se clasifican atendiendo a varios criterios.

Nivel de abstracción	Forma de ejecución	Enfoque de programación
Bajo nivel	Compilados	Imperativos
Nivel medio	Interpretados	Funcionales
Alto nivel		Lógicos
		Estructurados
		Orientados a objetos



# Nivel de abstracción

Si comenzamos por los lenguajes de bajo nivel, encontramos en primer lugar el **lenguaje máquina**. Este lenguaje es el que entiende directamente el sistema y está compuesto por unos y ceros. A continuación encontraríamos el **lenguaje ensamblador**, un lenguaje realmente difícil de aprender y particular de cada procesador. Este último necesita ser traducido a lenguaje máquina para poder ejecutarse.

C003 86 13	INITA	LDA A	#RESETA	RESET ACIA
C005 B7 80 04		STA A	ACIA	
C008 86 11		LDA A	#CTLREG	SET 8 BITS AND 2 STOP
C00A B7 80 04		STA A	ACIA	
C00D 7E C0 F1		JMP	SIGNON	GO TO START OF MONITOR

# Nivel de abstracción

A continuación encontramos **lenguajes de nivel medio**, que son aquellos que tienen **ciertas características** que los acercan a los lenguajes de **bajo nivel** pero, al mismo tiempo, también tienen características de lenguajes de **alto nivel**. Podríamos destacar **arduino y C**.

```
/* Programa: Hola mundo */

#include <conio.h>
#include <stdio.h>

int main()
{
    printf( "Hola mundo." );

    getch(); /* Pausa */

    return 0;
}
```

# Nivel de abstracción

Por último encontraríamos los lenguajes de alto nivel, que suelen ser los más fáciles de aprender. Para ejecutarlos necesitamos un **compilador o programa intérprete** que traduzca las instrucciones escritas en el código. Estos lenguajes son independientes del hardware.

```
<html>
  <head>
    <title>Prueba de PHP</title>
  </head>
  <body>
    <?php echo '<p>Hola Mundo</p>'; ?>
  </body>
</html>
```

# Forma de ejecución

En primer lugar tendríamos los lenguajes compilados. Un compilador **no es más que un programa** que puede leer el programa escrito en un determinado lenguaje (un lenguaje fuente) y **traducirlo** a un lenguaje de sentido que será ejecutado en la máquina.

Por otro lado tenemos los lenguajes interpretados, los cuales **no producen** un programa destino como proceso de traducción, sino que el intérprete nos permite ejecutar **directamente** las operaciones especificadas en el programa fuente.

# Enfoque de programación

Por un lado tenemos los lenguajes imperativos, que son aquellos que básicamente **realizan asignaciones** y donde las estructuras de control permiten establecer el orden de ejecución de las instrucciones.

En segundo lugar tenemos los lenguajes funcionales que están basados en el concepto matemático de **función**. Por ejemplo, en este tipo de lenguajes no existe la asignación.

En tercer lugar encontraríamos los lenguajes lógicos. En este tipo de lenguajes un **cálculo** es el proceso de encontrar qué elementos de un dominio **cumplen** una determinada relación definida sobre dicho dominio.

# Enfoque de programación

En penúltimo y último lugar encontramos los lenguajes de **programación estructurados** y **lenguajes orientados a objetos**. Podríamos decir que ambos son un subconjunto de los lenguajes imperativos.

El primero consiste en un **conjunto de instrucciones** que pueden ser leídas de principio a fin sin perder la continuidad de lo que hace. Con el tiempo surgió la división por módulos, que actualmente se conoce como **programación modular**. Esta división aporta claridad, reducción de costes al abordar problemas más pequeños y simultaneidad.

# Enfoque de programación

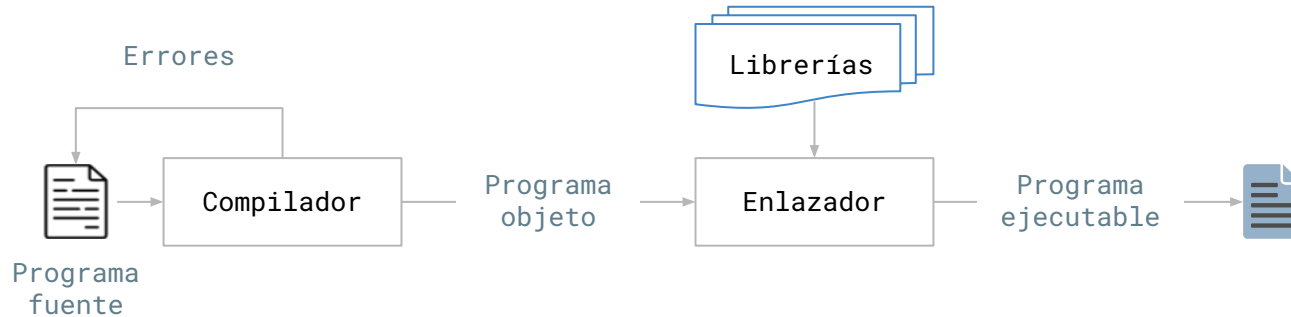
En el **segundo**, el programa está **compuesto por un conjunto de objetos**, no por un conjunto de instrucciones o módulos. Estos objetos **constan de una estructura de datos y de una colección de métodos y operaciones que manipulan esos datos.**

```
1. #include <iostream>
2. using namespace std;
3.
4. int main() {
5.     cout << "Hola mundo, bienvenidos!" << endl;
6.     return 0;
7. }
```

**Tarea:** Investiga sobre los lenguajes de programación existentes y crea una tabla para clasificarlos según sus características (con 10 lenguajes sería suficiente).

# Compilación

En este proceso intervienen dos programas: el **compilador** y el **enlazador**. Si el compilador detecta **algún error**, entonces no se genera el código objeto y será necesario **modificar** el programa fuente y pasarlo nuevamente al compilador.



Durante la segunda etapa, el programa enlazador **inserta** en el código objeto las funciones de librería necesarias para producir el programa ejecutable.



# Java

## Principales componentes

# ¿Qué es Java?

Java es un lenguaje de **alto nivel**, orientado a objetos y conocido principalmente por su simplicidad, independencia de la plataforma de ejecución y versatilidad.

Fue creado por James Gosling y su equipo en Sun Microsystems (hoy en día controlado por **Oracle**) a mediados de los años 90'.



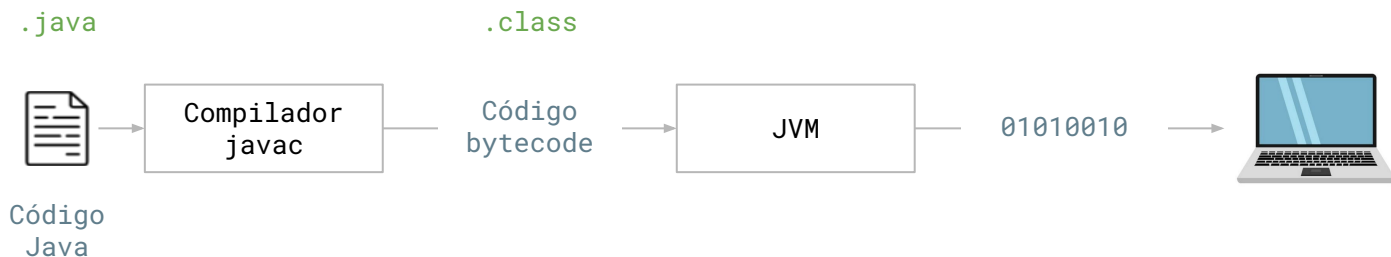
# Java Virtual Machine (JVM)

Los programas compilados en lenguaje Java se pueden ejecutar en cualquier SO. Esto es debido a que el código generado por el compilador **no lo ejecuta el procesador** del ordenador, sino que lo ejecuta la Máquina Virtual de Java. Sus principales tareas son las siguientes:

- ❑ Reserva espacio en memoria para los objetos creados y libera espacio no utilizado
- ❑ Se comunica con el sistema huésped para ciertas funciones tales como controlar el acceso a los dispositivos hardware
- ❑ Vigila el cumplimiento de las normas de seguridad de las aplicaciones Java

# Java Virtual Machine (JVM)

Lo que hace básicamente es **coger el código fuente** del programa, que está escrito en **texto plano** con la extensión **.java**. La **compilación** de este fichero (siempre que no haya errores) **da lugar a otro fichero .class** que contiene **código muy cercano al lenguaje máquina** pero que **al mismo tiempo es independiente del SO** en el que se ejecuta. Este código es conocido como **bytecode**.



# Java Development Kit (JDK)

El JDK es un conjunto de herramientas y utilidades proporcionados por Oracle que permiten a los desarrolladores crear, compilar y ejecutar aplicaciones programadas en Java.

Este kit incluye **todos los componentes necesarios** para escribir, testear y desplegar programas Java.



# Java Runtime Environment (JRE)

El JRE es un paquete de software que permiten ejecutar aplicaciones Java en tu ordenador. Incluye la Java Virtual Machine y las librerías esenciales que se necesitan para ejecutar programas Java.



**Java Runtime  
Environment**

# Entornos de Desarrollo

Introducción y componentes  
de un entorno

# Introducción

Los IDE (Integrated Development Environment) son aplicaciones informáticas que están compuestas por una serie de **componentes** que van a facilitar las tareas de programación y hacen que el rendimiento del desarrollo de aplicaciones sea mayor.





# Componentes

Un IDE normalmente consiste en un editor de texto donde escribir el código resaltado de sintaxis y corrector automático, un compilador y/o intérprete, un depurador, control de versiones, constructor de interfaz gráfica (GUI) y muchas otras funcionalidades.

- ❑ **Editor de texto:** es la parte que nos permite escribir el código fuente del programa. Ofrece funcionalidades tales como copiar, cortar, pegar o buscar. Es capaz de reconocer el lenguaje y su sintaxis.
- ❑ **Compilador:** es el encargado de traducir el código fuente de alto nivel a un programa escrito en lenguaje de bajo nivel, llamado lenguaje máquina.

# Componentes

- ❑ **Intérprete:** a diferencia de un compilador, el intérprete sólo realiza la traducción a medida que se van ejecutando las instrucciones. Son más lentos debido a la necesidad de traducir el programa mientras se ejecuta.
- ❑ **Debugger:** depura y limpia los errores en el programa fuente. Examina paso a paso el contenido de las variables y los valores retornados por las funciones. Admite puntos de ruptura para detener el código en la instrucción deseada.
- ❑ **Constructor de interfaz gráfica:** simplifica la construcción de interfaces permitiendo al programador arrastrar los elementos.
- ❑ **Control de versiones:** controlan los cambios producidos en la aplicación.

# Eclipse

Instalación, plugins, triggers y syntax  
coloring

# Instalación

Eclipse es una plataforma compuesta por un conjunto de herramientas de programación de **código abierto** y **multiplataforma**.

Fue desarrollado originalmente por IBM, pero ahora lo desarrolla la Fundación Eclipse, una organización **sin ánimo de lucro** que fomenta una comunidad de código abierto.


Se da por hecho que ya se ha realizado la instalación en la asignatura de Programación.



# Plugins

Mediante el uso de plugins en Eclipse nos será posible diseñar la interfaz de usuario con un tema oscuro. También permite acelerar el desarrollo con marcos o la integración con sistemas de control de versiones, así como poder detectar problemas de calidad, como el análisis de código estático a medida que vamos escribiendo.

Para instalar un plugin nos vamos a la pestaña de **Help** y seleccionamos **Eclipse Marketplace**.



### Darkest Dark Theme with DevStyle 2022.6.13a

Darkest Dark theme from DevStyle - a free plugin providing an enhanced set of experiences for Eclipse. Included: Darkest Dark theme - #1 in the Marketplace: ... [more info](#)

by [Genuitec, LLC](#), Commercial - Free  
[dark theme Darkest Dark Genuitec](#)

★ 4844

Installs: **2.01M** (18,803 last month)

Install

# Plugins

Uno de los más utilizados es EGit, que permite descargar código desde **Github** y proporciona una integración de Git con Eclipse.

TestNG es muy útil para realizar tests **JUnit** (se verán en el Tema 3). Puedes ejecutar bancos de prueba, facilita la creación de tests en grupos o métodos individuales.



## TestNG for Eclipse

This plug-in lets you run your TestNG tests from Eclipse. You can run suites, groups or individual methods. Errors are reported in a separate tab that lets you... [more info](#)

by [Cédric Beust](#), Apache 2.0

[testng](#) [junit](#) [testing](#) [unit](#) [integration](#) [functional](#) [selenium](#)

★ 696



Installs: **1.49M** (28,385 last month)

Install



## EGit - Git Integration for Eclipse 6.0.0

EGit is the Git integration for Eclipse. Git is a distributed versioning system, which means every developer has a full copy of all history of every revision of... [more info](#)

by [Eclipse.org](#), EPL

[egit](#) [jgit](#) [git](#) [dvcs](#) [scm](#)

★ 1780



Installs: **619K** (991 last month)

Installed

# Plugins

Otro plugin realmente útil cuando utilizamos el popular framework de Java **Spring**, es **Spring Tools**. Este nos ayuda a crear fácilmente proyectos e integrarlos, facilitando el desarrollo.

También cabe destacar **Maven**, que proporciona una integración con **proyectos Maven**, los cuales son usados en Ingeniería de Software para la construcción y gestión de software.

## Spring Tools 4 (aka Spring Tool Suite 4) 4.15.3.RELEASE



Spring Tools 4 is the next generation of Spring Boot tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support... [more info](#)

by [VMware](#), EPL

[spring](#) [Spring IDE](#) [Cloud](#) [Spring Tool Suite](#) [STS](#)

★ 3634



Installs: **2.31M** (28,597 last month)

Install

## Eclipse m2e - Maven support in Eclipse IDE Latest



M2Eclipse provides tight integration for Apache Maven into the Eclipse IDE with the following features: Rich editor for pom.xml files Launching Maven builds... [more info](#)

by [Eclipse m2e project](#), EPL 2.0

[nature](#) [org.eclipse.m2e.core.maven2Nature](#)  
[fileExtension](#) [pom.xml](#)

★ 124



Installs: **7.52K** (430 last month)

Installed

# Triggers

Cuando estamos escribiendo código, es común utilizar la función **autocompletado**. En **Java**, normalmente esta utilidad es lanzada cuando escribimos **“.”** a la hora de acceder a **métodos**, por ejemplo. Sin embargo, **existe la posibilidad de lanzar esa utilidad para cada una de las teclas que pulsemos**. De esta forma, las sugerencias de código aparecerán mucho antes.

Para ello pulsamos sobre la pestaña **Window** y después en **Preferences**. Desplegamos **Java**, luego desplegamos **Editor** y por último seleccionamos **Content Assist**. En el recuadro “Auto activation triggers for Java” podemos escribir **QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm**. **Ahora, por cada pulsación, se lanzará el trigger**. Cabe destacar que esta utilidad puede llegar a ser molesta en ciertas ocasiones.



# Beautiful syntax

Una de las partes que más gusta a los programadores tener perfectamente establecida es todo lo referente a la sintaxis y a cómo se ve el código. Algunos prefieren temas oscuros, otros claros o grises. Coloreado muy destacado o más leve.

Para configurar los colores de nuestro código Java debemos ir a **Window** y pulsar nuevamente en **Preferences**. Luego nos vamos a **Java**, pulsamos en **Editor** y seleccionamos **Syntax Coloring**. Aquí podemos modificar cualquier estilo y color sobre la sintaxis de nuestro código.

# Beautiful syntax

Aquí tenéis una comparación entre el estilo por defecto y mi configuración.

```
/**
 * This is about <code>ClassName</code>.
 * {@link com.yourCompany.aPackage.Interface}
 * @author author
 * @deprecated use <code>OtherClass</code>
 */
public class ClassName<E> extends AnyClass implements InterfaceName
    enum Color { RED, GREEN, BLUE };
    record Point(int x, int y) {};
    /* This comment may span multiple lines. */
    static Object staticField;
    // This comment may span only this line
    private E field;
    private AbstractClassName field2;
    // TASK: refactor
    @SuppressWarnings(value="all")
    public int foo(Integer parameter) {
        abstractMethod(inheritedField);
        int local= 42*hashCode();
        var varLocal= local;
        staticMethod();
        return bar(varLocal) + parameter;
    }
}
```

```
/**
 * This is about <code>ClassName</code>.
 * {@link com.yourCompany.aPackage.Interface}
 * @author author
 * @deprecated use <code>OtherClass</code>
 */
public class ClassName<E> extends AnyClass implements InterfaceName
    enum Color { RED, GREEN, BLUE };
    record Point(int x, int y) {};
    /* This comment may span multiple lines. */
    static Object staticField;
    // This comment may span only this line
    private E field;
    private AbstractClassName field2;
    // TASK: refactor
    @SuppressWarnings(value="all")
    public int foo(Integer parameter) {
        abstractMethod(inheritedField);
        int local= 42*hashCode();
        var varLocal= local;
        staticMethod();
        return bar(varLocal) + parameter;
    }
}
```

# Eclipse

Errores y debugger

# Introducción

Ser eficiente a la hora de resolver problemas de código es una de las tareas más importantes para los desarrolladores.

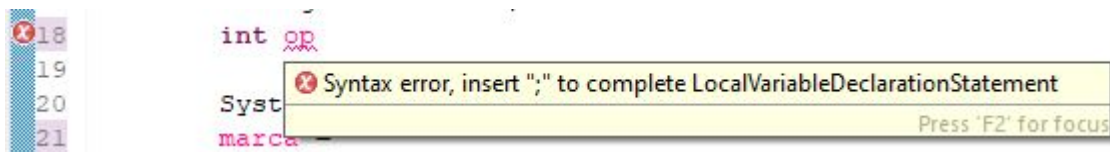
El uso de herramientas que nos permitan la elaboración de pruebas es un arma indispensable en nuestro día a día y nos ahorrará muchas horas de trabajo en vano.

Al desarrollar programas, podríamos decir que básicamente nos encontramos con dos tipos de errores.

- ❑ Errores en tiempo de compilación
- ❑ Errores en tiempo de ejecución

# Errores en tiempo de compilación

Normalmente son **fáciles de corregir**, ya que el IDE es capaz de darse cuenta de si estamos cometiendo algún error de sintaxis. Además, el entorno suele **proporcionar** información útil sobre cómo resolver el problema.



# Warnings en tiempo de compilación

No son errores como tal, así que no tiene sentido crear una nueva categoría dentro de los mismos. Estos warnings son detectados por el IDE que, al igual que los errores, proporciona una posible solución. Los warnings **no detienen** la ejecución del programa, pero tampoco deben ser **ignorados**.

Description

▼ ⚠ Warnings (2 items)

- ⚠ Project 'Coche' has no explicit encoding set
- ⚠ Resource leak: 'sc' is never closed

```
13      */
14      public static void main(String[] args) {
15          Resource leak: 'sc' is never closed; Scanner(System.in);
16          String marca = "";
```

# Errores en tiempo de ejecución

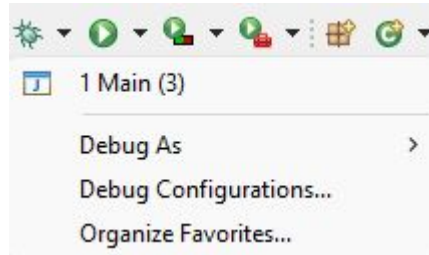
Este tipo de errores son **más complejos** de detectar y resolver. Se produce cuando la ejecución del programa retorna **errores inesperados**, los cuales son comúnmente conocidos como **bugs**.

En Java, estos errores también son conocidos como **excepciones**. Algunos ejemplos son:

- ❑ `NullPointerException`
- ❑ `DivisionByZero`
- ❑ `EOFException` (que hereda de `IOException`)

# Debugger

Los entornos incorporan una herramienta conocida como debugger, la cual nos permite **analizar** el código mientras se ejecuta. En Eclipse podemos lanzar el debugger y cambiar de perspectiva.





# Debugger

Los puntos de ruptura se utilizan para **inspeccionar** el código en una determinada instrucción.

En la página **143** y **144** del libro de Entornos aparece una leyenda con los símbolos que comúnmente son utilizados en la etapa de depuración.

# Debugger

Con el ejemplo anterior, vamos a explorar el entorno y ver algunas de las opciones típicamente usadas.

- ❑ Ejecutar el código instrucción a instrucción, método a método o hasta que encontramos un nuevo breakpoint.
- ❑ Observar los paneles outline, breakpoints, variables, debug y problems.
- ❑ Ver los hilos de ejecución activos.

FW[ /8' fiZBSa SbSeal ħ` WS ħ` WZE[ geS ħTmSe ħWS\_ Tl'e` WUoVYa VWSbSe SbSeaz  
FWahW/8( fiZBSa SbSeal ħ` WS ħ` WbWa ea^a Wfg UoVYaž@a hSS 'Se ħTmSeUa\_ a WS` fWladž  
FWMgd /8) fiZ6SeS'fae WfdW\_ éfaVaežHS VWWVW [Ua VW\_ éfaVa ZSeS WdMgd ž

# Codificación y depuración en Java



Curso 2023/24