



 **ECLIPSE[®]**
FOUNDATION

Introducción a Eclipse¹

1. Tabla de contenido

Introducción a Eclipse.....	1
2. ¿Qué es Eclipse?.....	2
3. Instalación de Eclipse	2
4. Primeros pasos.....	2
5. Términos importantes para trabajar con Eclipse	4
5.1. Workspace o área de trabajo.....	4
5.2. Proyecto	4
5.3. Vistas y editores.....	4
5.4. Perspectiva	5
6. Primer programa Java.....	5
6.1. Crear un proyecto	5
6.2. Crear una clase Java.....	6
6.3. Importar proyectos	8
6.4. Compilar y detectar errores	8
6.5. Ejecutar una clase	10
6.5.1. Suministrar parámetros de ejecución	10
6.5.2 Detener la ejecución	11
7. Algunos trucos.....	12
7.1. Quick Fix	12
7.2. Comentar/des-comentar un código	12
7.3. Ctrl + espacio.....	12
7.4. Mostrar los métodos de un objeto	14
7.5. Plantillas	14
7.6. Formatear código	16
7.7. Configurar formato del código	16
7.8. Rodear código con try - catch.....	16
7.9. Buscar uso (referencias) de variables, métodos, clases....	17
7.10. Refactoring	18
7.11. Optimizar imports.	20
7.12. Generar métodos get/set.	21
7.13. Generar constructores.	22
7.14. Incluir el esqueleto de métodos de una superclase o de una interfaz.	22
7.15. Crear jar.....	23
8. Depuración de programas	25
9. Referencias:	28

¹ Autores: Julen Sánchez Oroz, Ana Romero Ibáñez y Francisco J. García Izquierdo, Dpto. de Matemáticas y Computación, Universidad de La Rioja.Febrero de 2015.

2. ¿Qué es Eclipse?

Eclipse es un entorno de desarrollo integrado (IDE) para Java y otros lenguajes de programación como PHP, JavaScript.... Hoy en día es el entorno de desarrollo para Java más utilizado, con una cuota de mercado de aproximadamente 65%.

Eclipse está desarrollado por una comunidad de código abierto y se utiliza en varias áreas diferentes, por ejemplo, para el desarrollo de aplicaciones Java o Android. La comunidad de código abierto Eclipse cuenta con más de 200 proyectos de software libre que cubren diferentes aspectos del desarrollo de software. El IDE de Eclipse se puede ampliar con diversos componentes de software adicionales llamados plugins.

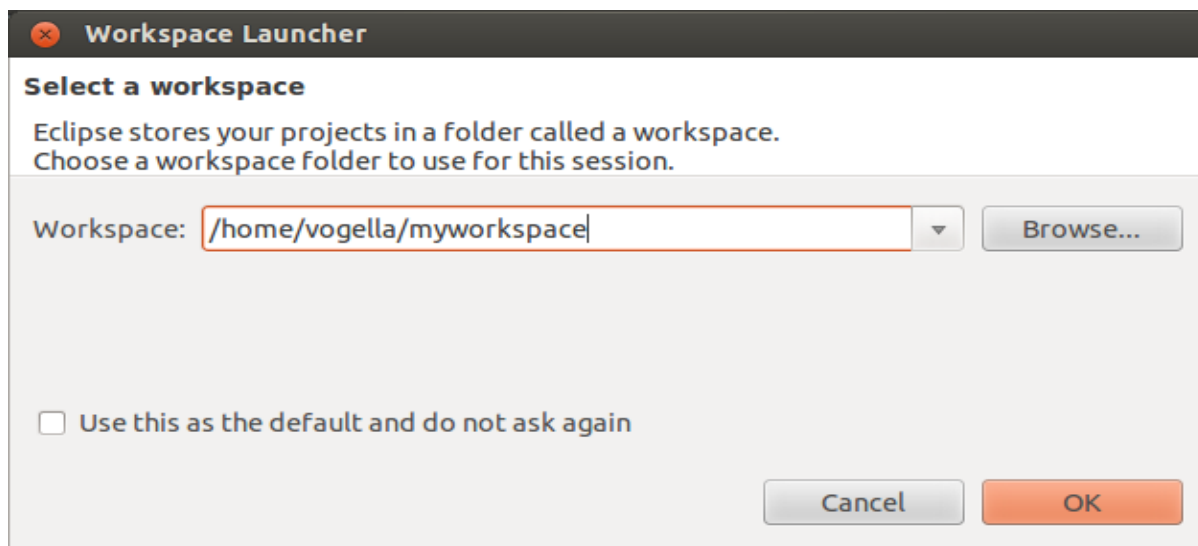
3. Instalación de Eclipse

Antes de instalar Eclipse hay que asegurarse de que tenemos un JRE o JDK instalado en el sistema, ya que Eclipse está desarrollado en Java y necesita de un entorno de ejecución Java para poder ejecutarse.

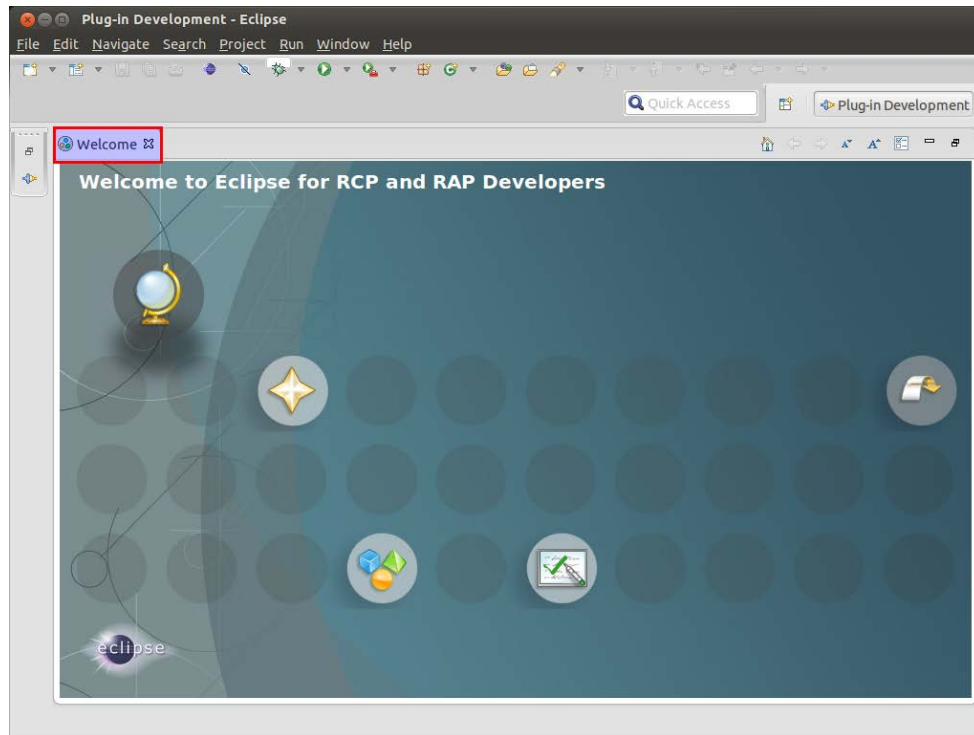
La instalación de Eclipse es muy sencilla. Podemos descargárnoslo de www.eclipse.org en forma de archivo ZIP y sólo tenemos que descomprimirlo en la carpeta donde queramos tenerlo instalado. Para ejecutarlo sólo hay que arrancar el fichero eclipse.exe (Windows) o Eclipse (Linux y Mac), sin necesidad de realizar ninguna instalación.

4. Primeros pasos

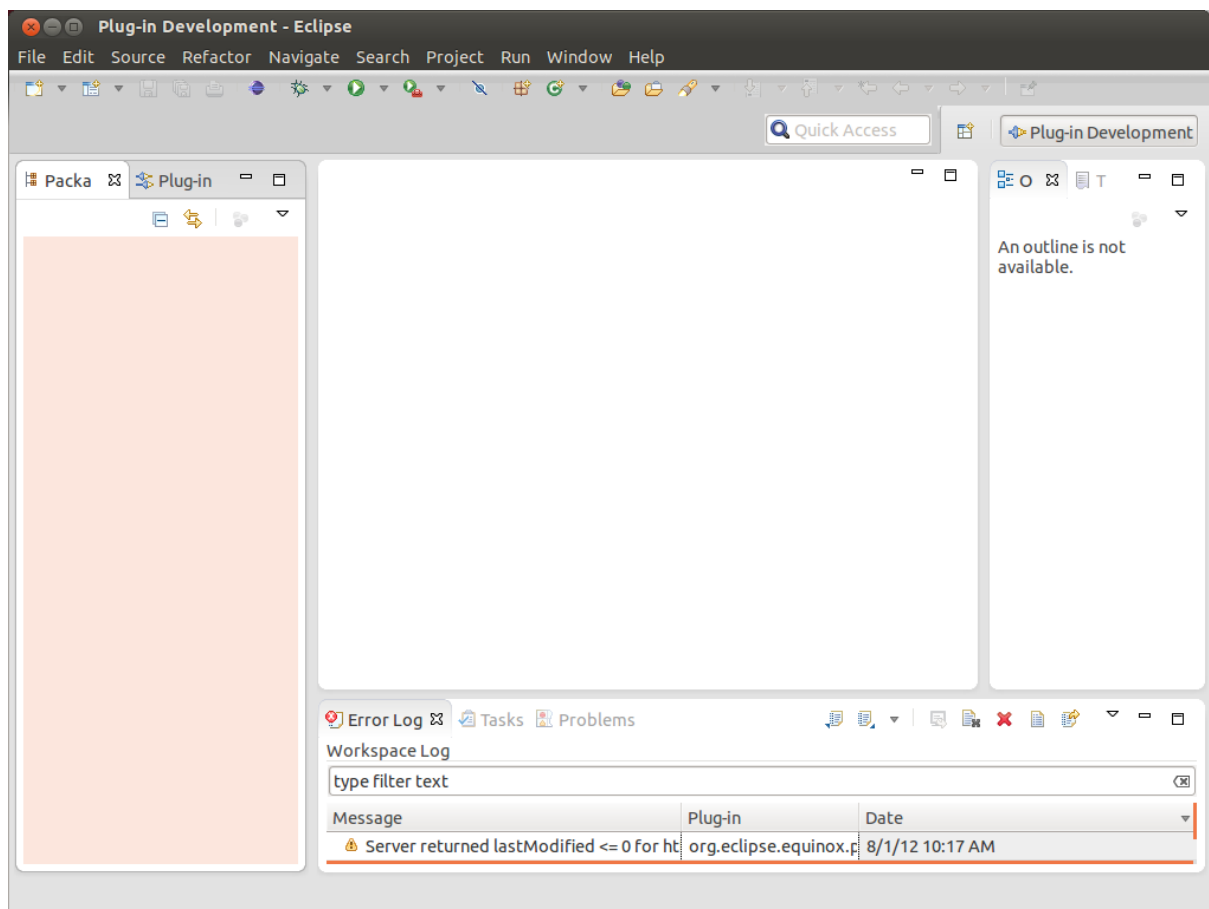
Para iniciar Eclipse, hacer doble clic en el archivo "eclipse.exe/eclipse" en el directorio donde se ha descomprimido Eclipse. En primer lugar el sistema pedirá un workspace o área de trabajo que es el lugar en el que se guardarán todos los proyectos creados en Eclipse. Hay que seleccionar un directorio vacío y pulsar el botón Aceptar.



Eclipse se iniciará y mostrará la página de bienvenida.



Cerramos la pantalla de bienvenida y se debería ver una pantalla similar a la siguiente.



5. Términos importantes para trabajar con Eclipse

5.1. Workspace o área de trabajo

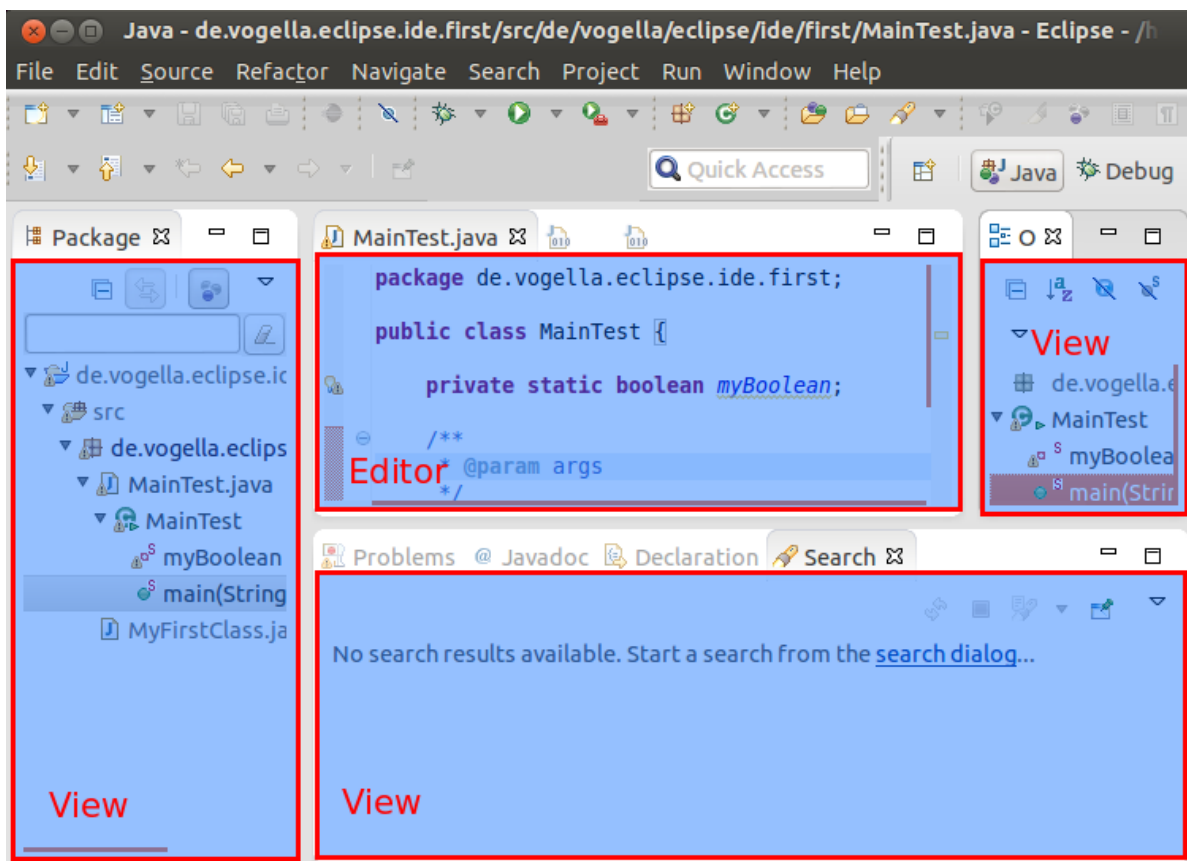
El **workspace** es la ubicación lógica donde existen uno o varios proyectos. Se puede elegir el workspace durante el inicio de Eclipse o a través del menú "File/Switch/Workspace/ Other".

5.2. Proyecto

Un proyecto de Eclipse contiene los archivos fuente, configuración y binarios relacionados con una tarea determinada y los agrupa en componentes reutilizables en el **directorio del proyecto**. Es importante saber cuál es ese directorio pues, por ejemplo, será donde Eclipse busque los ficheros a los que nuestros programas quieran acceder. En el apartado 5.1. Crear un proyecto se explica cómo se crea un proyecto.

5.3. Vistas y editores

La interfaz de usuario de Eclipse está dividida en **vistas** y editores que permiten navegar a través de los diferentes datos y modificarlos.



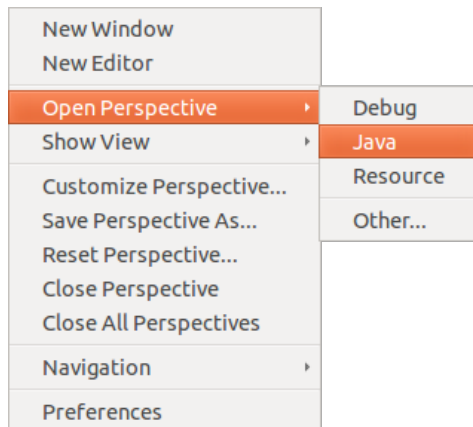
Las Vistas son ventanas que nos muestran distinta información sobre el proyecto en el que nos encontramos. Un ejemplo de Vista es el **Explorador de paquetes**, que permite navegar por los archivos de los proyectos de Eclipse. Si se cambia algún dato en el Explorador de paquetes, por ejemplo, cambiar el nombre de un archivo, el nombre del archivo se cambia directamente en el sistema de archivos.

Los editores se utilizan normalmente para modificar los diferentes archivos. Por ejemplo, el **Editor de Java** se utiliza para modificar los archivos fuente de Java. Los cambios en el archivo de origen se aplican una vez que el usuario selecciona el comando "Guardar".

5.4. Perspectiva

Una **perspectiva** es un contenedor visual para un conjunto de vistas y editores. Cada perspectiva contendrá una barra de herramientas con las tareas de desarrollo correspondientes (que pueden cambiar de una perspectiva a otra). Para cambiar la perspectiva actual, hay que seleccionar “*Window/Open Perspective*”.

Las perspectivas principales que se utilizan en Eclipse son la **perspectiva Java** para el desarrollo de Java y la perspectiva **Debug** para depurar nuestros programas.



Se puede cambiar el contenido de una perspectiva añadiendo nuevas vistas en el menú “*Window/Show View*”. También se pueden modificar la barra de herramientas y distintas opciones del menú en “*Window/Customize Perspective*”.

La perspectiva modificada se puede guardar a través de “*Window/Save Perspective as*”.

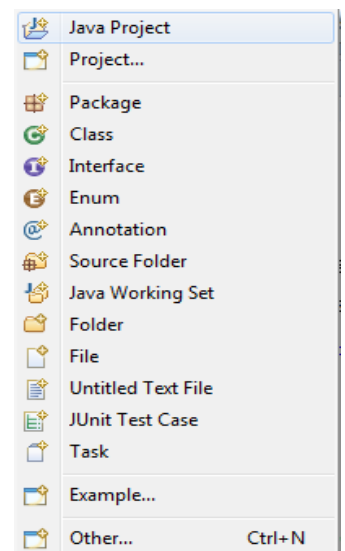
6. Primer programa Java

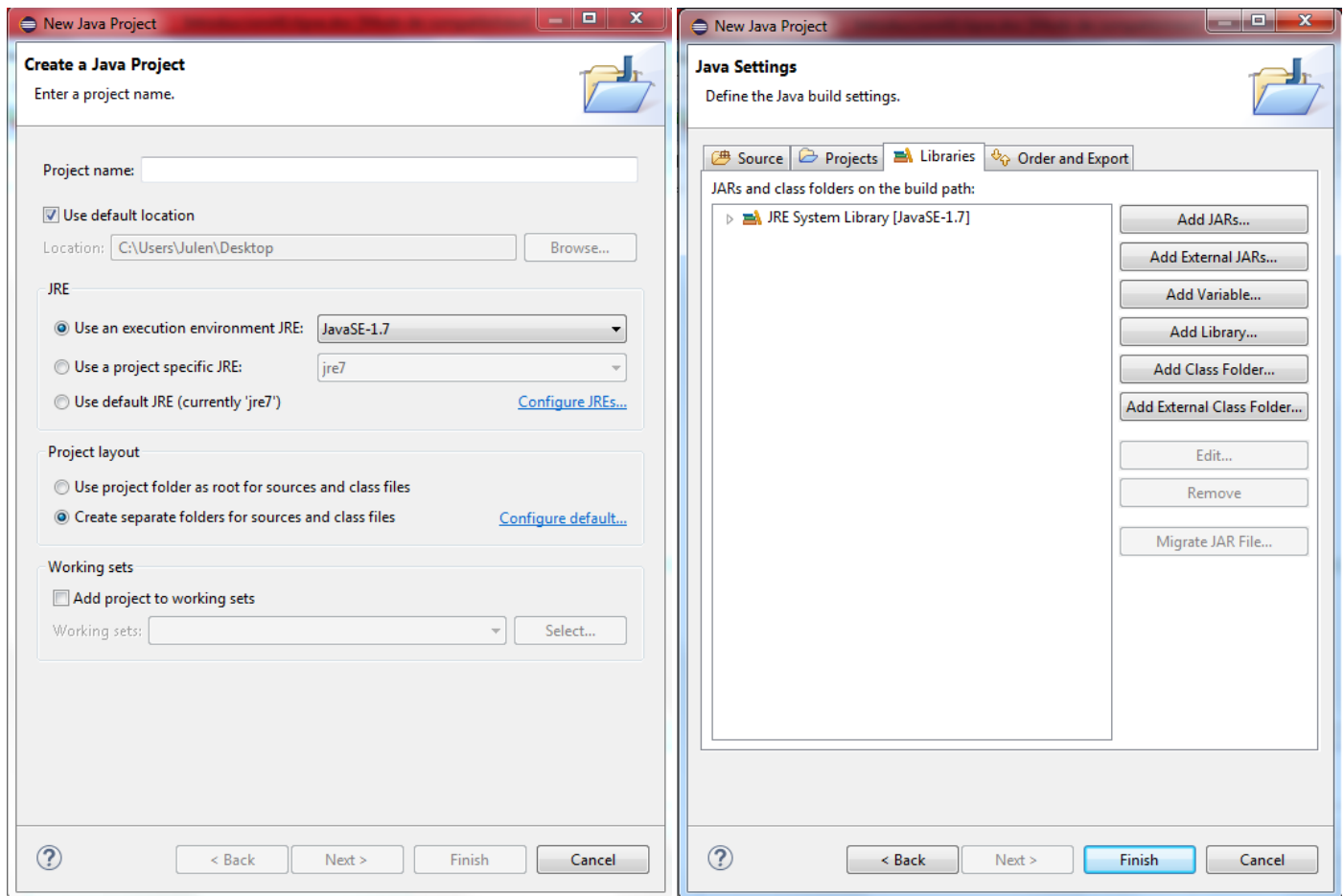
A continuación se describe cómo crear un programa de Java usando Eclipse.

6.1. Crear un proyecto

Seleccionar “*File/New/Java Project*” y elegir el nombre del proyecto. **Error muy común, no confundir proyecto con proyecto java**

Se puede cambiar el directorio por defecto en donde se guardará el proyecto. Basta con eliminar la selección “*Use default location*” y elegir el directorio deseado. También podemos añadir librerías o JARs externos a nuestro proyecto. Para ello seleccionamos “*Next >*” y en la nueva ventana seleccionamos la pestaña “*Libraries*”.





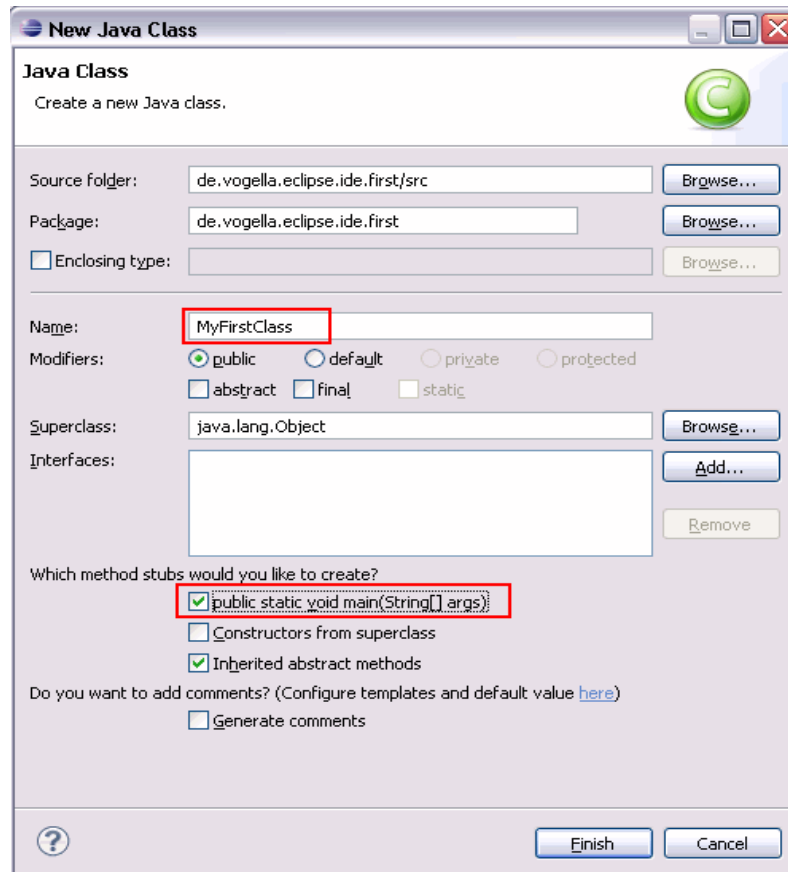
El nuevo proyecto se crea y se muestra como una carpeta y su contenido se puede inspeccionar en la vista del explorador de paquetes que aparece a la izquierda.

6.2. Crear una clase Java

Hacer clic con el ratón en el proyecto y seleccionar "New/Class". Elegimos el nombre de la clase y del paquete contenedor (las convenciones de Java sugieren que el nombre de una clase debería comenzar con mayúscula). Si el paquete no existiese, se crearía junto con la clase. Si no especificamos ningún paquete, se guardarán dentro de un paquete por defecto.

También existen otros modificadores opcionales que pueden ser añadidos a una clase en el mismo momento de su creación (también podrían ser añadidos manualmente en otras fases más avanzadas del proceso de desarrollo). Si se pretende que la nueva clase extienda (herede de) otra clase existente, se debería especificar la clase "padre" dentro del campo "Superclass". El botón "Browse..." es de gran utilidad a la hora de encontrar clases que sean posibles candidatas para ser extendidas. Aunque Java sólo soporta herencia única (sólo puede extenderse de una única clase) sí que es posible que una clase implemente más de una interfaz. Una vez más, el botón "Browse..." simplifica la tarea de seleccionar interfaces implementadas.

Si se desea que la nueva clase contenga un método "main" (es decir, el punto inicial de ejecución del programa), puede añadirse dicho método automáticamente sólo con marcar la casilla con la opción apropiada. También pueden implementarse de esta manera los constructores de la superclase y todos los métodos abstractos heredados. Esta última opción es muy útil si se desea instanciar la clase puesto que para esto, todo método abstracto debería estar implementado.



Veamos cómo se hace esto mediante un ejemplo. Creamos la clase Empleado, que como podemos ver contiene un método abstracto.

```
1
2 public abstract class Empleado {
3     private String dni;
4     private double precioBase;
5
6     public Empleado(String dni, double precioBase){
7         this.dni = dni;
8         this.precioBase = precioBase;
9     }
10
11     public String getDni(){
12         return dni;
13     }
14
15     public double getPrecioBase(){
16         return precioBase;
17     }
18
19     abstract double PrecioTotal();
20
21 }
22
```

A continuación vamos a crear la clase Técnico, que heredará de Empleado, por lo que queremos que implemente el método "PrecioTotal()". Para ello, como ya hemos explicado anteriormente, en la creación de esta clase se escoge como superclase Empleado, y se marca la casilla "Constructors from superclass" y "Inherited abstract methods" (de este modo crearemos automáticamente el constructor heredando del de Empleado, y todos los métodos abstractos de la clase padre). El resultado es el siguiente.


```

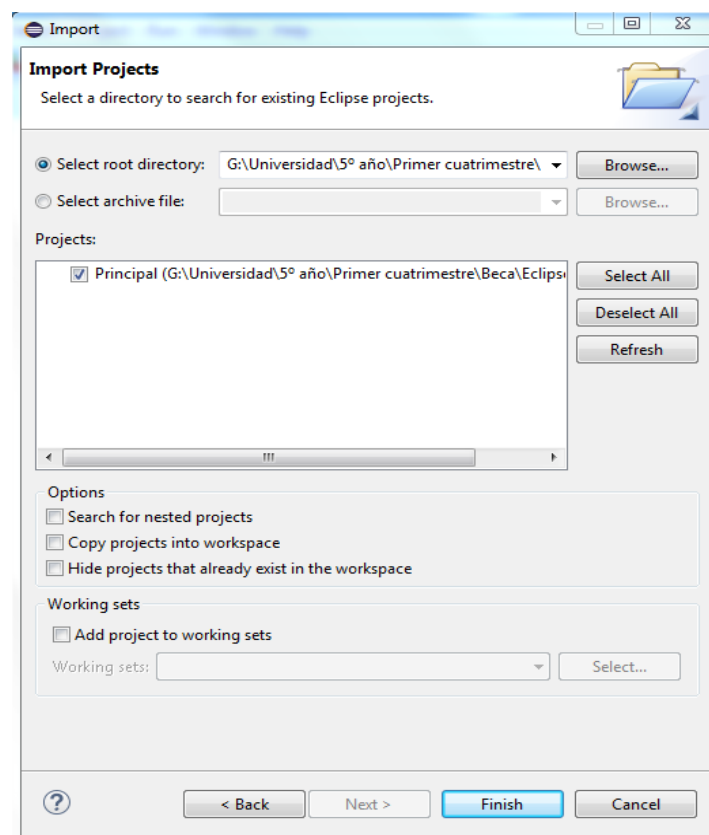
1 |
2 | public class Tecnico extends Empleado {
3 |
4 |     public Tecnico(String dni, double precioBaso) {
5 |         super(dni, precioBaso);
6 |         // TODO Auto-generated constructor stub
7 |     }
8 |
9 |     @Override
10 |    double PrecioTotal() {
11 |        // TODO Auto-generated method stub
12 |        return 0;
13 |    }
14 |
15 | }
16 |

```

6.3. Importar proyectos (abrir proyectos existentes)

A continuación explicaremos cómo importar nuestro proyecto java, ya que este es el modo de abrir proyectos en Eclipse. Para ello acudimos al menú contextual "File/Import../Existing Projects into Workspace".

Seleccionamos el proyecto que queremos importar (si seleccionamos un directorio, nos importará todos los proyectos java que contenga) y le damos a *Finish*. Ya tenemos nuestro proyecto importado a Eclipse.

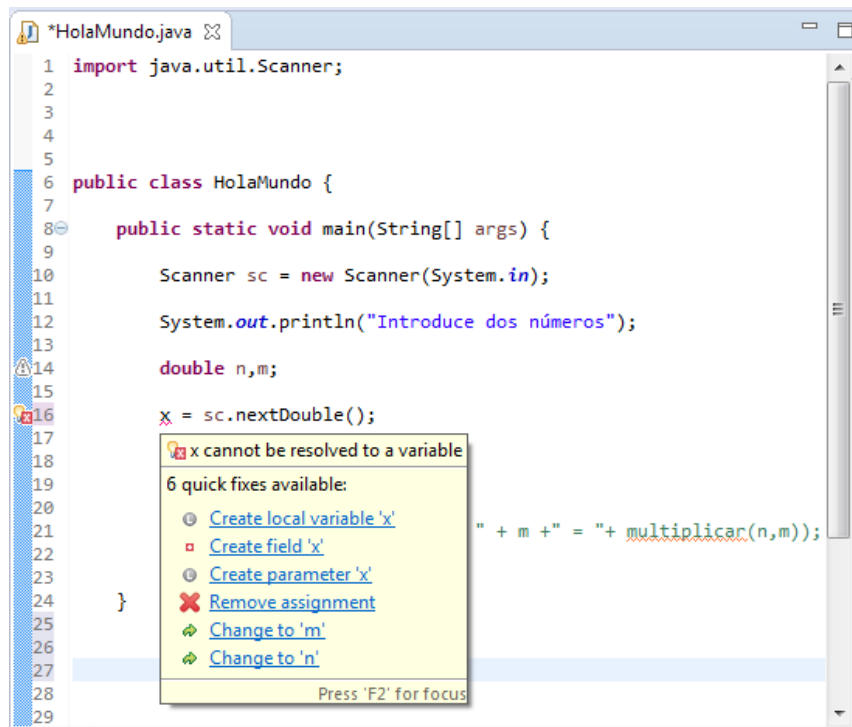


6.4. Compilar y detectar errores

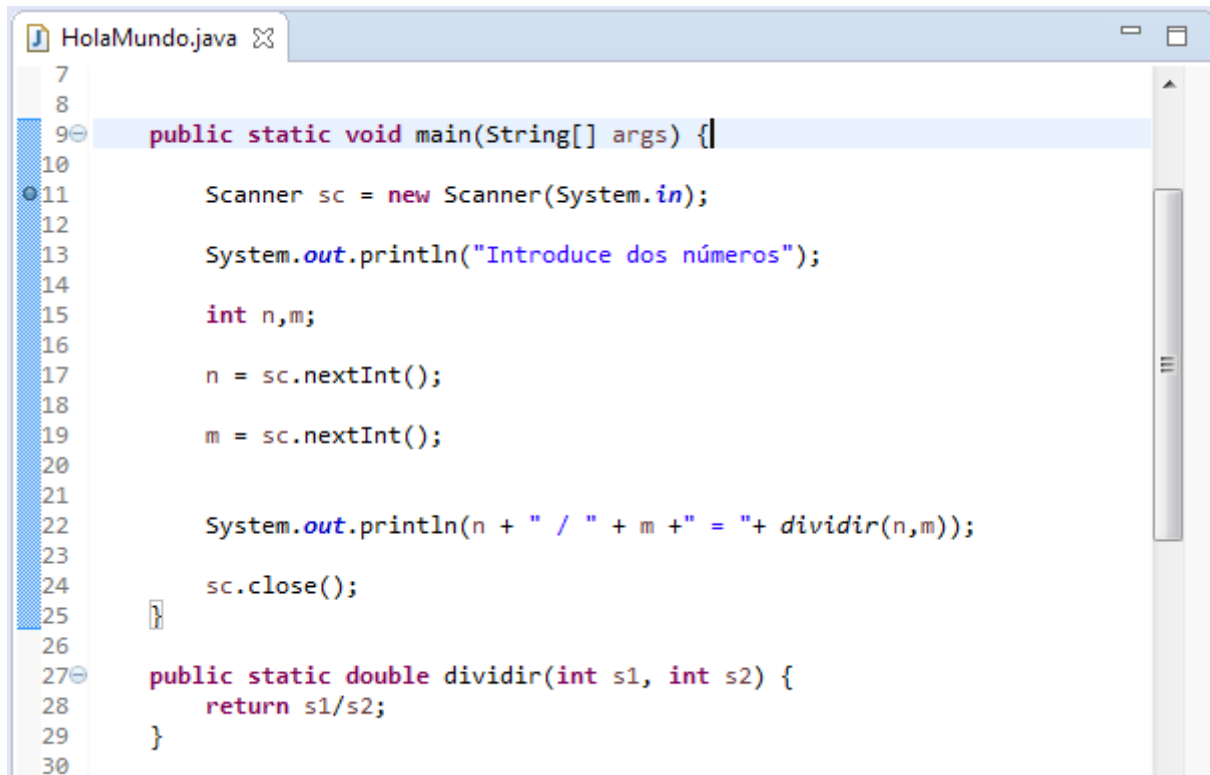
En Eclipse los errores de compilación se muestran en tiempo real subrayando el fragmento de código adecuado con una línea roja. Además el entorno compila automáticamente los archivos salvados.

Los errores pueden encontrarse fácilmente porque se muestran además como marcas rojas en el margen izquierdo del editor de código Java. Las advertencias (warnings) se muestran de la misma manera, pero con marcas amarillas.

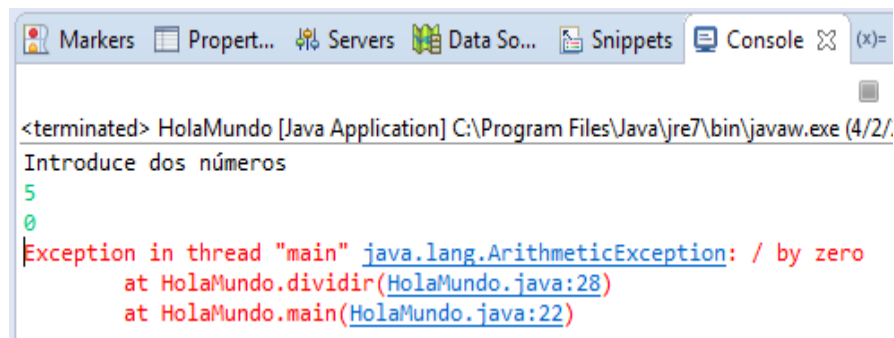
Por tanto, simplemente situando el cursor encima del error, eclipse nos mostrará información acerca de este.



Eclipse también nos informa de los errores de ejecución, errores que se producen mientras se ejecuta el programa. Se puede usar la traza de error que devuelve el programa para acudir a la línea de código en la que se ha producido el error (directamente haciendo clic sobre dicha traza). Veámoslo con el siguiente programa, en el que realizamos una división por o.




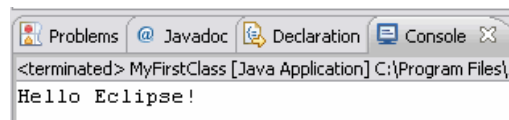
Y finalmente haciendo clic sobre las referencias, acudimos a la parte del código en la que se ha producido el error.



```
<terminated> HolaMundo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (4/2/
Introduce dos números
5
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at HolaMundo.dividir(HolaMundo.java:28)
    at HolaMundo.main(HolaMundo.java:22)
```

6.5. Ejecutar una clase

Para ejecutar una clase dentro de Eclipse hay que seleccionar "Run/ Run..." del menú principal o pulsar el botón . Eclipse ejecutará el programa Java. La salida debe verse en la vista **consola**.



```
<terminated> MyFirstClass [Java Application] C:\Program Files\J
Hello Eclipse!
```

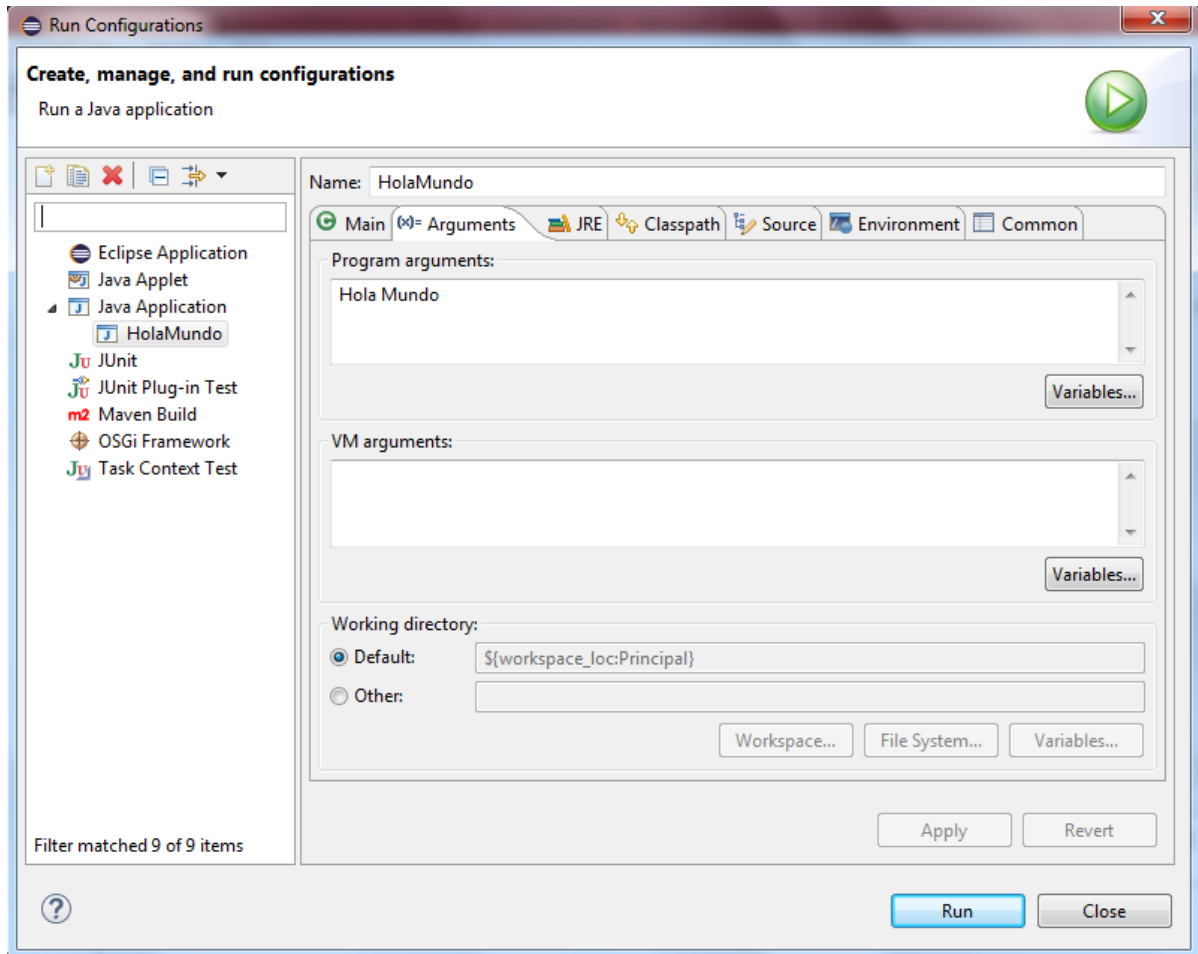
PISTA: Es posible que se pierda la vista consola y tengas que recuperarla. Para acceder a cualquier vista basta con acudir al menú contextual "Windows/Show view" y seleccionar la vista deseada.

6.5.1. Suministrar parámetros de ejecución

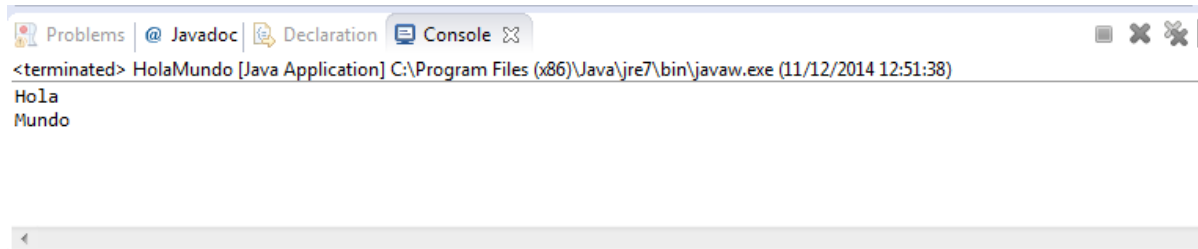
También podemos realizar una ejecución de una clase pasándole parámetros al método main. Consideremos como ejemplo la siguiente clase

```
6 public class HolaMundo {
7
8     public static void main(String[] args) {
9
10         for(int i = 0; i < args.length; i++)
11         {
12             System.out.println(args[i]);
13         }
14     }
15
16 }
```


Para asignarle parámetros de entrada al método main (los valores de args[]), acudimos a "Run/Run Configurations.." y a continuación en la pestaña "Arguments" podemos introducir todos los parámetros que deseemos separados por un espacio. Pinchamos el botón "Apply" y a continuación "Run".

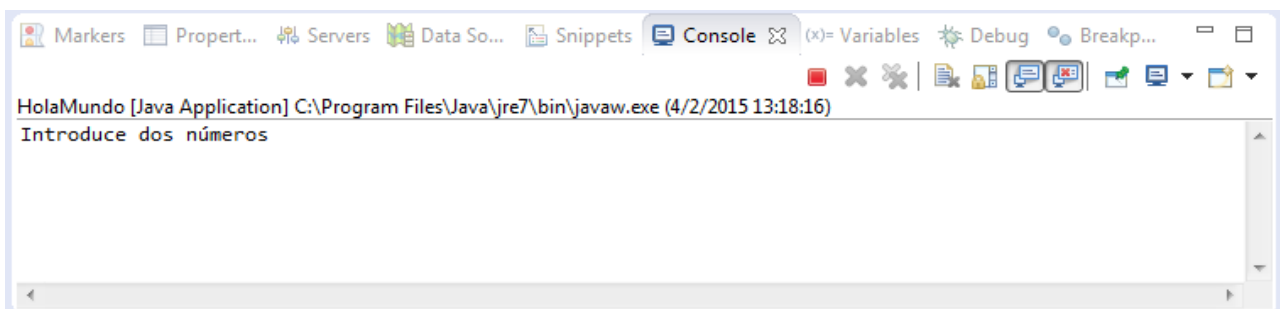


El resultado de la ejecución del programa es el siguiente.



6.5.2 Detener la ejecución

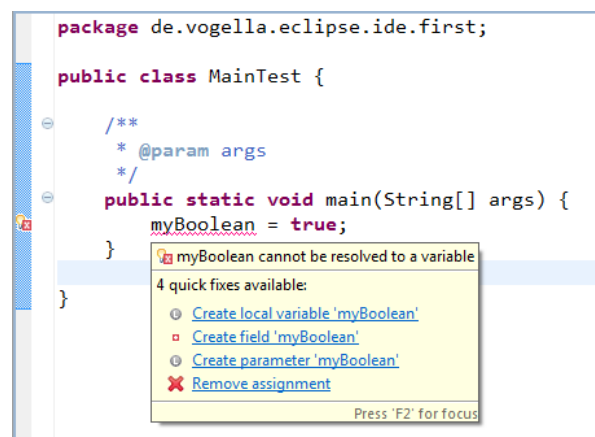
Si queremos detener la ejecución de un programa, podemos hacerlo de manera sencilla simplemente pulsando el icono  situado en la vista consola.



7. Algunos trucos

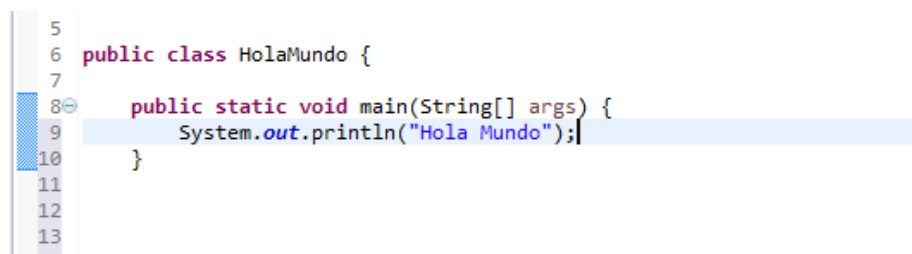
7.1. Quick Fix

Siempre que Eclipse detecta un problema, podemos seleccionar el texto subrayado y presionar “Ctrl+1” o seleccionar la cruz roja en el margen izquierdo para ver las propuestas de cómo resolver este problema. Esta opción hay que usarla meditando cuidadosamente las opciones que Eclipse nos ofrece. No siempre encontraremos la solución que necesitamos entre las que nos ofrece el entorno.

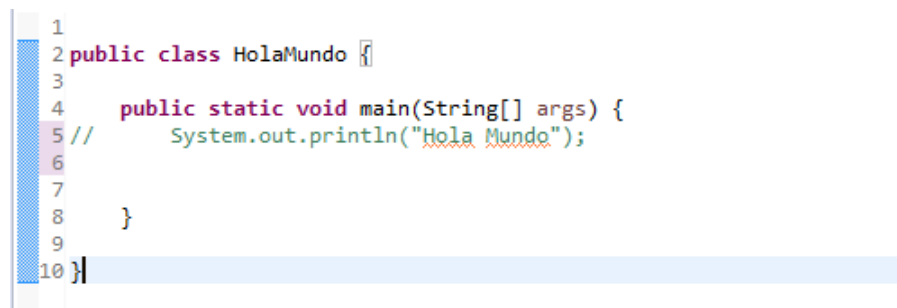


7.2. Comentar/des-comentar un código

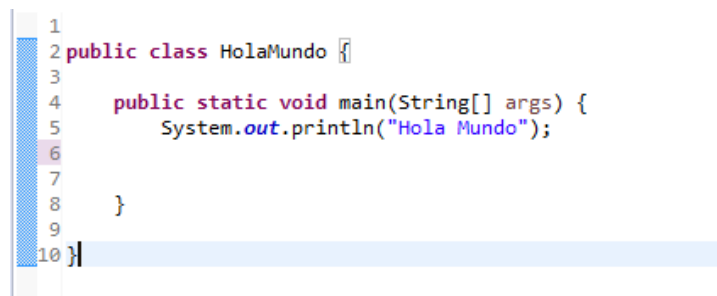
Eclipse proporciona un modo muy cómodo para comentar o des-comentar parte de código. Seleccionamos la línea que queremos comentar.



A continuación pulsamos “Ctrl+”/ (“Ctrl+ la tecla 7”) y la línea quedará comentada.



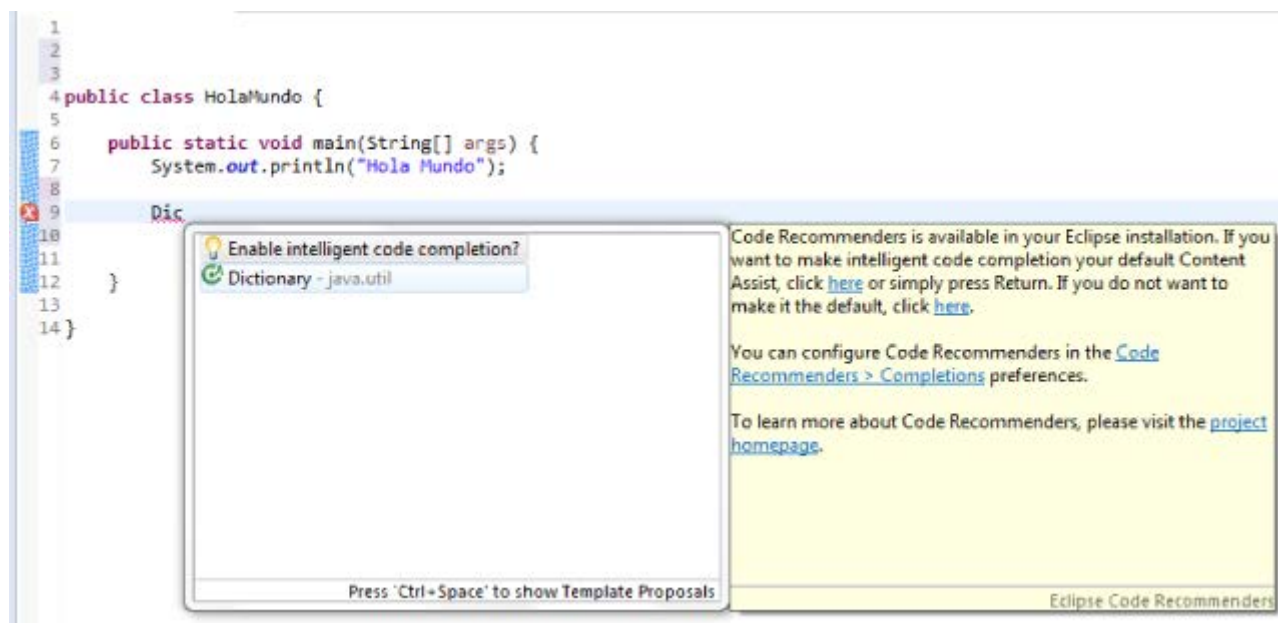
El caso de des-comentar es idéntico. Seleccionamos la línea que acabamos de comentar, y pulsamos “Ctrl+”/ (“Ctrl+ la tecla 7”). La línea queda des-comentada.



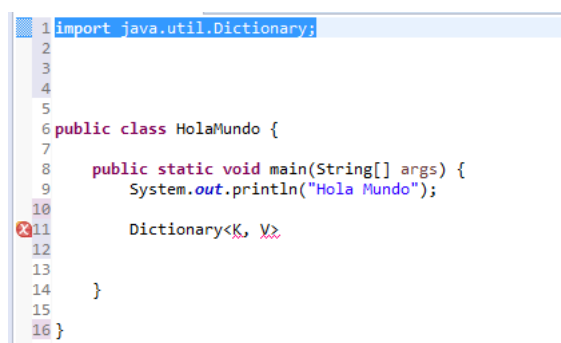
7.3. Ctrl + espacio

Eclipse nos ayuda a encontrar el nombre de una clase escribiendo los primeros caracteres y pulsando “Ctrl + espacio”. Eclipse mostrará las distintas posibilidades y simplemente haciendo clic sobre él se creará el objeto.

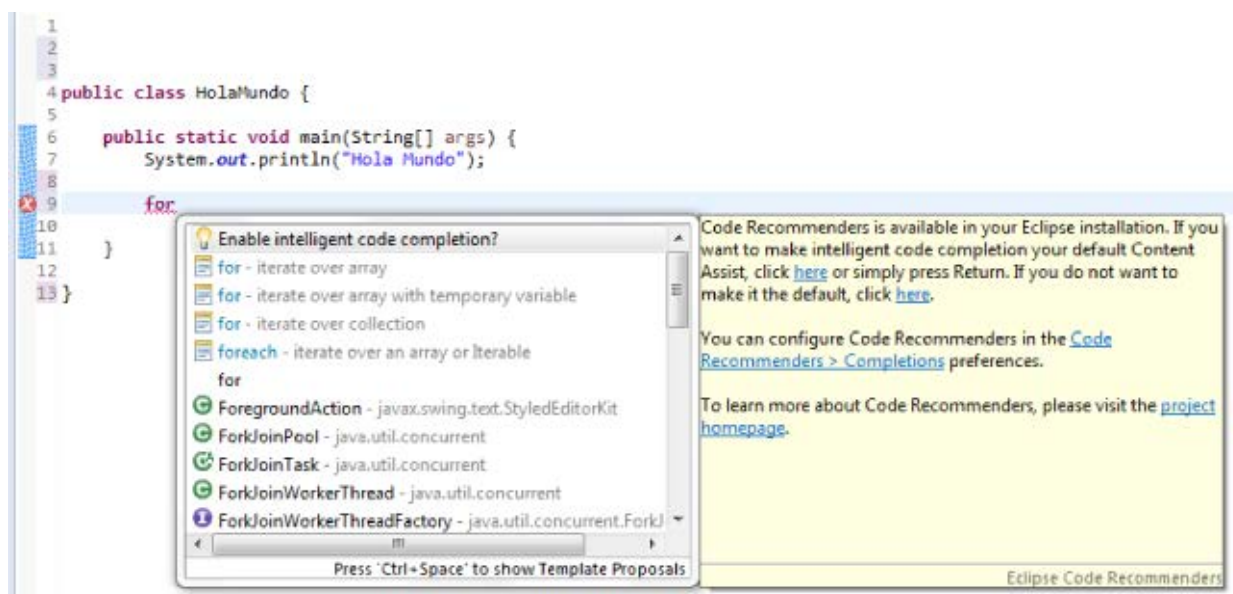
Por ejemplo, supongamos que queremos crear en nuestro proyecto un diccionario. No nos acordamos del nombre correcto, así que escribimos "dic" y pulsamos "Ctrl + espacio".



A continuación seleccionamos la opción deseada y Eclipse directamente nos importará la clase.



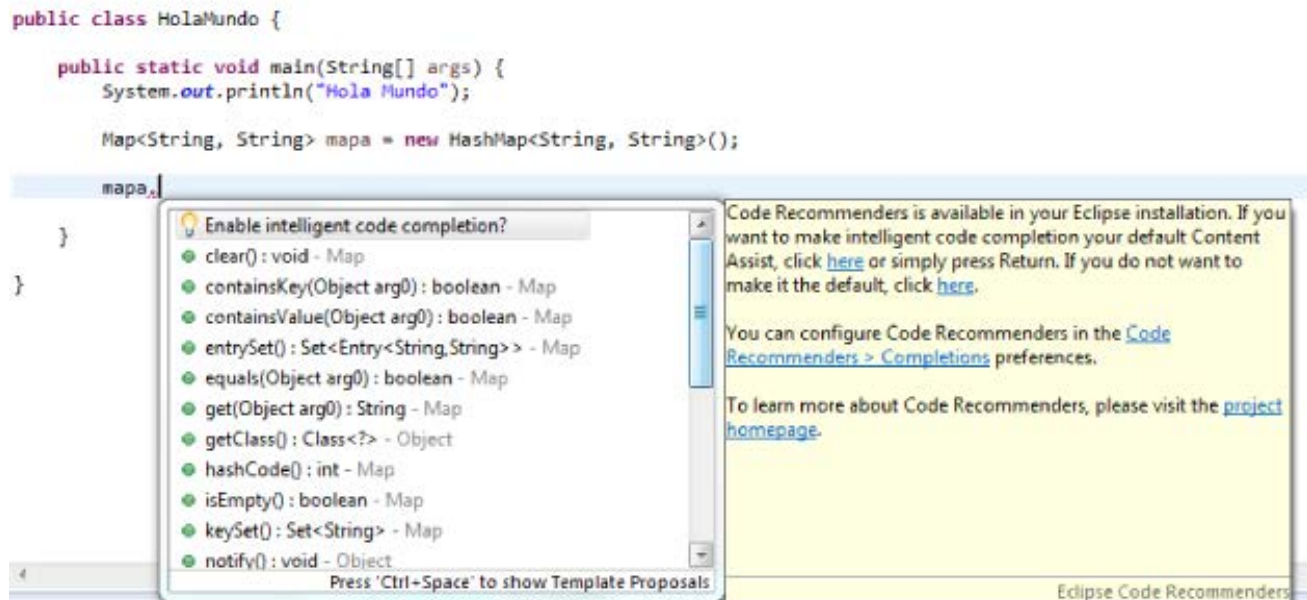
Además la opción "Ctrl + espacio" también sirve para autocompletar nombres de variables, métodos o estructuras (while,do,for...). Vamos a añadir un for a nuestra clase. Escribiremos for y a continuación pulsamos "Ctrl + espacio".



Nos muestra las distintas opciones a la hora de crear el bucle *for*, sobre que queremos iterar....

7.4. Mostrar los métodos de un objeto

Eclipse también nos permite conocer todos los métodos de una clase. Para ello basta con escribir "." a continuación de la variable de la clase deseada. A continuación vemos un ejemplo con la clase "Map" de java.



7.5. Plantillas

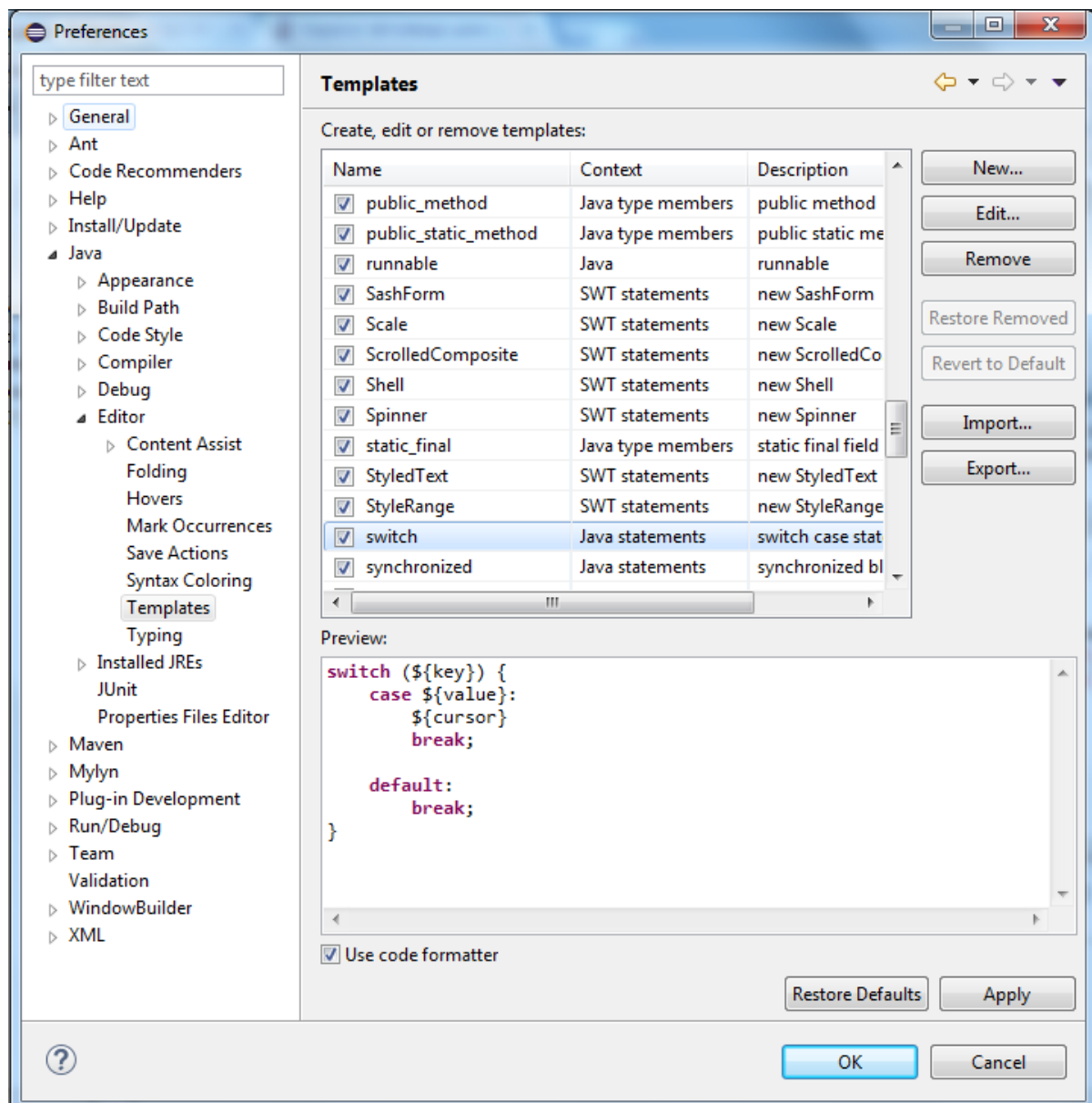
Ya hemos comentado la funcionalidad de "Ctrl + espacio" para el autocompletado. Por ejemplo, a partir de *sysout* y pulsando "Ctrl + espacio" obtenemos el código `System.out.println("")`.

A continuación mostramos una serie de plantillas más utilizadas, ordenadas alfabeticamente

PLANTILLA	CÓDIGO RESULTANTE
sysout (o syso)	<code>System.out.println(\${word_selection}\${});\${cursor}</code>
syserr	<code>System.err.println(\${word_selection}\${});\${cursor}</code>
main	<code>public static void main(String[] args) { \${cursor} }</code>
catch	<code>catch (\${Exception} \${exception_variable_name}) { \${cursor}</code>
do	<code>do { \${line_selection}\${cursor} } while (\${condition:var(boolean)});</code>
for(varias opciones)	<code>for (int \${index} = 0; \${index} < \${array}.length; \${index}++) { \${line_selection}\${cursor} }</code>
foreach	<code>for (\${iterable_type} \${iterable_element} : \${iterable}) { \${cursor} }</code>
ifelse	<code>if (\${condition:var(boolean)}) { \${cursor} } else { \${cursor} }</code>

switch	<pre> switch (\${key}) { case \${value}: \${cursor} break; default: break; } </pre>
try	<pre> try { \${line_selection}\${cursor} } catch (\${Exception} \${exception_variable_name}) { // \${todo}: handle exception } </pre>
while(varias opciones)	<pre> while (\${en:var(java.util.Enumeration)}.hasMoreElements()) { \${type:argType(en)} \${elem:newName(type)} = (\${type}) \${en}.nextElement(); \${cursor} } </pre>

Podemos acudir a la lista completa de plantillas a partir de "Window/Preferences/ Java/Editor/Templates"

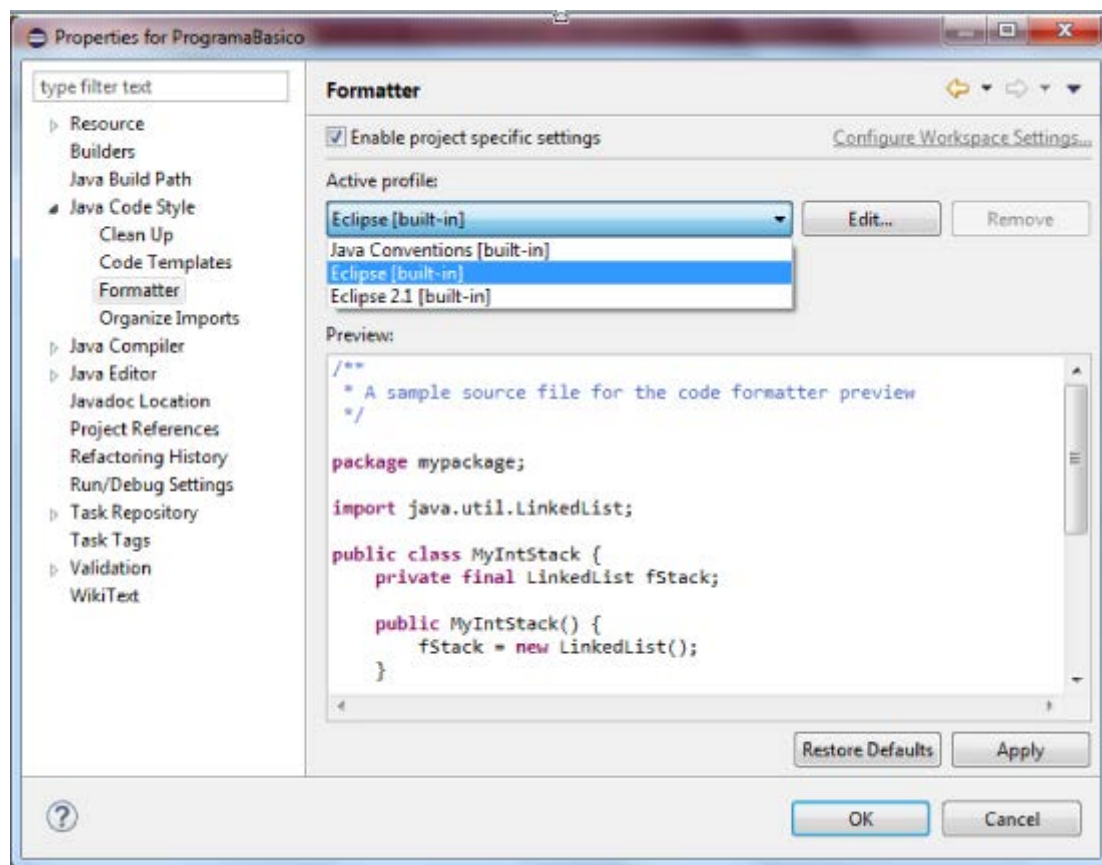


7.6. Formatear código

Podemos formatear código, es decir, dejar el código en un formato correcto (tabulaciones, saltos de línea, etc...) acudiendo al menú contextual, "source/format".

7.7. Configurar formato del código

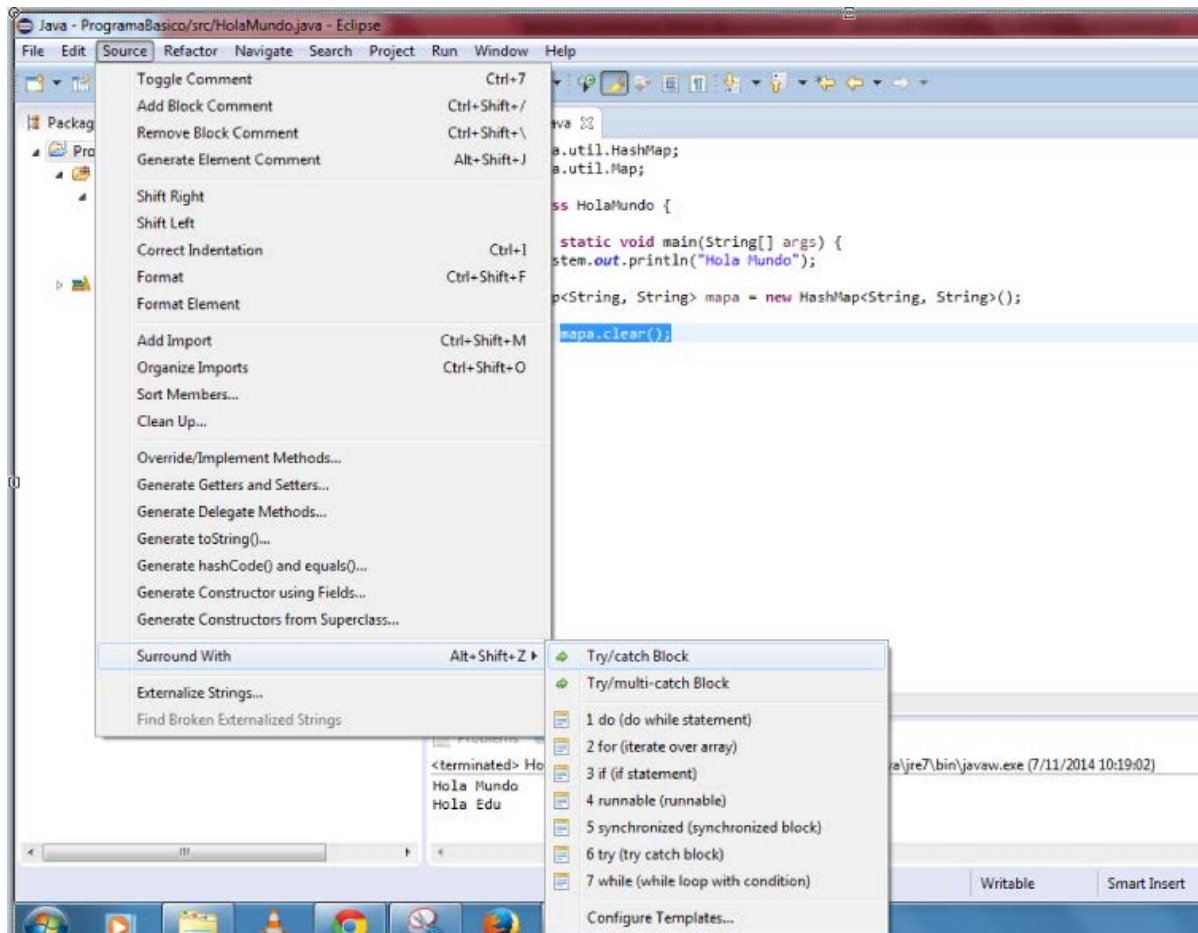
Si por el contrario queremos dar un formato al código distinto del que le da Eclipse por defecto, podemos ir al icono del proyecto y pinchando con botón derecho "Properties/Java Code Style/Formatter".



Podemos elegir distintas opciones que nos proporciona Eclipse, incluso editar algunas de las propiedades de cada opción.

7.8. Rodear código con try - catch

Si queremos introducir parte de un código en un bloque *try-catch*, seleccionamos dicho código y vamos a "Source/Surround With/Try/Catch Block".



Nótese que podemos rodear el código con más estructuras aparte de try-catch, como pueden ser do, for, if, A continuación vemos cómo nos incluye el código seleccionado en un bloque try-catch.

```

1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class HolaMundo {
5
6     public static void main(String[] args) {
7         System.out.println("Hola Mundo");
8
9         Map<String, String> mapa = new HashMap<String, String>();
10
11         try {
12             mapa.clear();
13         } catch (Exception e) {
14             // TODO Auto-generated catch block
15             e.printStackTrace();
16         }
17
18     }
19 }
20 }

```

7.9. Buscar uso (referencias) de variables, métodos, clases...

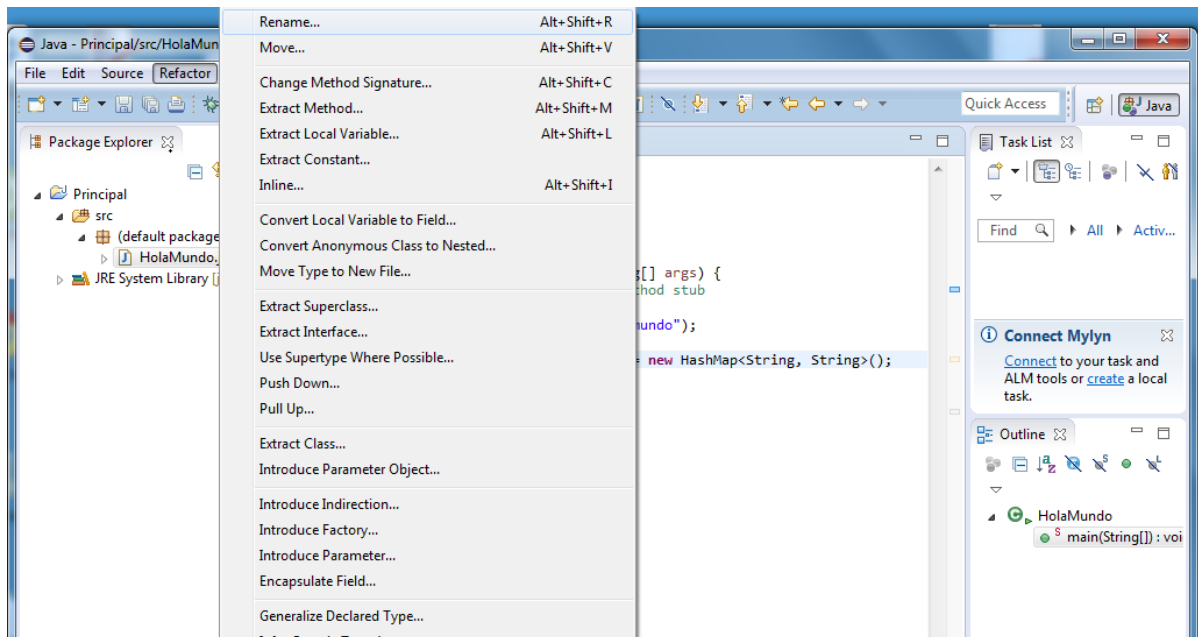
Podemos examinar con mucha rapidez en donde están referenciadas las distintas variables, métodos y clases de nuestro proyecto. Para ello seleccionamos la variable, método o clase, pinchamos botón derecho "References/Project".

Esto es muy útil, por ejemplo para comprobar que puedo eliminar un método, clase o variable sin afectar a mi programa.

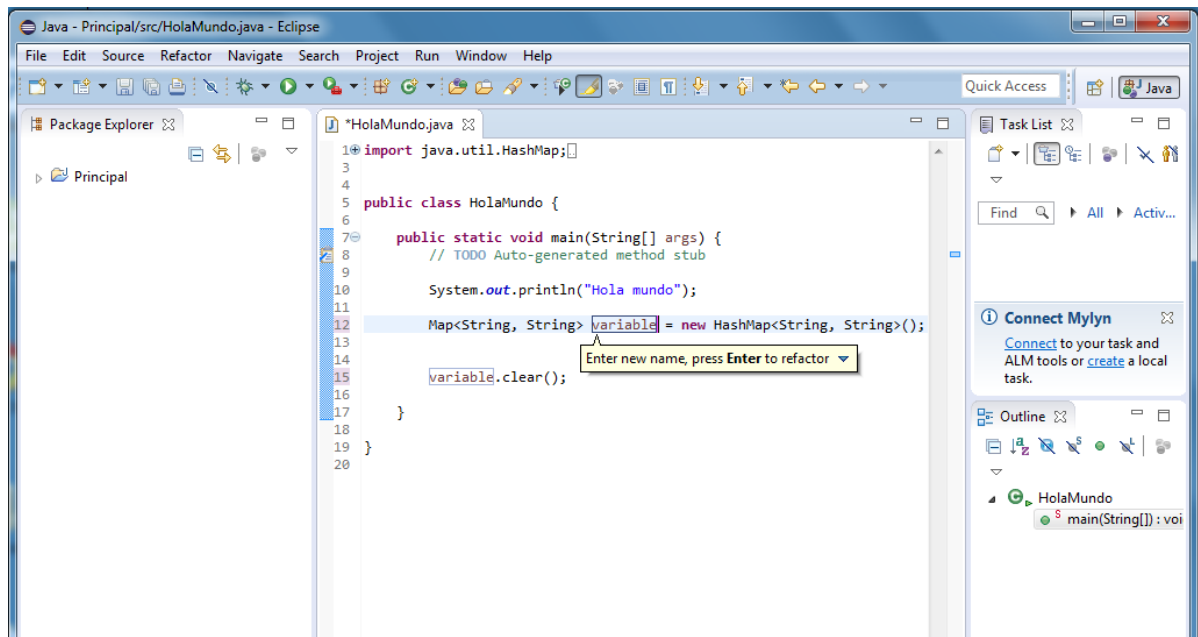
7.10. Refactoring

Consiste en cambiar el nombre de una variable, clase o método mediante una sola acción, es decir, se cambiará el nombre directamente en todas las apariciones de dicha variable, clase o método en el código.

Para ello seleccionamos la variable, clase o método al que queremos cambiarle el nombre. Accedemos al menú contextual "Refactor/Rename".



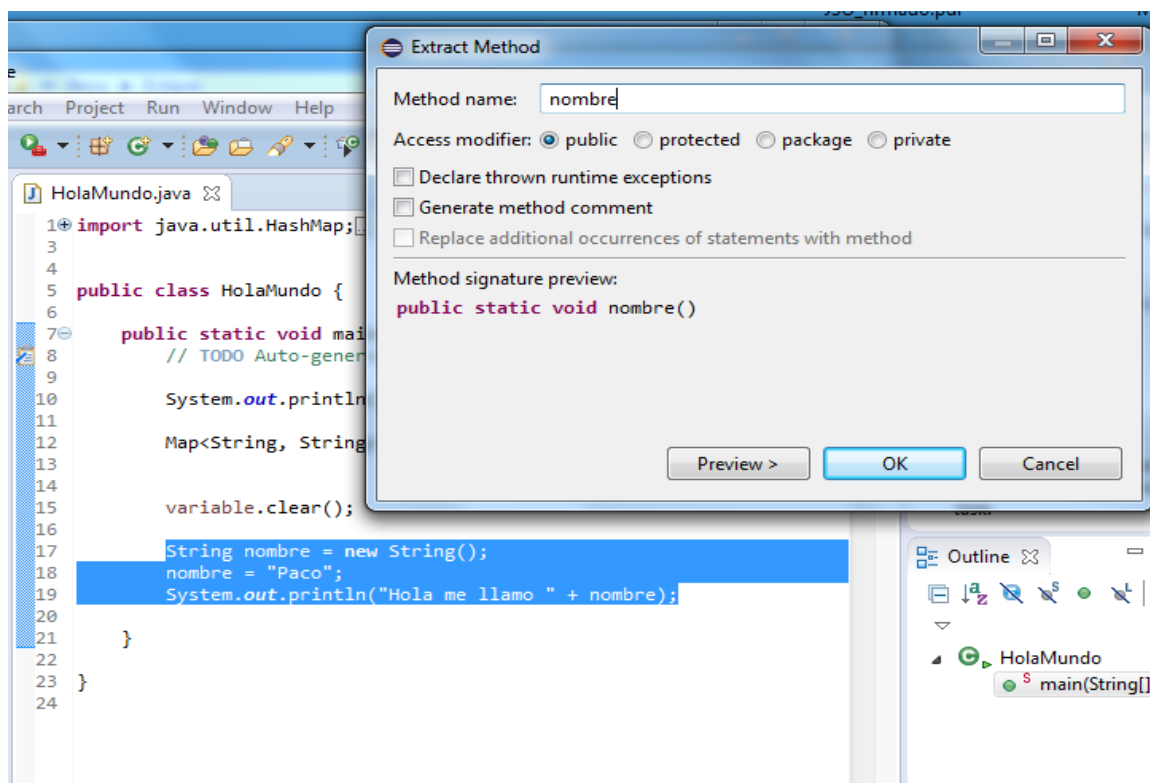
Y a continuación se modifica el nombre de la variable en todas las apariciones.



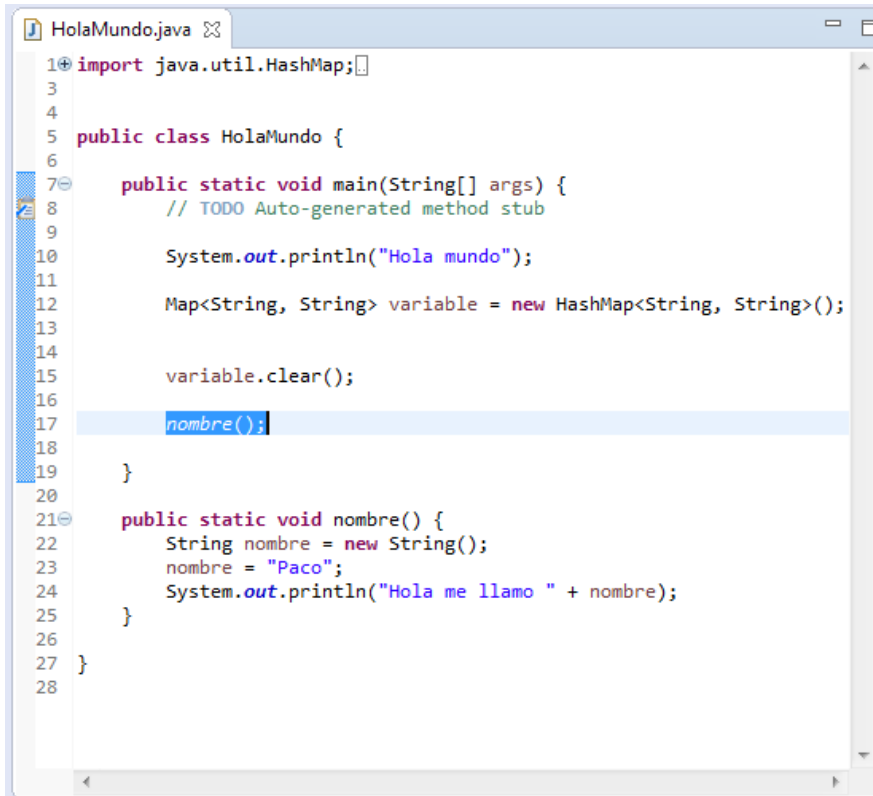
Las opciones disponibles dependen de lo que estamos refactorizando. Por ejemplo, si se trata de un método, podemos seleccionar que mantenga el método original como delegado del método renombrado.

```
20
21
22 System.out.println(n + " x " + m + " = " + mul(n,m));
23
24 sc.close();
25 }
26
27
28
29
30
31 public static double multiplicar(double s2, double s1) {
32     return mul(s2, s1);
33 }
34
35
36
37
38
39 public static double mul(double s2, double s1) {
40
41     return s1*s2;
42 }
43
```

También podemos extraer parte del código y formar un método con él. Para ello seleccionamos dicho código, vamos a menú contextual *"refactor/Extract Method"* y a continuación seleccionaremos el nombre del método, si es público, privado....



Y como vemos, crea el nuevo método, y en el método donde teníamos el código, llama al nuevo método.

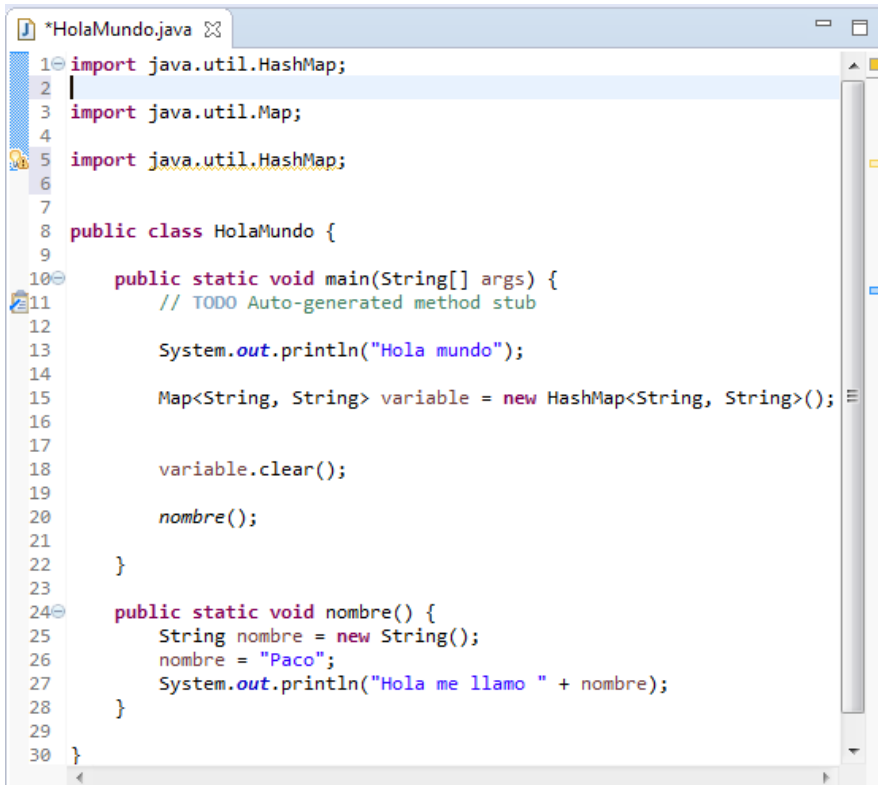


```
1 import java.util.HashMap;
2
3
4
5 public class HolaMundo {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10        System.out.println("Hola mundo");
11
12        Map<String, String> variable = new HashMap<String, String>();
13
14
15        variable.clear();
16
17        nombre();
18    }
19
20
21    public static void nombre() {
22        String nombre = new String();
23        nombre = "Paco";
24        System.out.println("Hola me llamo " + nombre);
25    }
26
27 }
28
```

7.11. Optimizar imports.

Podemos tanto eliminar los *imports* sobrantes o repetidos, como añadir necesarios del siguiente modo, acudimos a menú contextual "source/Organize Imports".

Por ejemplo duplicamos import "java.util.HashMap".



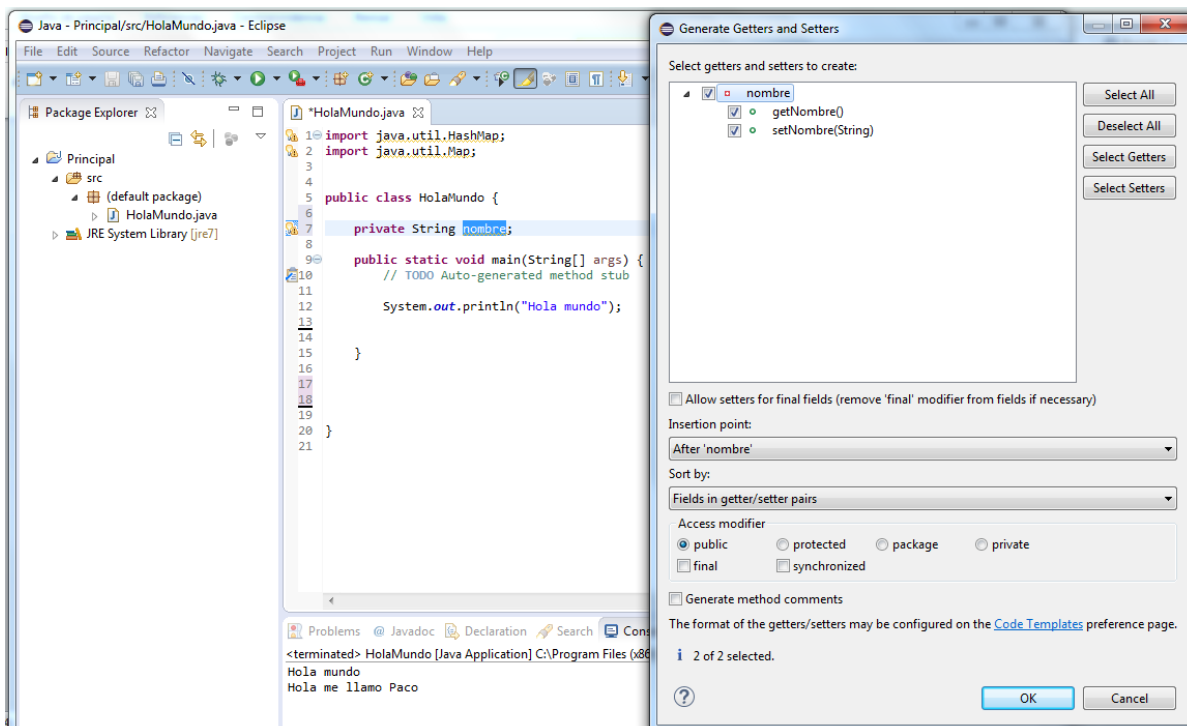
```
1 import java.util.HashMap;
2 import java.util.HashMap;
3 import java.util.Map;
4 import java.util.HashMap;
5
6 public class HolaMundo {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10
11        System.out.println("Hola mundo");
12
13        Map<String, String> variable = new HashMap<String, String>();
14
15        variable.clear();
16
17        nombre();
18    }
19
20
21    public static void nombre() {
22        String nombre = new String();
23        nombre = "Paco";
24        System.out.println("Hola me llamo " + nombre);
25    }
26
27 }
28
```

Y a continuación utilizando la funcionalidad comentada, eliminamos los imports sobrantes.

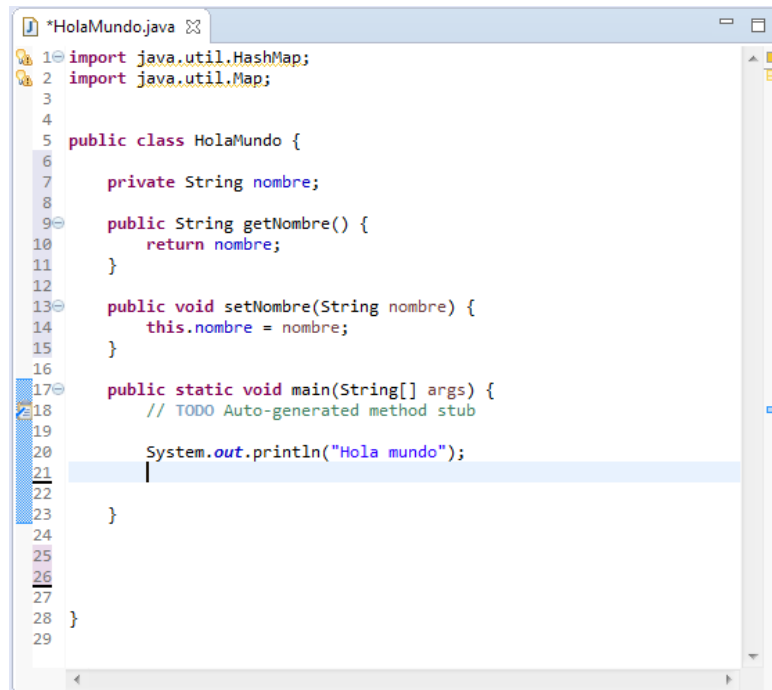
```
*HolaMundo.java
1 import java.util.HashMap;
2 import java.util.Map;
3
4
5 public class HolaMundo {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10        System.out.println("Hola mundo");
11
12        Map<String, String> variable = new HashMap<String, String>();
13
14        variable.clear();
15
16        nombre();
17    }
18
19    public static void nombre() {
20        String nombre = new String();
21        nombre = "Paco";
22        System.out.println("Hola me llamo " + nombre);
23    }
24 }
25
26
27
28
```

7.12. Generar métodos get/set.

Podemos generar de modo rápido y sencillo métodos *get* y/o *set* para un atributo determinado. Para ello seleccionamos el atributo deseado y vamos a menú contextual */source/Generate Getters and Setters*.



A continuación podemos elegir si queremos el método *get*, *set* o ambos, la posición en la que van a aparecer en el código, si van a ser métodos públicos, privados...

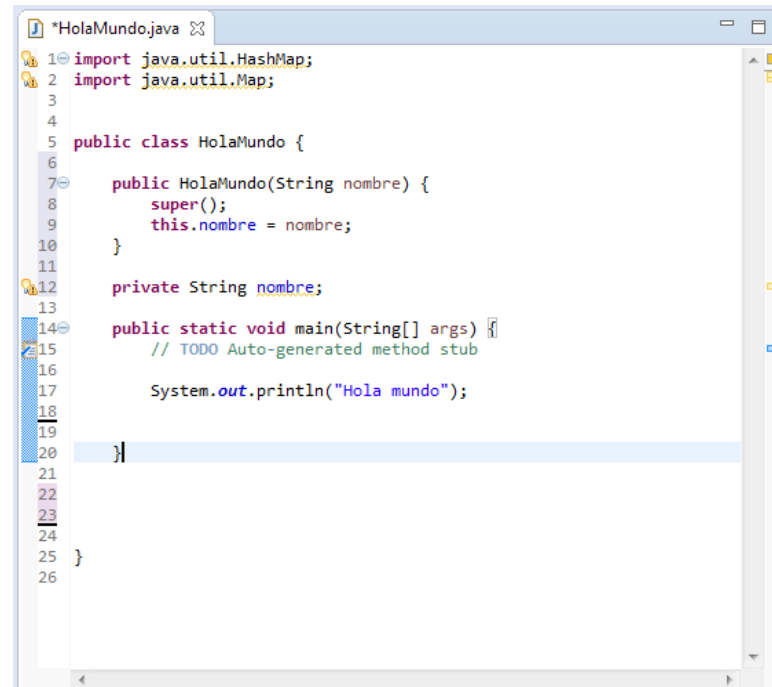


```
1 import java.util.HashMap;
2 import java.util.Map;
3
4
5 public class HolaMundo {
6
7     private String nombre;
8
9     public String getNombre() {
10         return nombre;
11     }
12
13     public void setNombre(String nombre) {
14         this.nombre = nombre;
15     }
16
17     public static void main(String[] args) {
18         // TODO Auto-generated method stub
19
20         System.out.println("Hola mundo");
21     }
22 }
23
24
25
26
27
28
29
```

7.13. Generar constructores.

De modo similar podemos generar un constructor para una clase. Para ello seleccionamos dicha clase, y nos dirigimos al menú contextual "source/Generate Constructor using Fields".

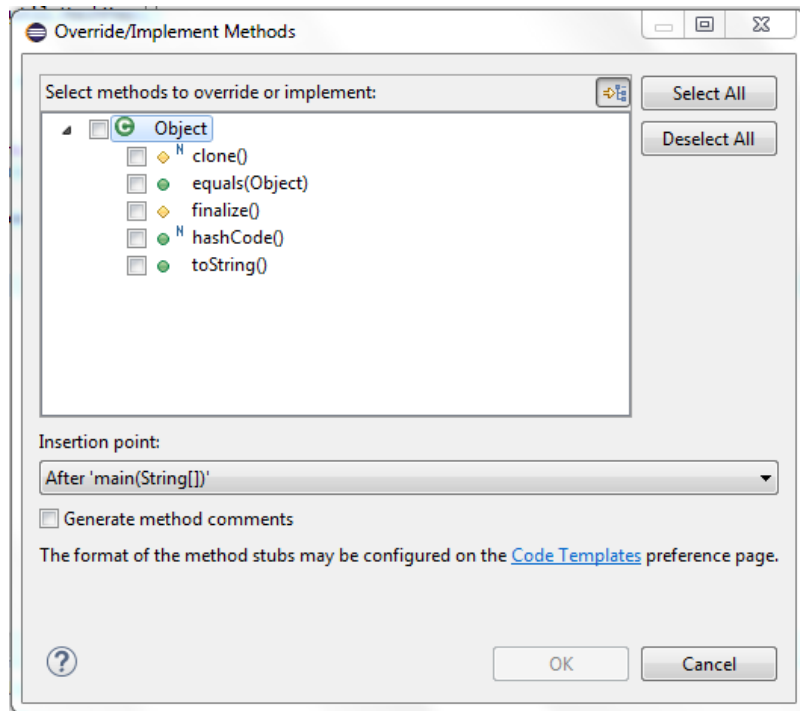
Aparecerá la misma ventana que nos aparece cuando generamos métodos *get/set*. El resultado es el siguiente.



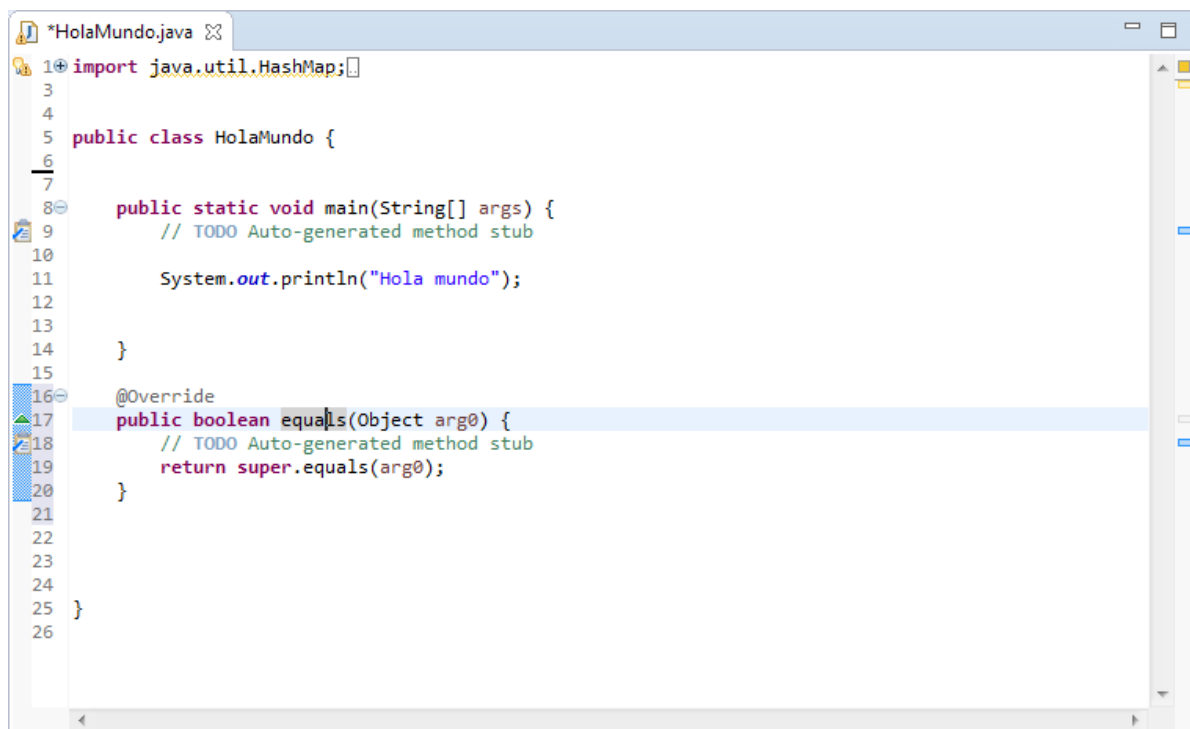
```
1 import java.util.HashMap;
2 import java.util.Map;
3
4
5 public class HolaMundo {
6
7     public HolaMundo(String nombre) {
8         super();
9         this.nombre = nombre;
10     }
11
12     private String nombre;
13
14     public static void main(String[] args) {
15         // TODO Auto-generated method stub
16
17         System.out.println("Hola mundo");
18     }
19 }
20
21
22
23
24
25
26
```

7.14. Incluir el esqueleto de métodos de una superclase o de una interfaz.

También podemos incluir en nuestro programa el esqueleto de un método de una superclase o de una interfaz. Para ello nos dirigimos al menú contextual "source/Override/Implement Methods".



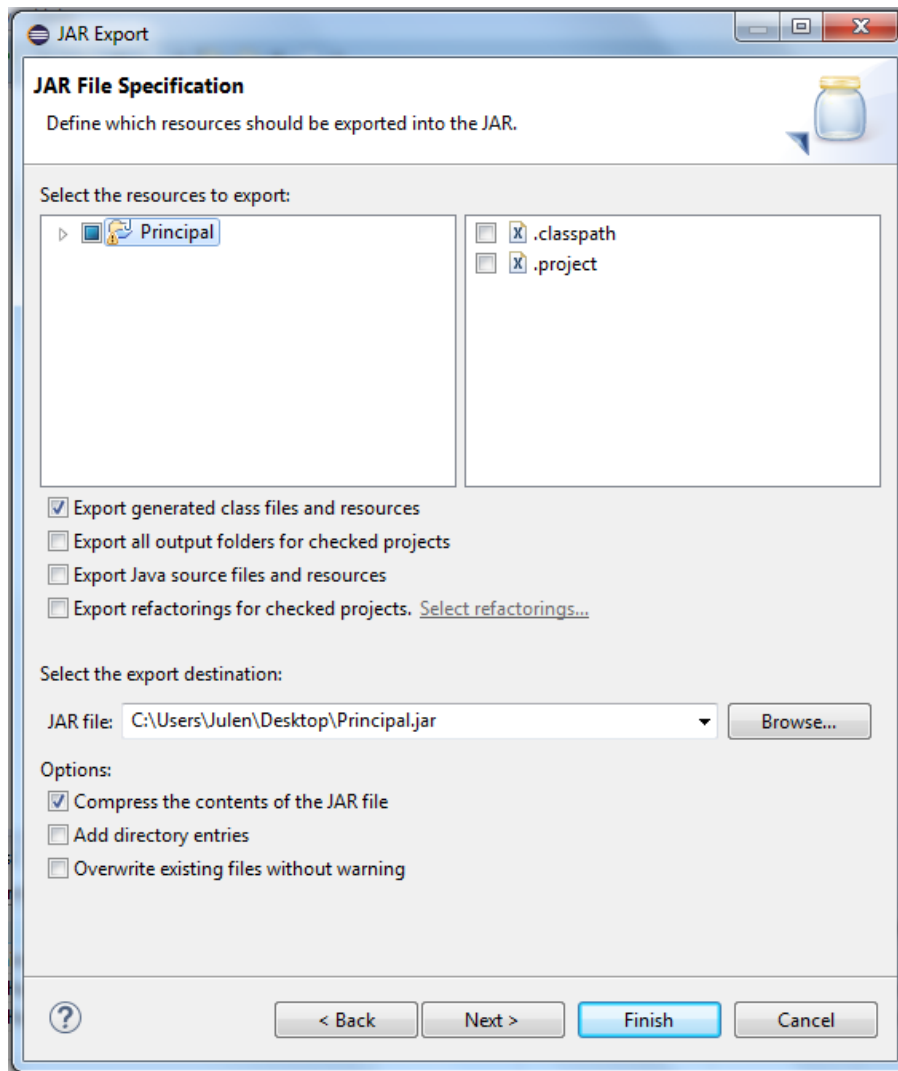
Aparece la ventana anterior, en donde debemos elegir el método que deseamos incluir en nuestro programa. Por ejemplo seleccionemos el método `equals(Object)` de la clase `Object`. Como vemos, se incluye el método a nuestro programa.



7.15. Crear jar.

Podemos exportar un archivo en formato .jar en Eclipse. Para ello en el menú contextual seleccionamos "File/Export/Java/JAR file".

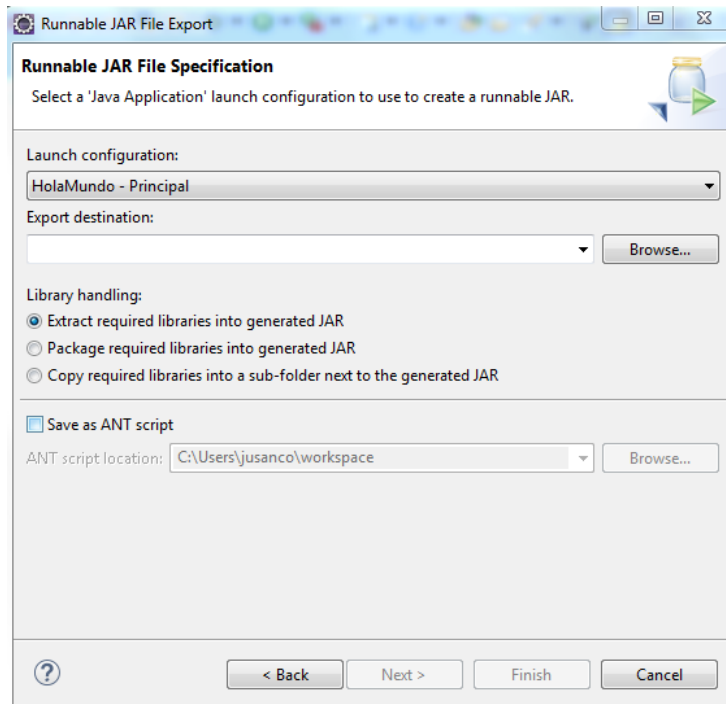
A continuación deberemos seleccionar los ficheros deseados, ubicación de destino, etc...



7.15. Crear jar ejecutable.

Otra opción de la que disponemos en Eclipse, es exportar nuestro programa como un jar ejecutable, es decir, un jar que se puede ejecutar directamente, por ejemplo, haciendo doble clic sobre el o ejecutándolo en la consola.

Para exportar nuestro proyecto de este modo, acudimos a *"File/Export/Java/Runnable JAR file"*. De modo similar a como acabamos de hacer, seleccionamos donde queremos guardar el jar generado y seleccionamos la clase con el método principal. Y ya tenemos creado nuestro archivo .jar ejecutable.



8. Depuración de programas

Eclipse facilita la depuración de código, es decir, nos permite ejecutar paso a paso el código y analizar el contenido de las variables.

Para depurar un programa debemos lograr que el depurador se detenga en algún punto del código. A partir de ese lugar podemos empezar a controlar la ejecución paso a paso. Por eso, lo primero que hay que hacer para depurar un programa es definir en qué punto o puntos del código queremos que la ejecución del programa se pause (**puntos de ruptura o breakpoints**). Para ello nos situamos en la línea deseada y hacemos doble-clic o clic derecho en la barra a la izquierda de la ventana del código, apareciendo en dicha barra un punto.

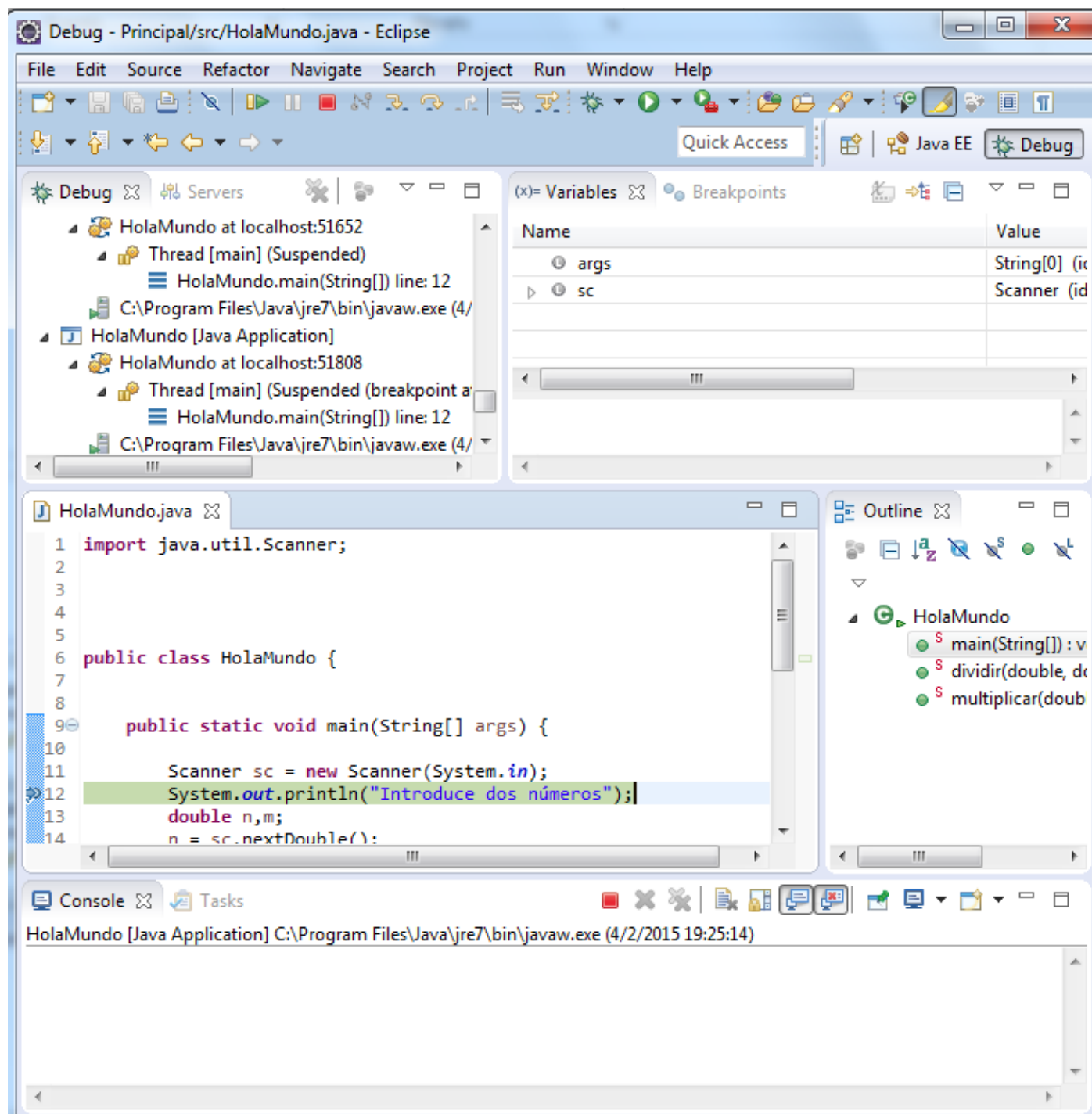
```

5
6 public class HolaMundo {
7
8
9     public static void main(String[] args) {
10
11         Scanner sc = new Scanner(System.in);
12         System.out.println("Introduce dos números");
13         double n,m;
14         n = sc.nextDouble();
15         m = sc.nextDouble();
16
17         System.out.println(n + " / " + m + " = " + dividir(n,m));
18         System.out.println(n + " * " + m + " = " + multiplicar(n,m));
19         sc.close();
20     }
21
22     public static double dividir(double s1, double s2) {
23         return s1/s2;
24     }
25
26     public static double multiplicar(double s1, double s2) {
27         return s1*s2;
28     }
29 }
30
31

```

Una vez tenemos definidos los puntos de ruptura deseados, iniciamos el depurador pulsando el botón  .

Lo primero que vemos es que Eclipse cambia la perspectiva Java a la perspectiva Debug, en donde además del código fuente, podemos ver la información de las variables en cada momento y una ventana donde se muestran los hilos de ejecución.



Con la intención de seguir la pista a las modificaciones que va realizando el programa a las variables, podemos indicar que la ejecución siga o se detenga a nuestro gusto, utilizando los siguientes botones.



Resume: Continúa la ejecución del programa hasta el próximo punto de ruptura o hasta que finaliza la ejecución.



Terminate: Finaliza la ejecución del programa.



Step into: Se ejecuta la línea actual y, en caso de ser una llamada a un método, la ejecución continúa dentro del método.



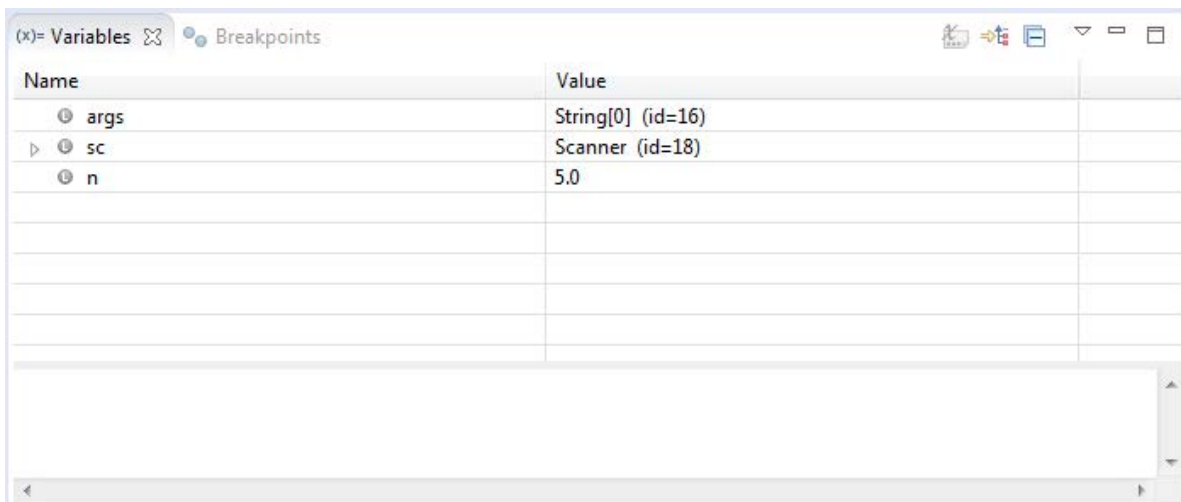
Step over: Se ejecuta la línea actual y se pasa a la línea siguiente sin entrar en los métodos.



Step return: Se sigue ejecutando hasta que se ejecute un *return*. Es decir, se ejecuta por completo el método en el que estamos y se pasa a la línea siguiente a la invocación del método.

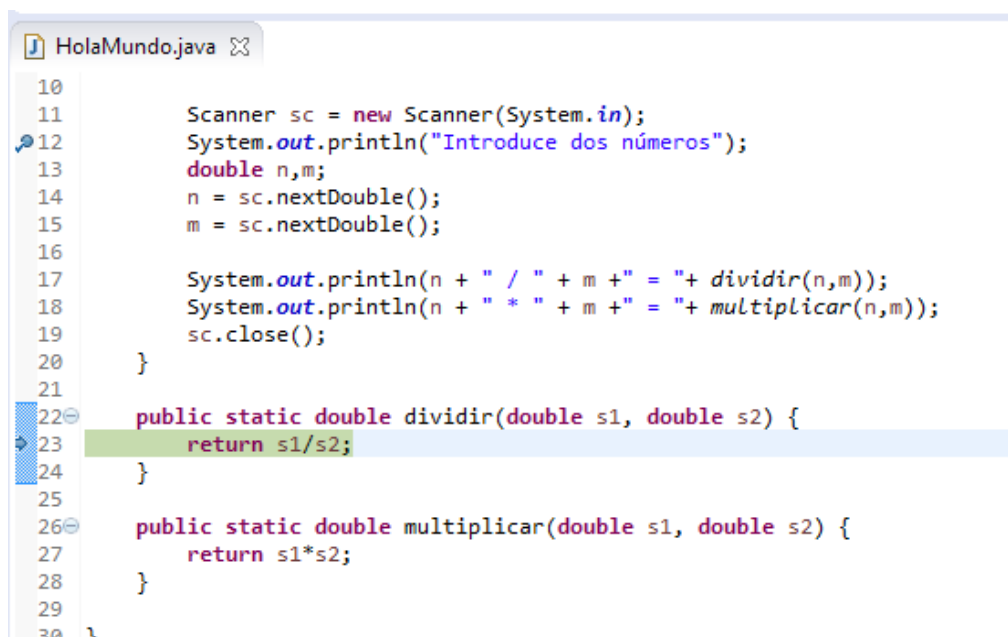
Por ejemplo, depuremos el último programa que hemos visto. Nos situamos en la perspectiva Debug.

A continuación vamos ejecutando el programa línea a línea mediante la opción que acabamos de comentar, "Step into". Avanzamos hasta la línea en la que introducimos un valor para la variable "n".

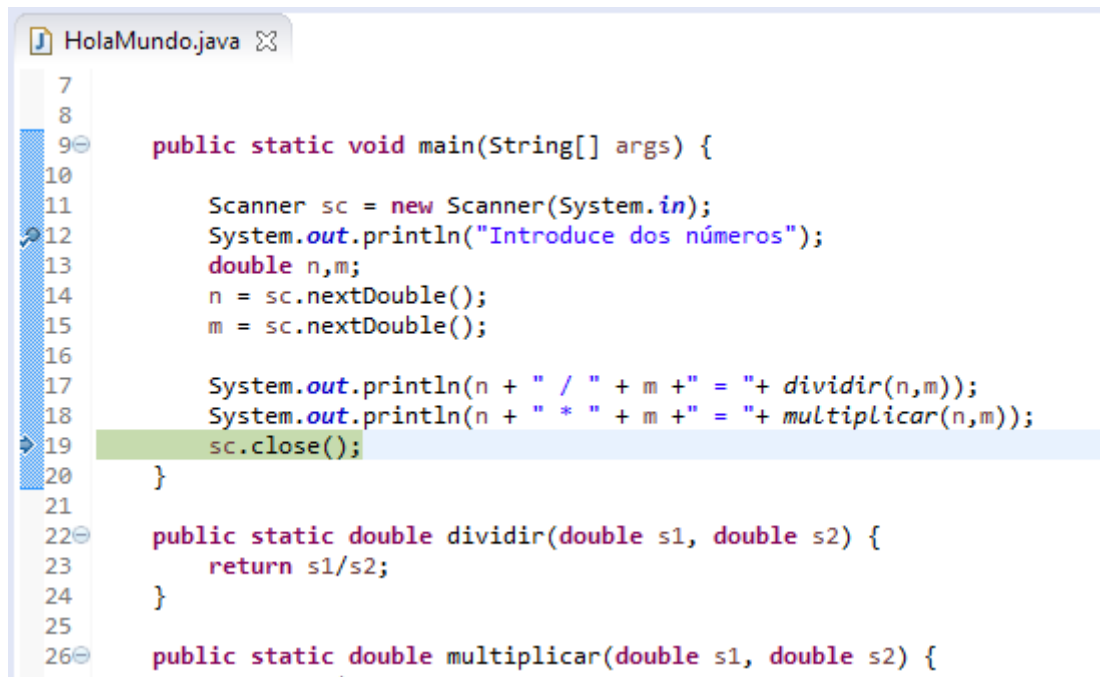


Vemos como en la pestaña que nos muestra las variables creadas, ya disponemos de "n" y "sc".

Continuamos ejecutando el programa paso a paso mediante la opción "Step into", por lo que, como ya hemos comentado, al llegar a la línea que llama al método `dividir(n,m)`, la ejecución continúa dentro del método.



Notad que Eclipse sombrea la línea de código en la que nos encontramos. A continuación seguimos ejecutando el código mediante la opción "Step over", por lo que al llegar a la línea que llama al método `multiplicar(n,m)`, en este caso se ejecutará la línea, pero no entraremos en dicho método.



```
7
8
9 public static void main(String[] args) {
10
11     Scanner sc = new Scanner(System.in);
12     System.out.println("Introduce dos números");
13     double n,m;
14     n = sc.nextDouble();
15     m = sc.nextDouble();
16
17     System.out.println(n + " / " + m + " = " + dividir(n,m));
18     System.out.println(n + " * " + m + " = " + multiplicar(n,m));
19     sc.close();
20 }
21
22 public static double dividir(double s1, double s2) {
23     return s1/s2;
24 }
25
26 public static double multiplicar(double s1, double s2) {
```

Por último, una vez queráis volver a la perspectiva java, tendréis que acudir a la parte derecha de la pantalla, arriba, donde podéis elegir entre ambas perspectivas:



9. Referencias:

Eclipse IDE Tutorial.

<http://www.vogella.com/articles/Eclipse/article.html>

Otro Tutorial de Eclipse.

[http://dis.um.es/~bmoros/privado/bibliografia/tutorial%20eclipse%20para%20novatos%20java%20\(Pollino\).pdf](http://dis.um.es/~bmoros/privado/bibliografia/tutorial%20eclipse%20para%20novatos%20java%20(Pollino).pdf)

Java Debugging with Eclipse Tutorial.

<http://www.vogella.com/articles/EclipseDebugging/article.html>