Sesión
El entorno de Programación Eclipse

En el laboratorio de Complementos de programación utilizaremos Eclipse, un entorno gráfico de desarrollo para programar en Java. En esta sesión nos familiarizamos con el entorno y además crearemos y ejecutaremos programas simples en Java.

1. Introducción a Eclipse

Eclipse es una herramienta que integra diferentes componentes permitiendo realizar las tareas de edición, compilación, ejecución y depuración de programes de forma simple y amigable. A estas aplicaciones se las llama *Entornos Integrados de Desarrollo* (en inglés IDE, Integrated Development Environment). Aunque, con los plug-ins adecuados, soporta cualquier tipo de desarrollo de software mayoritariamente se utiliza para desarrollar aplicaciones en Java; probablemente porque la plataforma Eclipse SDK incluye el Java Development Toolkit (JDT) un IDE para programar en Java. Otras aplicaciones, no incluidas en la distribución estándar de Eclipse, se pueden añadir en forma de plug-ins, y son reconocidas automáticamente por la plataforma. Es importante remarcar que tras el JDT se ejecutan las herramientas del JDK (Java Development Kit) y por lo tanto, todo lo que vemos se puede también ejecutar mediante comandos en una ventana de DOS o en un terminal de Unix o Linux.

Eclipse es una aplicación desarrollada en Java y por lo tanto para utilizarla hemos de tener instalada la versión 1.3 del JDK o posterior. Se recomienda tener instalada la última versión.

Podéis encontrar información sobre la herramienta en:

- `http://www.eclipse.org/`: creadores de la herramientas.
- `http://help.eclipse.org/help21/index.jsp`: tutoriales sobre los distintos módulos de la herramienta. Estos tutoriales están incluidos en el help de Eclipse y se pueden consultar seleccionando la opción Help→Help Contents.

Bibliografía: *The Java Developer's Guide to Eclipse* y *Eclipse in Action*.

2. Arrancando Eclipse

Al arrancar la aplicación Eclipse[1] aparece la ventana mostrada en la Fig. 1 que ameniza la espera mientras se carga el entorno.

Figura 1: Pantalla Inicial

A continuación aparece un cuadro de diálogo (ver Fig. 2) que nos pide la carpeta en que crear los proyectos (workspace). Por defecto la carpeta es *HOME_ECLIPSE/workspace/*[2]. Seleccionad la carpeta en la que queréis almacenar vuestros proyectos; si la carpeta no existe se creará automáticamente.

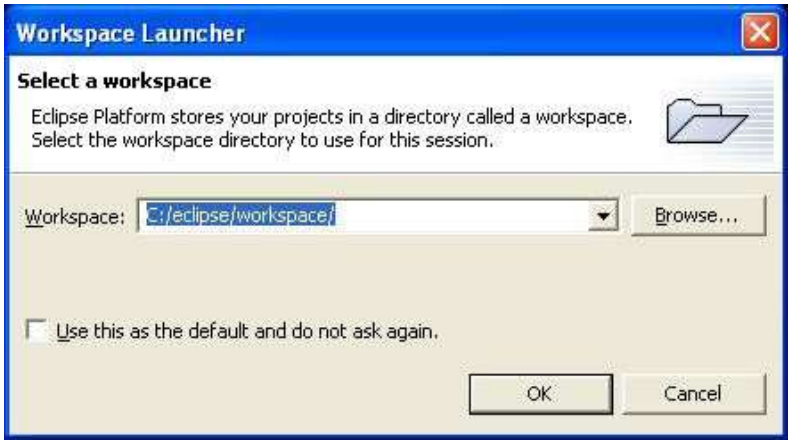

Figura 2: Cuadro de diálogo para escoger el directorio de trabajo.

Una vez seleccionada la carpeta de trabajo aparece la pantalla de bienvenida mostrada en la Fig. 3.

[1] Las descripciones y las pantallas corresponden a la versión 3.0 de Eclipse ejecutándose en Windows XP.

[2] *HOME_ECLIPSE* es el path de donde está instalado el eclipse (en el ejemplo mostrado en la Fig. 1.2 Eclipse está instalado en C:/eclipse).

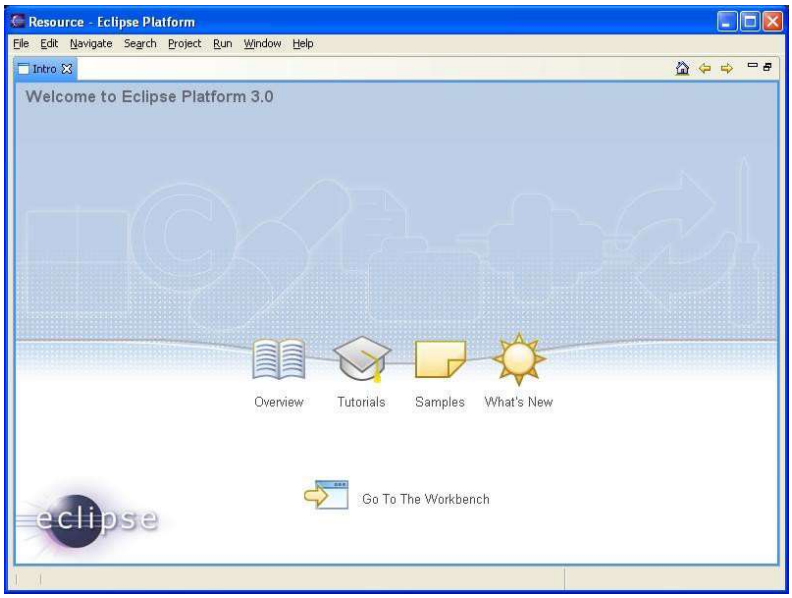

Figura 3: Pantalla de bienvenida.

Clicad el icono *Go To the Workbench* para entrar en la ventana de trabajo, Fig. 4. El workbench de Eclipse ofrece distintas perspectivas para realizar distintas tareas, como pueden ser escribir o depurar un programa en Java. La primera vez que entremos en Eclipse este presenta la perspectiva de recursos (*Resource perspective* - ver Fig. 4 -). Esta es una perspectiva de propósito general para crear, ver y manipular todos los tipos de recursos.

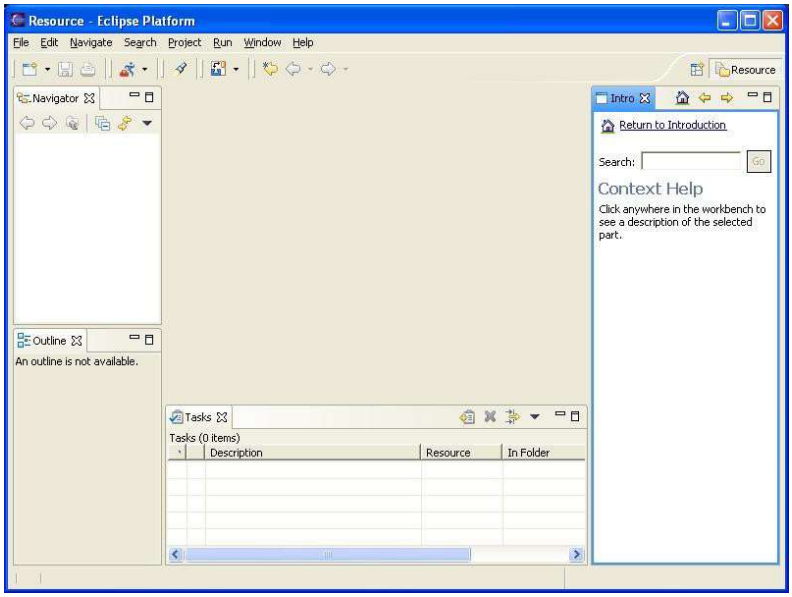

Figura 4: Perspectiva de Recursos

En nuestro caso trabajaremos con una perspectiva específica de Java (*Java perspective*). Para cambiar a esta perspectiva seleccionad *Window→Open perspective→Java*, como se muestra en

la Fig. 5, con lo que nuestra ventana tendrá el aspecto mostrado en la Fig. 6.

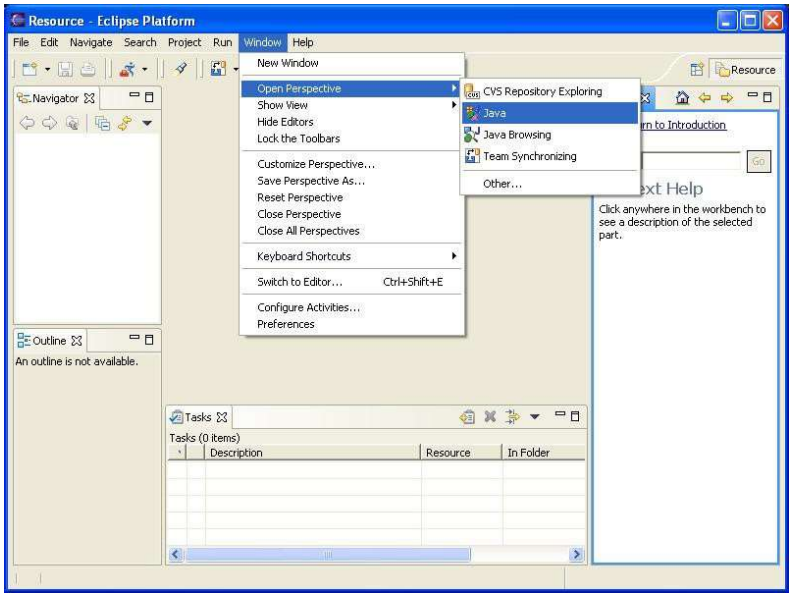

Figura 5: Abriendo una nueva perspectiva

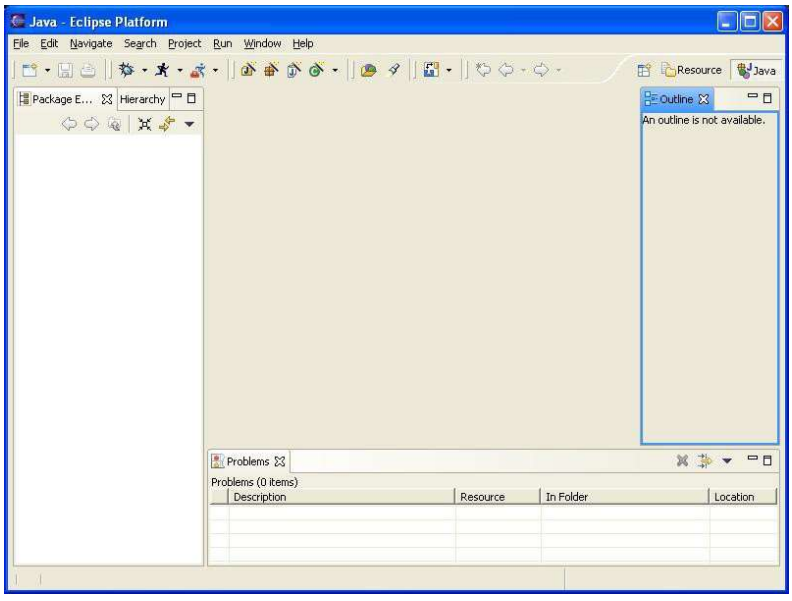

Figura 6: Perspectiva Java.

3. Creación y ejecución de un programa

Veamos como crear y ejecutar un programa en Java utilizando el clásico ”Hola, mundo”. Antes de hacer cualquier otra cosa, en Eclipse, como por ejemplo escribir un programa en Java,

necesitamos crear un proyecto (*project* en inglés).

3.1. Crear un proyecto

Para crear un proyecto seguid los pasos siguientes:

1. Seleccionad la opción *File→New→Project*, tal como muestra la Fig. 7.

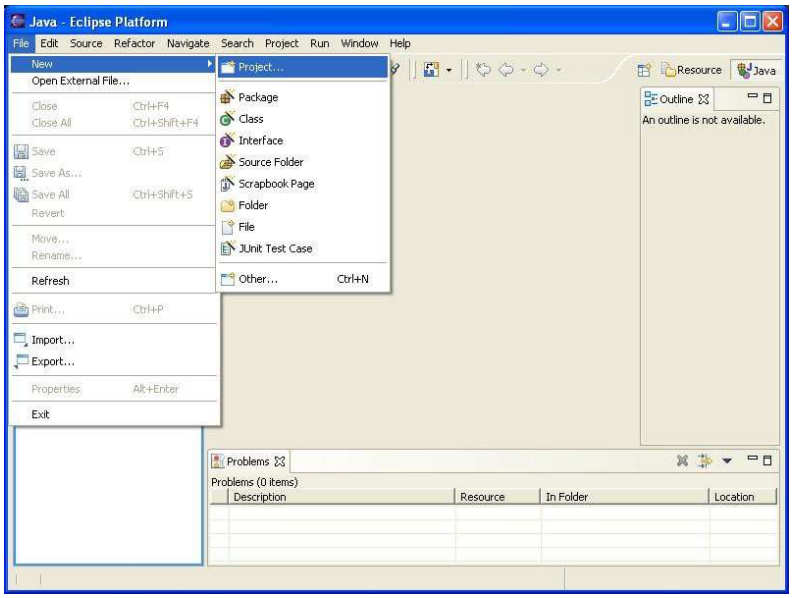

Figura 7: Creando un nuevo proyecto.

2. La pantalla de diálogo *New Project*, mostrada en la Fig. 8, os ofrece dos opciones[3]: Java Project y Plug-in Project; seleccionad la opción Java Project y apretad el botón Next.

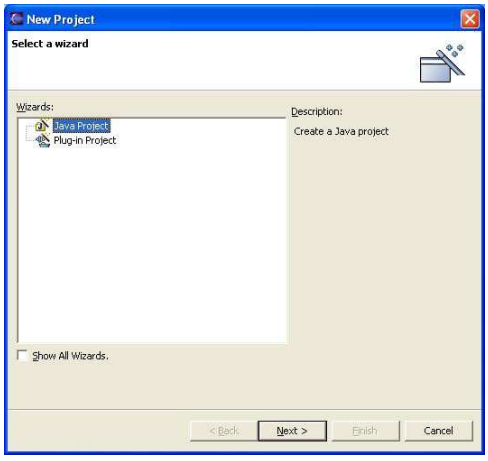

Figura 8: Cuadro de diálogo *New Project*.

[3]Si hay instalados otros plug-ins de Java (como por ejemplo EJBs o servlets) también aparecerán como opciones.

3. A continuación aparece una pantalla de diálogo que pide el nombre del proyecto (ver Fig. 9), en nuestro caso podemos introducir como nombre del proyecto `HolaMundo` (atención en la Fig. 9 lo tenéis escrito en catalán, pero mejor no mezclar idiomas).

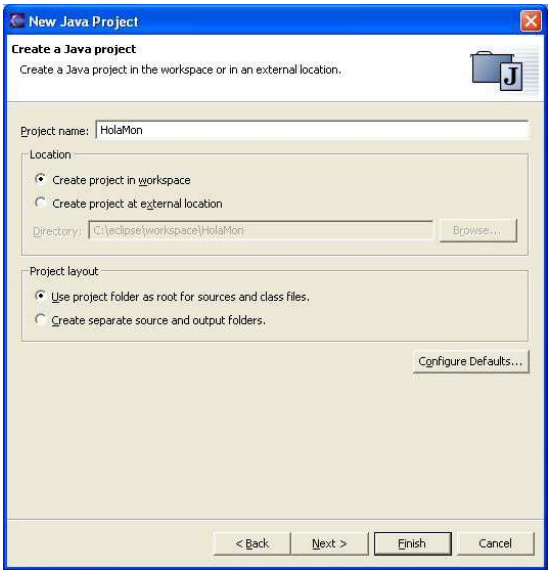

Figura 9: Cuadro de diálogo *Project Name*.

4. Apretad el botón Finish y se creará el proyecto, la ventana *package explorer* de la pantalla principal mostrará la carpeta del nuevo proyecto tal como se puede ver en la Fig. 10.

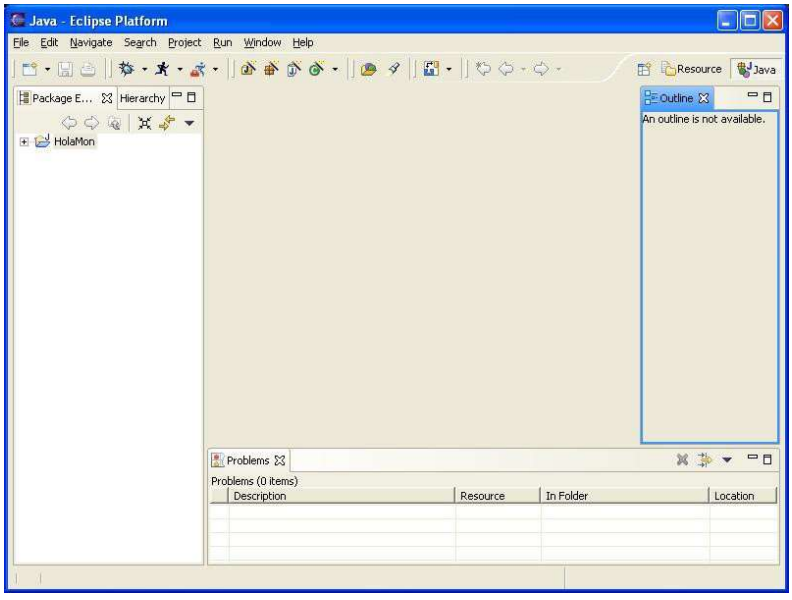

Figura 10: Pantalla principal después de crear el proyecto.

3.2. Añadir una clase a un proyecto

Una vez creado el nuevo proyecto podemos añadirle clases. Comencemos por añadir la clase correspondiente al programa `HolaMundo`, para hacerlo seguid los pasos siguientes:

1. Seleccionad la opción *File→New→Class*, tal como muestra la Fig. 11.

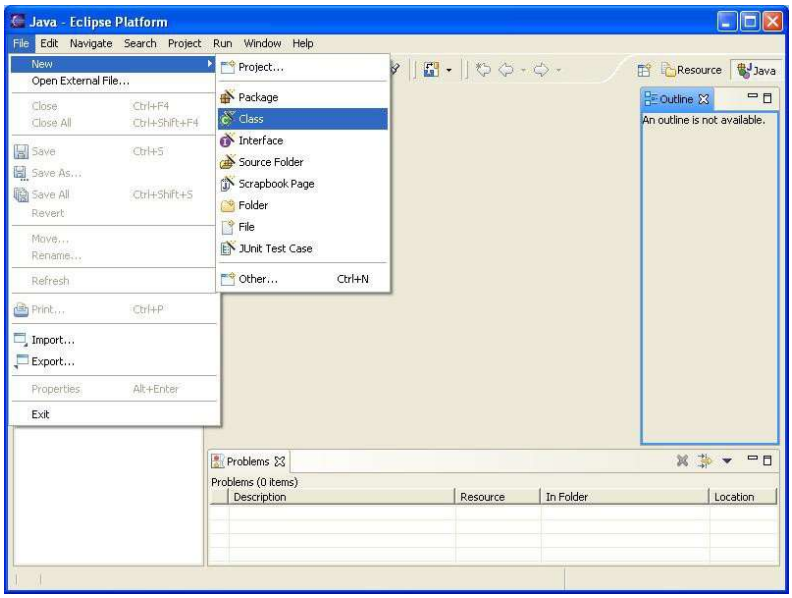

Figura 11: Creando una clase.

2. La pantalla de diálogo *New Java Class*, mostrada en la Fig. 12, os pedirá el nombre de la clase, en nuestro caso introducimos como nombre de la clase ”HolaMundo”.

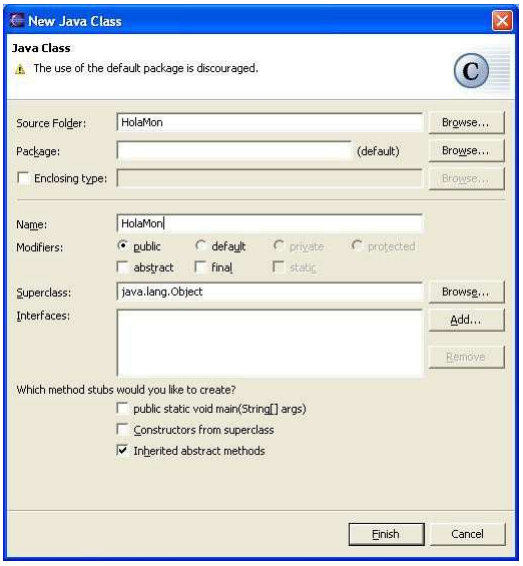

Figura 12: Cuadro de diálogo *New Java Class*.

3. Apretad el botón Finish y se creará la clase. La pantalla principal mostrará el aspecto de la Fig. 13, en el que en la ventana *package explorer* veréis que se ha añadido el fichero `HolaMundo.java` al proyecto y que la ventana de edición muestra el contenido de este fichero con el código Java de la clase generado per defecto.

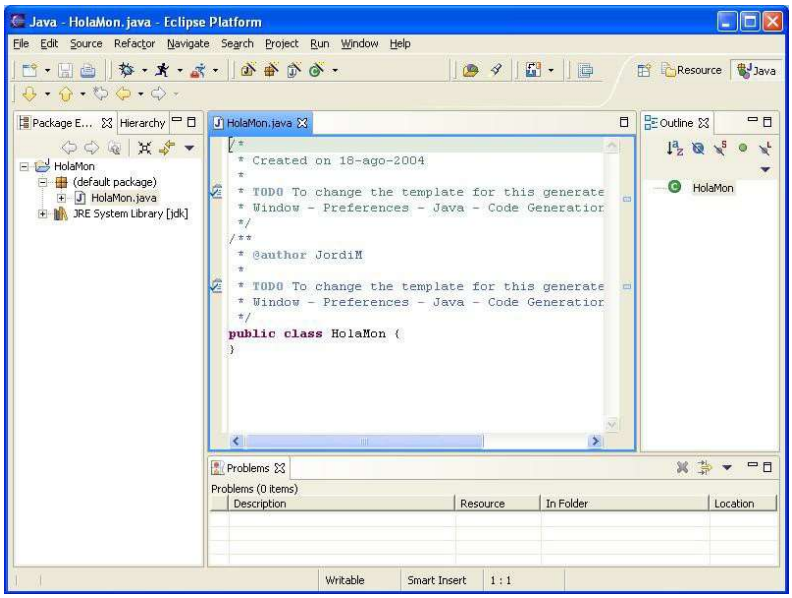

Figura 13: Pantalla principal después de crear la clase.

Ahora podemos añadir el código correspondiente al programa `HolaMundo`, que se muestra a continuación:

```
public class HolaMundo {

        public static void main(String[] args) {
                System.out.println("Hola Mundo!");
        }
}
```

Una vez añadido el código grabamos el fichero. Al grabar el fichero Eclipse compila automáticamente el programa y avisa de los errores. En este caso no hay ninguno.

3.3. Ejecución de un programa

Para ejecutar el programa, seleccionad la opción *Run→Run As→Java Application*, como se muestra en la Fig. 14. Automáticamente se llama a la máquina virtual de Java y se ejecuta la aplicación. El resultado de la ejecución se puede ver en la ventana consola de la pantalla principal (ver Fig. 15).

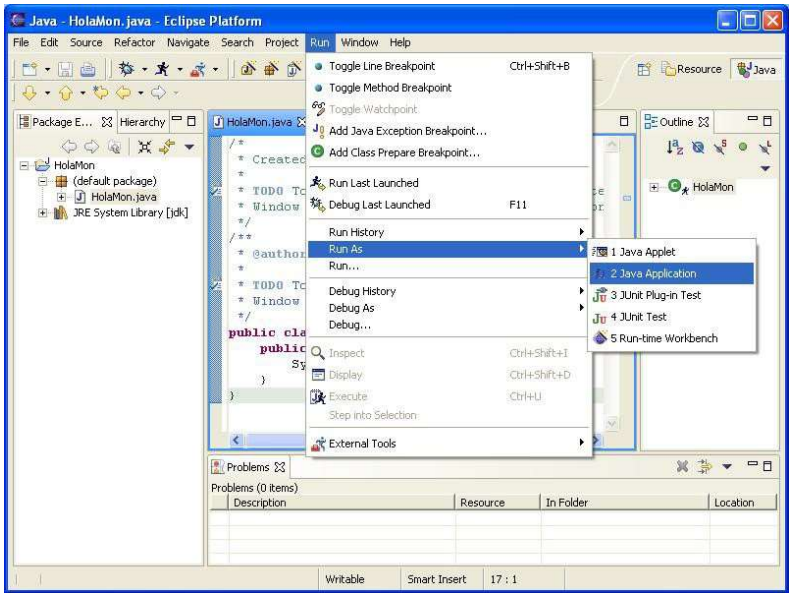

Figura 14: Ejecutando el programa.

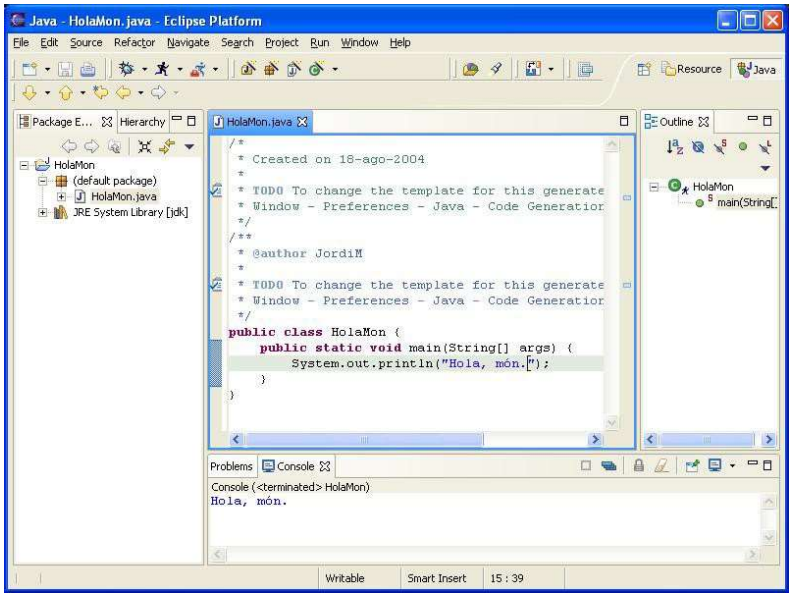

Figura 15: Resultado de la ejecución

4. Ejercicios

1. (Leer y Sumar) Cread un nuevo proyecto `MisFunciones` y añadid una clase `SumaEnteros`, con el siguiente código:

```
// código de la clase SumaEnteros

// La sentencia import es similar al #include de C++.
// En este caso importamos la clase Scanner que nos
// permite realizar lecturas de diferentes flujos de entrada
import java.util.Scanner;

public class SumaEnteros {
	public static void main(String[] args) {
		int x,y;
		// Declaramos una variable de la clase Scanner y
		// le asignamos la creación  de un objeto de la clase Scanner
		// asociado al canal estándar de entrada (System.in)
		Scanner scanner = new Scanner(System.in);
		// Escribimos en el canal estándar de salida (System.out)
		// utilizando el método println
		System.out.println("Escribe un entero:");
		// Leemos un entero del flujo de entrada asociado al
		// objeto scanner (es decir, del canal estándar de entrada)
		x = scanner.nextInt();
		System.out.println("Escribe otro entero:");
		y = scanner.nextInt();
		System.out.println(x + " + " + y + " = " + (x + y));
	}
}
```

   Ejecutad el programa SumaEnteros.
   Observad que se está utilizando la librería *Scanner* de java (`import java.util.Scanner;`) .
   Esta librería permite realizar lecturas desde teclado. En este caso realiza lectura de enteros utilizando el método *nextInt()*.

2. (MaxMax) Añadid al proyecto `MisFunciones` una clase `MaxDos` que permita leer dos enteros de consola y escribir el máximo. Añadid luego una clase `MaxTres` que haga lo mismo con tres enteros.

3. (División) Seguimos en el proyecto `MisFunciones`. Ahora queremos añadir programas que nos permitan dividir números reales, para ello tenéis que leer valores reales del canal estándar de entrada. Consultad la API de la librería *Scanner* (https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html) para ver que método/s permite/n leer reales. Añadid el código necesario para evitar errores a causa de una posible división por 0. Ejecutad `DividirReales`.

4. (HolaMundos repetidos). Una introducción a las iteraciones.

   - Añadid al proyecto `HolaMundo` otra clase llamada `WhileMundo`. Dicha clase ha de contener un programa `WhileMundo` que utilice un bucle `while` para escribir `Hola mundo!` 10 veces con el número de orden delante. La salida ha de ser similar a:

```
1: Hola mundo!
2: Hola mundo!
3: Hola mundo!
4: Hola mundo!
5: Hola mundo!
6: Hola mundo!
7: Hola mundo!
8: Hola mundo!
9: Hola mundo!
10: Hola mundo!
```

- Añadid al proyecto `HolaMundo` otra clase llamada `ForMundo`. que haga lo mismo utilizando un bucle `for`.

5. (Mundos Alternados) Añadid al proyecto `HolaMundo` anterior una clase `HolaAdios`, que utilizando un bucle y una alternativa `if` escriba:

```
1: Hola mundo feliz!
2: Adiós mundo cruel!
3: Hola mundo feliz!
4: Adiós mundo cruel!
5: Hola mundo feliz!
6: Adiós mundo cruel!
7: Hola mundo feliz!
8: Adiós mundo cruel!
9: Hola mundo feliz!
10: Adiós mundo cruel!
```

6. En el proyecto `MisFunciones` añadid una clase `IntCalculoFactorial` que implemente un programa que lea un entero (pequeño) y calcule su factorial. Recordad que para $n \geq 1$ el factorial es:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$$

El programa utilizará (al menos) dos variables te tipo `int`.

- Una variable `n` para leer desde consola el valor de n.
- Otra variable `fact` para almacenar los productos parciales.

A lo largo de la iteración los distintos valores de `fact` serán

- Antes de entrar en la iteración `fact` vale 1.
- Tras el primer paso de la iteración `fact` vale $1 \cdot n$.
- Tras el segundo paso, `fact` vale $1 \cdot n \cdot (n-1)$.
- Así sucesivamente

Ejecutad y probad el programa `IntCalculoFactorial`. Mirad que pasa cuando el número n aumenta.

Cread otra clase `LongCalculoFactorial` en que las variables `n` y `fact` sean de tipo `long`. Qué ventajas (o inconvenientes) tiene respecto a la anterior.

5. El depurador

Depurar (en inglés *debug*) es el proceso de detectar, aislar y eliminar los errores (de un programa) o los fallos (de un sistema informático) de manera que el su comportamiento sea el esperado. El depurador (en inglés *debugger*) es la herramienta que permite seguir la ejecución de un programa de forma que se pueda detectar y corregir los errores.

El depurador permite ejecutar un programa línea a línea y también permite la consulta interactiva de los valores de las variables y del entorno. Permite fijar los puntos críticos en que detener la ejecución del programa (en inglés *breakpoints*) y posibilita ejecutar estas instancies críticas paso a paso.

Dentro del entorno de desarrollo de Eclipse se pueden depurar programes desarrollados en Java. Para hacerlo hay que cambiar a la perspectiva de depuración (debug perspective). Para ello seleccionad *Window→Open perspective→Debug*, como se muestra en la Fig. 16, con lo que nuestra ventana tendrá un aspecto similar al mostrado en la Fig. 17.

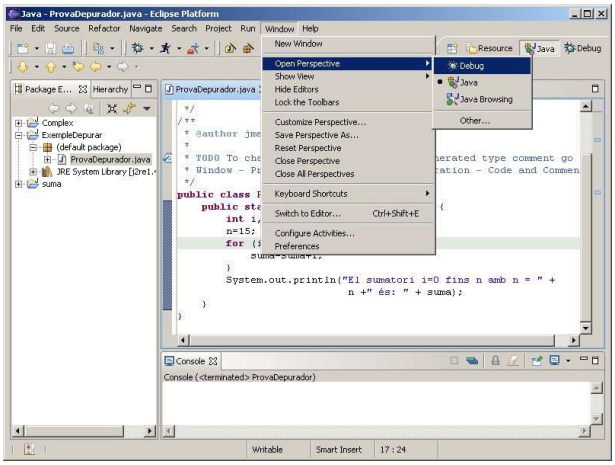

Figura 16: Abriendo la perspectiva *Debug*

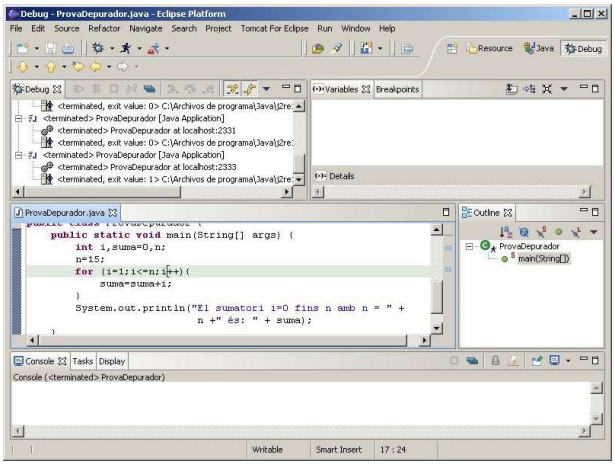

Figura 17: Perspectiva Debug.

Una vez hemos abierto la perspectiva de depuración podemos parar la ejecución del programa en una determinada línea (poniendo un breakpoint) e inspeccionar las variables locales. Para poner un breakpoint, en la vista en que se encuentra el código fuente seleccionamos la línea en que queremos que se detenga la ejecución y seleccionamos *Run→Toggle Line Breakpoint*, como se muestra en la Fig. 18. Veremos que se muestra un punto azul en la parte izquierda de la línea. Ahora ya podemos lanzar el depurador. Seleccionamos en el menú *As→Java Application*, como se muestra en la Fig. 19. La ejecución del programa se detiene en el primer breakpoint. Una vez el programa está parado, en una de las vistas podemos ver el valor de las variables o ver los breakpoints que hemos definido. En la Fig. 20 se muestra un programa parado en una línea en que en la ventana *Variables* se puede ver el contenido de las variables definidas en este punto. Una vez inspeccionado el código en el que se encuentra el problema se puede optar por ejecutar el programa hasta que acabe (Resume), acabar el programa inmediatamente (Terminate), ejecutar paso a paso el programa sin entrar en el código de los métodos (Step Over) o ejecutar paso a paso entrando en el código de los métodos (Step Into). Todas estas opciones se encuentran en el menú *Run* (ver Fig. 21)

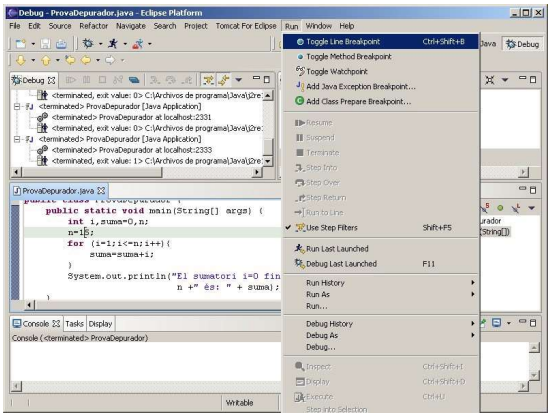

Figura 18: Añadiendo un breakpoint.

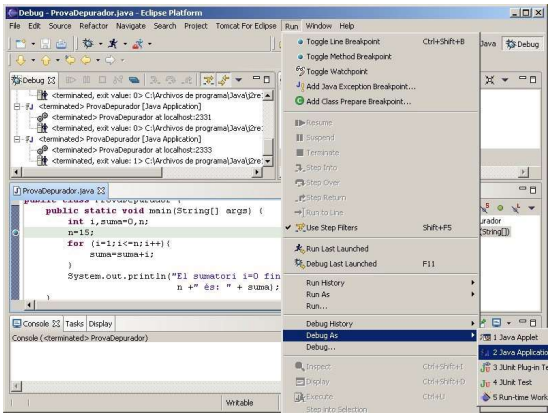

Figura 19: Lanzando el depurador.

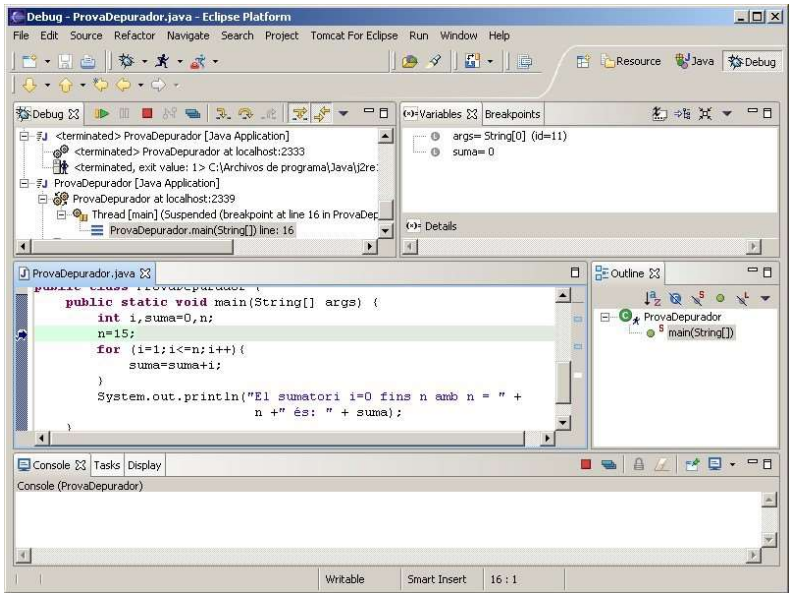

Figura 20: Depurando un programa.

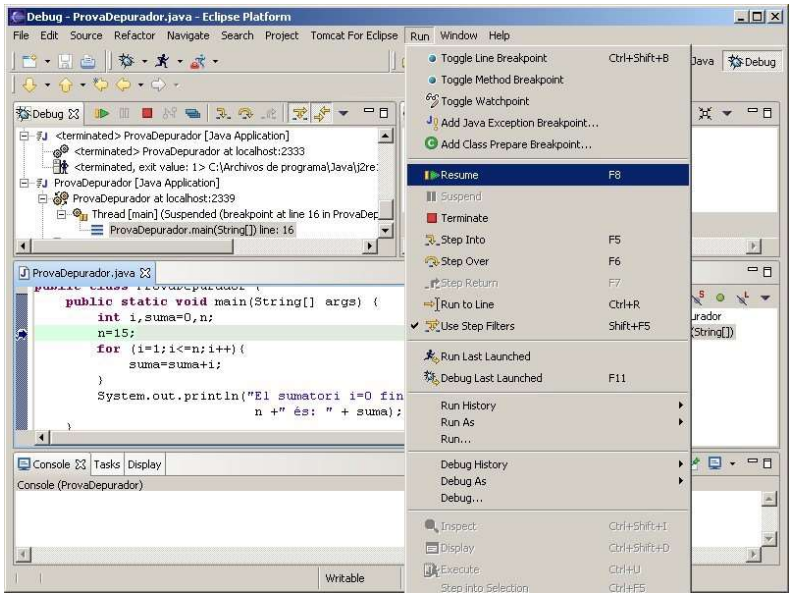

Figura 21: Opciones de ejecución del depurador.