

Taller Tema 2: Eclipse



Introducción

En este taller práctico vamos a conocer la programación a través de uno de los entornos de programación más utilizados actualmente: **Eclipse**.



Empezaremos viendo sus características principales para pasar inmediatamente a realizar nuestros primeros proyectos en JAVA.

¿Qué es Eclipse?

La mayoría de la gente conoce **Eclipse** como un **entorno de desarrollo integrado** (IDE) para Java. Hoy en día es el entorno de desarrollo para Java líder con una cuota de mercado de aproximadamente 65%.

Eclipse es creado por una comunidad de código abierto y se utiliza en varias áreas diferentes, por ejemplo, como un entorno de desarrollo para aplicaciones Java o Android. Eclipse se remonta a 2001.

La comunidad de código abierto Eclipse cuenta con más de 200 proyectos de software libre que cubren diferentes aspectos del desarrollo de software.

Los proyectos de Eclipse se rigen por la *Fundación Eclipse*. La *Fundación Eclipse* es una organización sin fines de lucro, corporación miembro que alberga los

proyectos de Eclipse de código abierto y ayuda a cultivar a la vez una comunidad de código abierto y un ecosistema de productos y servicios complementarios.

El IDE de Eclipse se puede ampliar con los componentes de software adicionales. Eclipse llama a estos componentes de software *plug-ins*. Por ejemplo, para programar en Android necesitamos el plug-in que nos permitirá utilizar Eclipse para programar en dicho sistema. Varios proyectos de código abierto y compañías han ampliado el IDE Eclipse. Y crece día a día.

También es posible utilizar Eclipse como base para la creación de aplicaciones de propósito general. Estas aplicaciones son conocidas como Eclipse Rich Client Platform (*Eclipse RCP*) aplicaciones.

Eclipse Public License

La *Licencia Pública Eclipse* (EPL) es una licencia de software de código abierto utilizado por la *Fundación Eclipse* para su software. El EPL está diseñado para los negocios. Los programas bajo licencia EPL pueden ser utilizados, modificados, copiados y distribuidos de forma gratuita y el receptor del EPL con licencia de software puede optar por utilizar este software en programas de código cerrado.

La *Fundación Eclipse* también valida que el código fuente contribuido a proyectos de Eclipse está libre de la propiedad intelectual (PI). Este proceso se conoce como la limpieza de PI.

El carácter permisivo del EPL y el esfuerzo de limpieza PI de la *Fundación Eclipse* hace que la reutilización del código fuente de proyectos de Eclipse sea de lo más atractivo.

Instalación de Eclipse

Los requisitos de Java de Eclipse

Eclipse requiere Java instalado en tiempo de ejecución. Consultar la documentación actualizada para conocer qué versión de Java mínimo requiere la versión de Eclipse con la que se vaya a trabajar.

La instalación de Java

Para la correcta ejecución de Eclipse se hace necesario el tener instalado JAVA en nuestro equipo de desarrollo. En la mayoría de las webs se hace necesario la instalación de JAVA, con lo cual, lo más seguro es que ya lo tengas instalado.

Para comprobarlo, en el disco duro, en Archivos de Programas debe de haber una carpeta llamada JAVA en sistemas Windows.

Para equipos Apple, los últimos sistemas operativos viene de serie con JAVA instalado. Lo único que hay que comprobar es que está actualizada la versión de JAVA a la última disponible.

Para Linux, ocurre también que la mayoría de las últimas distribuciones llevan JAVA instalado de serie. Comprobarlo en el gestor de paquetes para ver que tienen la última versión.

En caso negativo, se puede descargar desde Internet, de repositorios, ...

Descargar Eclipse

El sitio web Eclipse.org (<http://eclipse.org/downloads/>) ofrece distintas distribuciones de Eclipse. Nosotros utilizaremos el paquete “Eclipse IDE for Java Developers”. En las últimas versiones, ya sí hay un “Installer”.

La siguiente imagen muestra el sitio web de descarga de Eclipse para un sistema Windows:

The screenshot shows the Eclipse website's download page for Windows. At the top, there's a navigation bar with links like 'GETTING STARTED', 'MEMBERS', 'PROJECTS', and 'MORE'. Below this, a breadcrumb trail reads 'HOME / DOWNLOADS / ECLIPSE DOWNLOADS'. The main content area features a large banner for 'Eclipse Neon (4.6) Release for Windows' with a 'Try the Eclipse Installer' button and a download count of 1,205,132. Below the banner, three specific IDE packages are listed: 'Eclipse IDE for Java EE Developers', 'Eclipse IDE for Java Developers', and 'Eclipse IDE for C/C++ Developers', each with its own download count and bit options (32 bit | 64 bit). On the right side, there are sections for 'RELATED LINKS' (including 'Compare & Combine Packages', 'New and Noteworthy', 'Install Guide', 'Documentation', 'Updating Eclipse', and 'Forums') and 'MORE DOWNLOADS' (listing 'Other builds' like Eclipse Neon (4.6), Eclipse Mars (4.5), Eclipse Luna (4.4), and Eclipse Venier (4.3)).

La descarga es un .zip de archivos.

Instalar Eclipse

Una vez descargado el archivo .zip que contiene la distribución de Eclipse hay que descomprimirlo en un directorio local.

Utilice una ruta de directorio que no contenga espacios en su nombre, ya que Eclipse tiene a veces problemas con esos caracteres.

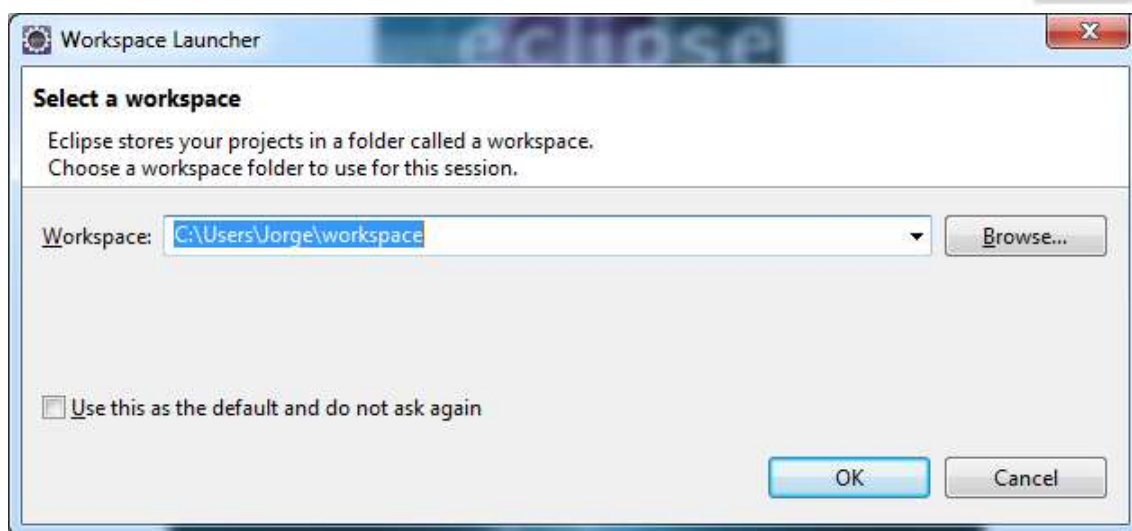
Después de descomprimir el archivo zip descargado, Eclipse está listo para ser utilizado, sin procedimiento de instalación adicional.

Primeros pasos

Iniciar Eclipse

Para iniciar Eclipse, haga doble clic en el archivo eclipse.exe (Microsoft Windows) o eclipse (Linux / Mac) en el directorio donde ha desempaquetado Eclipse.

El sistema, en primer lugar, le pedirá indicar un *espacio de trabajo*. Es el lugar en el que usted trabaja con sus proyectos JAVA. En este directorio se guardarán todos los archivos que se vayan creando a medida que vayamos creando programas y aplicaciones. Seleccione un directorio vacío y pulse el botón OK.

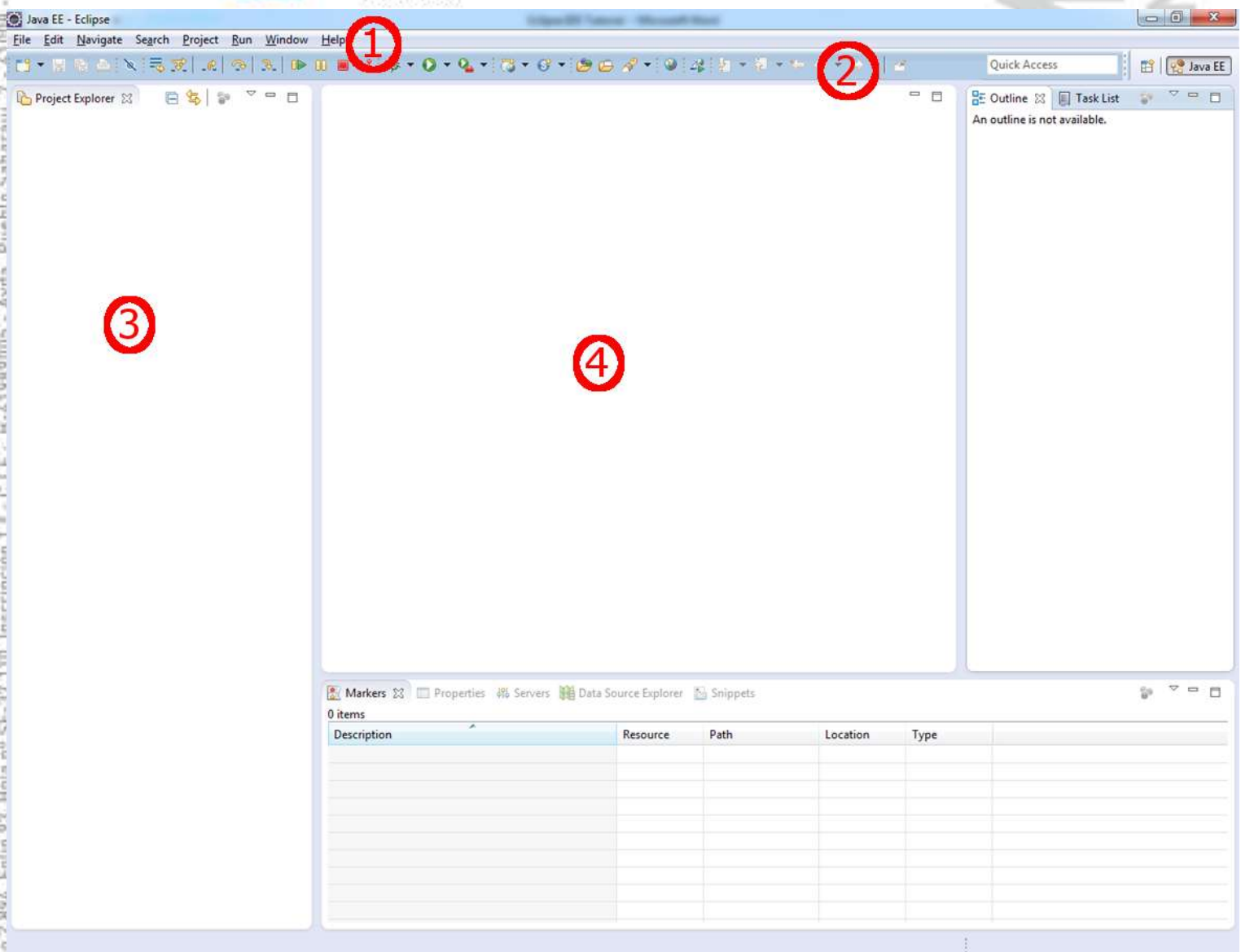


Podemos marcar la casilla de abajo para que no nos pregunte cada vez que entremos la ubicación de nuestro espacio de trabajo.

A continuación nos aparece la pantalla de Bienvenida de la instalación en el que nos podemos encontrar, entre otras cosas, un Vistazo (Overview) de las principales características de la actual versión de Eclipse, unos Ejemplos (Samples), unos Tutoriales (Tutorials) y las últimas novedades (What's New).



Si pulsamos en “Banco de Trabajo” (Workbench) nos iremos a la pantalla principal de nuestro entorno de desarrollo.



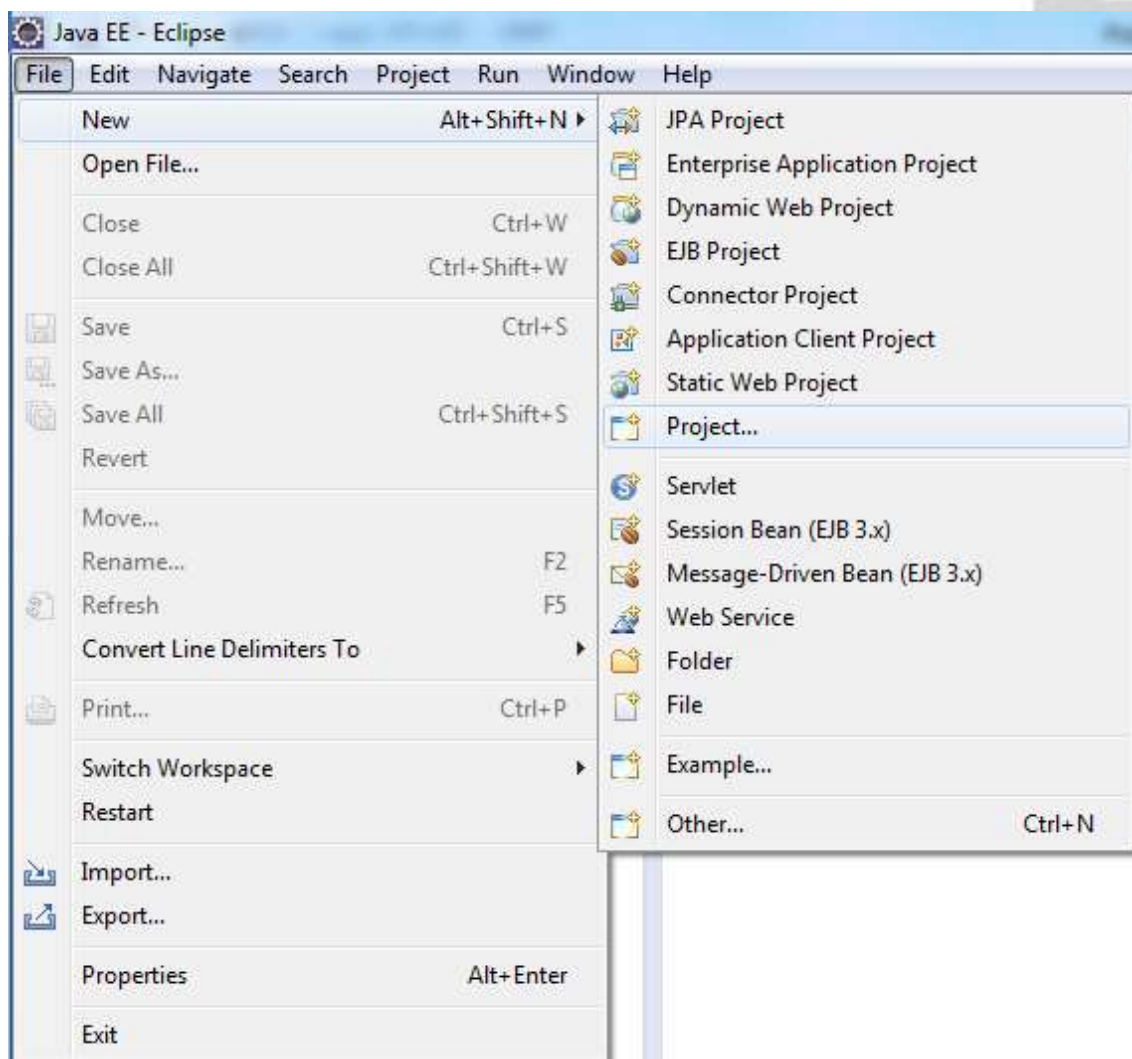
En esta pantalla aparecen, entre otros, la Barra de Menú (1), la Barra de Herramientas (2), el Explorador de Proyectos (3) y el Editor de código (4).

Primer programa

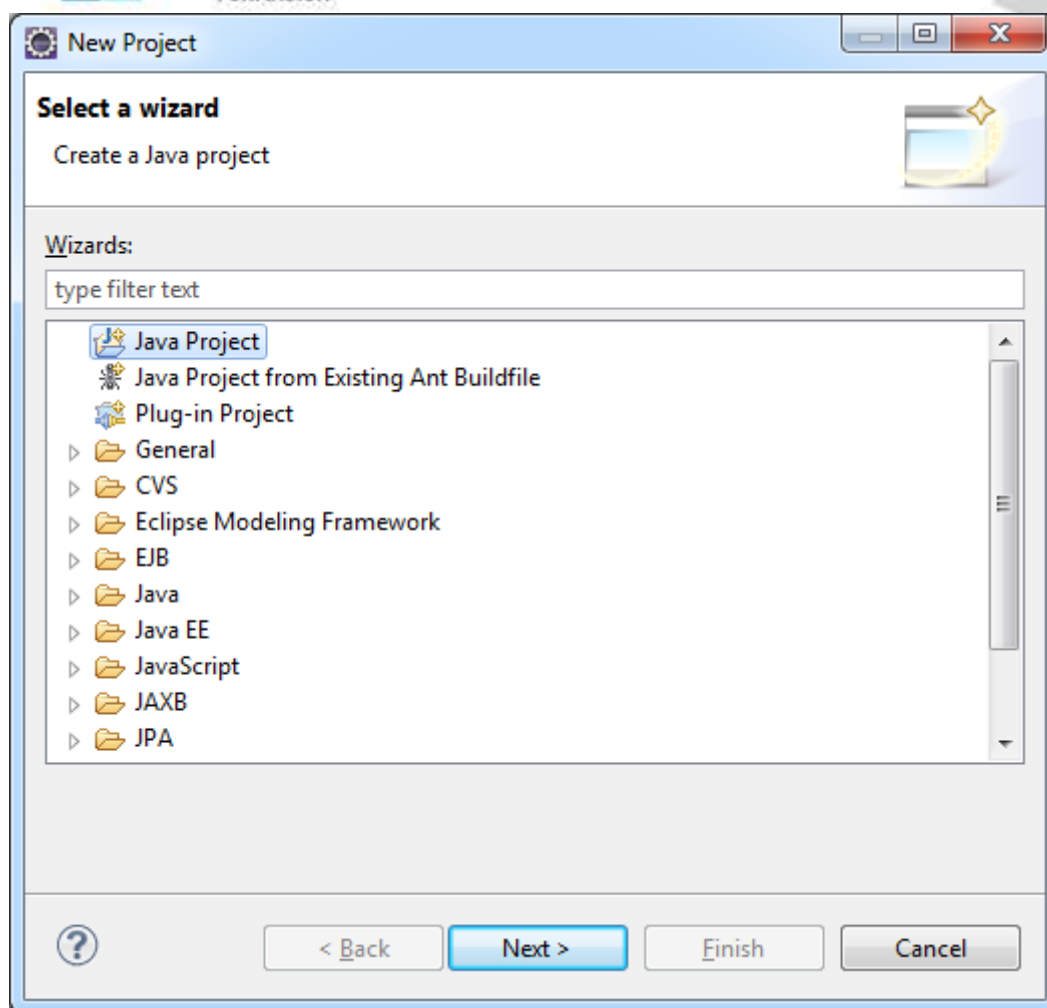
Históricamente, el primer programa que se suele hacer en todo nuevo lenguaje de programación que empezamos a aprender es el Hola mundo! Y no vamos a ser menos.

Para programar con Eclipse, todo debemos incluirlo dentro de un proyecto. Es decir, un programa, un proyecto. Así que veamos cómo se crean los proyectos.

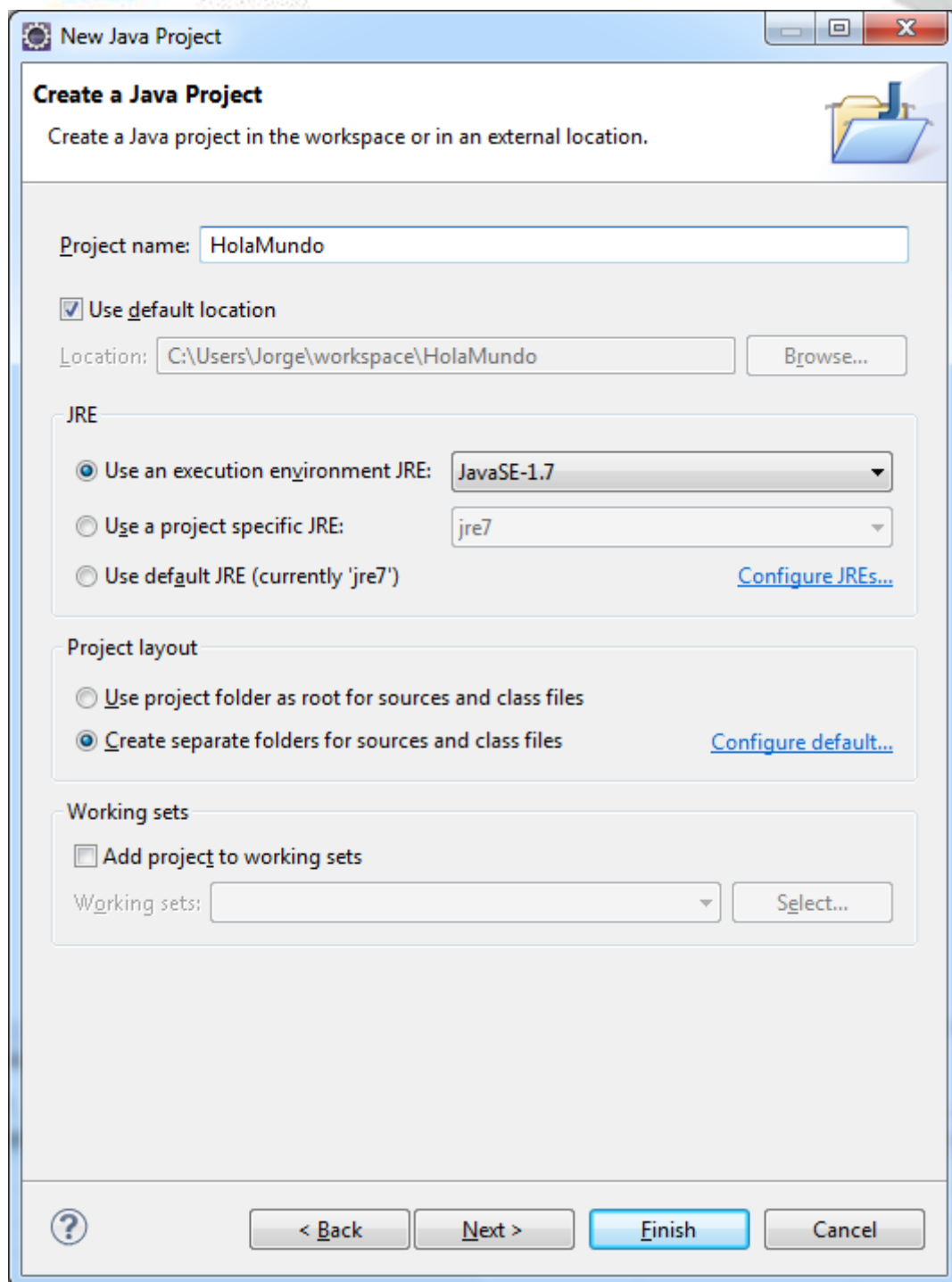
Seleccionamos File → New → Project



Ahora seleccionamos “Java Project” (Os recuerdo que este entorno Eclipse nos permite realizar multitud de tipos de proyectos y nos permite trabajar en diversos lenguajes de programación):



Y pulsamos "Next"...

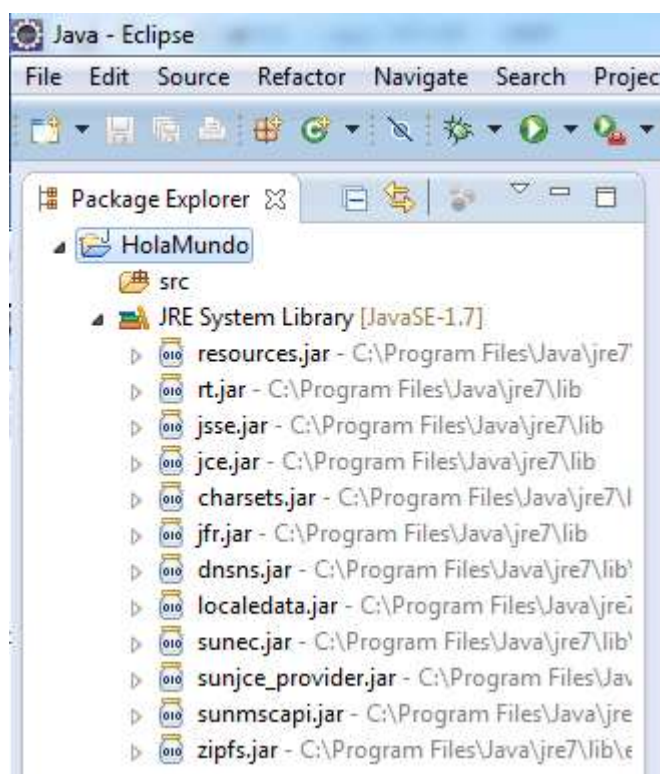


En esta pantalla pondremos el nombre del proyecto: “HolaMundo”. OJO: Evitar espacios en blanco, tildes, eñes,... Y pulsamos en “Finish”.

Se nos advierte de que por el tipo de proyecto elegido, en el entorno nos aparecerán las perspectivas asociadas. Contestaremos afirmativamente.

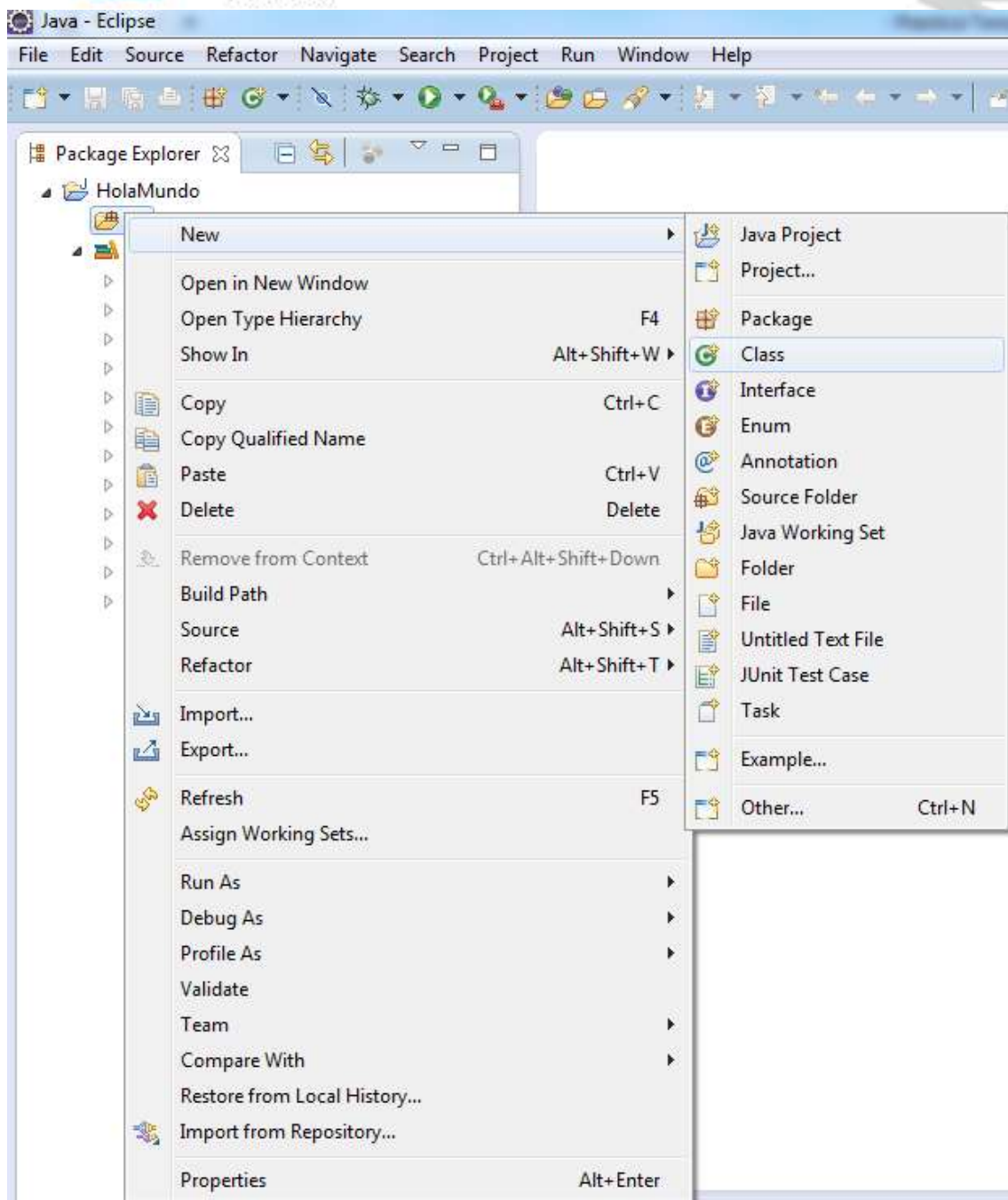


Se nos crea una estructura en árbol en la parte izquierda con los recursos que se han creado de forma automática (Explorador de paquetes). En esta pantalla o perspectiva, se nos mostrarán los ficheros, recursos, clases, librerías, etc. de nuestros proyectos. De momento aparece una carpeta llamada HolaMundo, el nombre de nuestro proyecto. Si pulsamos en el pequeño triángulo que aparece a su



izquierda, se nos van desplegando los elementos que componen nuestro proyecto. En este caso otra carpeta llamada src, y una Librería del Sistema con muchos recursos.

Vamos a crear nuestro fichero .java con el código fuente. Vamos a pulsar con el botón derecho del ratón sobre la carpeta llamada src y elegiremos la opción New y luego Class.

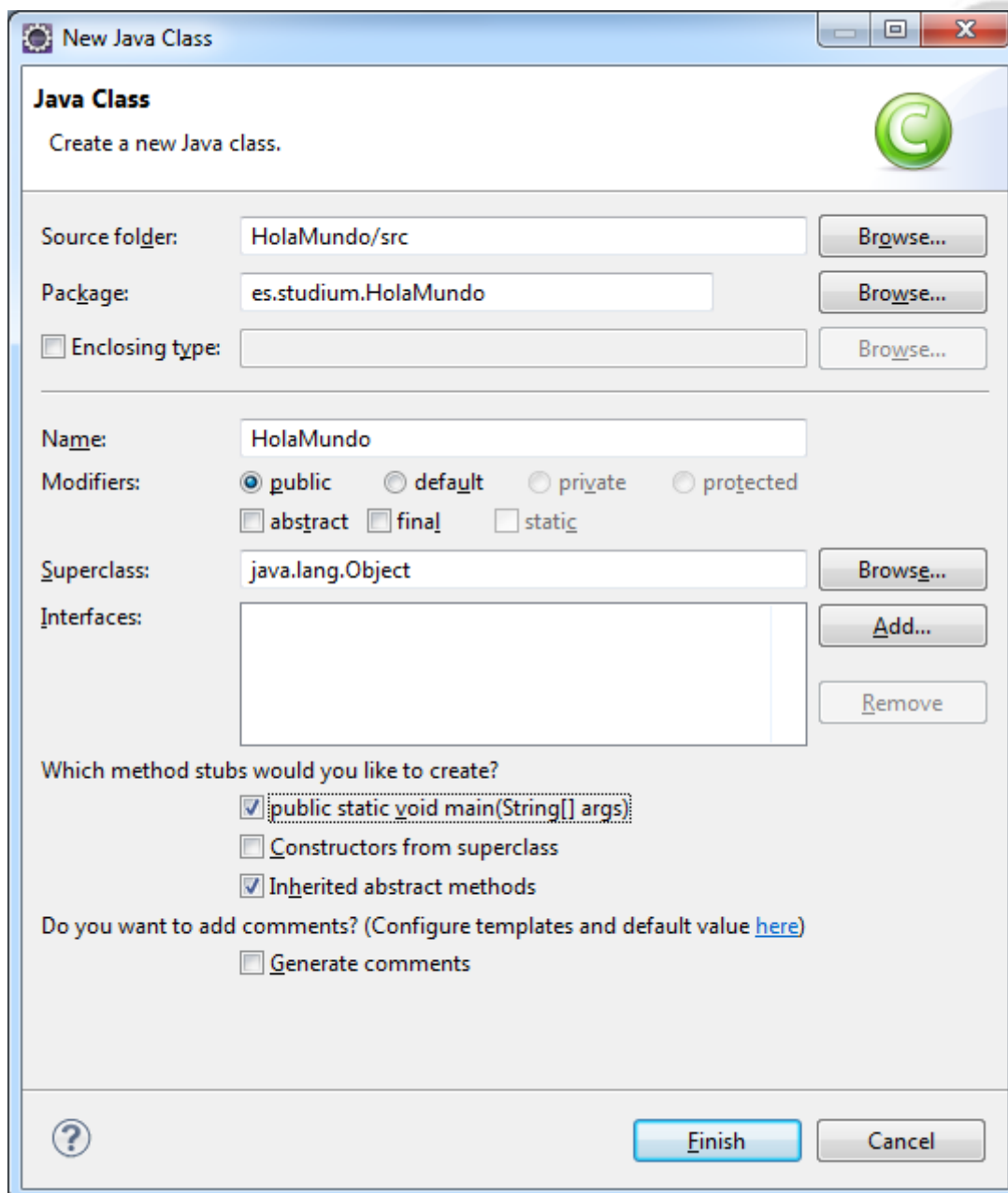


Vamos a ponerle un nombre al paquete al que pertenece esta clase que vamos a crear. Los paquetes nos van a servir para agrupar ficheros, clases, etc. Profundizaremos en este tema más adelante. De momento indicaremos un nombre tal que “es.studium.HolaMundo”.

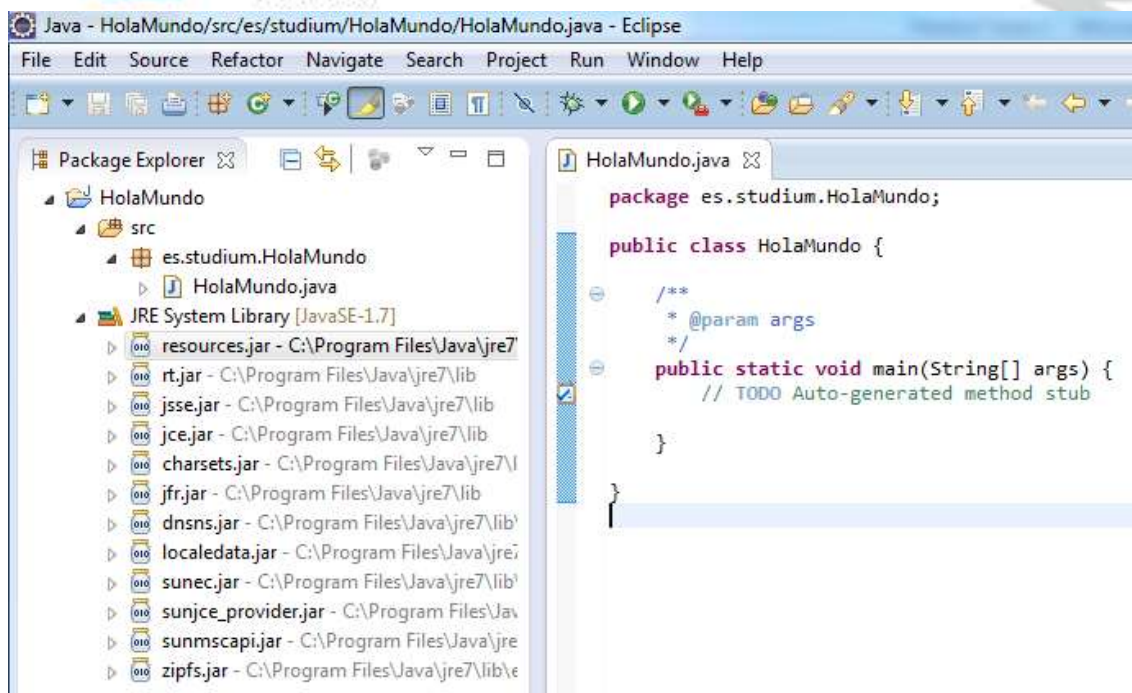
A continuación, se nos preguntará por el nombre del archivo. **MUY IMPORTANTE:** El nombre del fichero debe coincidir con el nombre de la Clase que va a contener, así que como vamos a trabajar con la clase HolaMundo, le ponemos

el mismo nombre al fichero. En este caso coincide con el nombre del Proyecto, pero no tiene que ser así siempre. Aquí lo hacemos por la simplicidad del ejemplo.

Por último, marcaremos la opción “public static void main(String[] args)” que nos evitará tener que escribirlo en nuestra clase. Más adelante veremos cuándo no es necesario usarlo, pero por ahora lo marcaremos SIEMPRE.



¿Ya estamos? Pues pulsamos el botón “Finish”.



Nos aparecen varias cosas:

- En el Explorar de Paquetes, dentro de la carpeta src, nos aparece un paquete nuevo llamado “es.studium.HolaMundo” y dentro de él, un fichero llamado “HolaMundo.java”
- Y en el editor, nos aparece el contenido de dicho fichero como indica en su pestaña superior.



Indicar que no siempre nos aparecerá este código, pero si algo parecido. Lo modificamos de manera que quede como sigue:

```
package es.studium.HolaMundo;

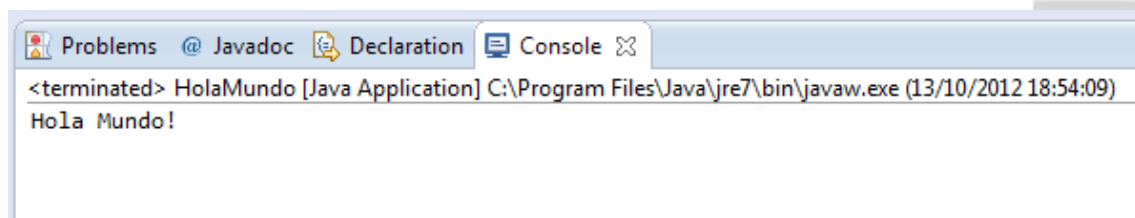
public class HolaMundo
{
    /**
     * @param args
     */
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub
        System.out.println("Hola Mundo!");
    }
}
```

Fijarse bien en hemos desplazado los corchetes de apertura hacia abajo a la derecha para que tanto el de apertura como el de cierre queden alineados en la

misma columna y sea más fácil editar el código y saber dónde empieza un bloque de código y dónde acaba dicho bloque. Funcionaría igualmente, pero de esta forma, trabajar con el código se hace más fácil.

Lo guardamos, lo ejecutamos y luego explicamos. Para guardar, pulsar en la barra de herramientas en el disco  o "Save". Para ejecutar, pulsaremos en el botón "Run" .

Se abre una nueva perspectiva en la parte inferior llamada "Console" con la ejecución de nuestro programa:



Pasemos a describir detalladamente las líneas de nuestro primer programa.

```
package es.studium.HolaMundo;
```

Indica el nombre del paquete en el que está englobado este programa. Cuando realicemos proyectos más complejos, con muchos ficheros y clases, etc. y nos interese que interactúen entre ellos, deberán estar en el mismo paquete.

```
public class HolaMundo
```

Empieza nuestra clase HolaMundo. Su ámbito abarca desde la llave que viene a continuación hasta la última llave que cierra nuestro fichero. Recordad que nuestro fichero se debe llamar igual que la clase que contiene.

```
public static void main(String[] args)
```

Es la función principal de nuestro programa. La ejecución del programa empezará con el código que haya dentro de esta función main.

```
System.out.println("Hola Mundo!");
```

Esta instrucción llama a la función println con el parámetro "Hola Mundo!" para que lo muestre por la salida estándar, que en nuestro caso inicialmente es la "Consola". La utilización de esta función implica el uso de System.out.println

indicando que *println* está dentro de *out* y ésta a su vez está en *System*. Ambas son librerías incluidas por defecto en cualquier proyecto JAVA.

Configurar el entorno

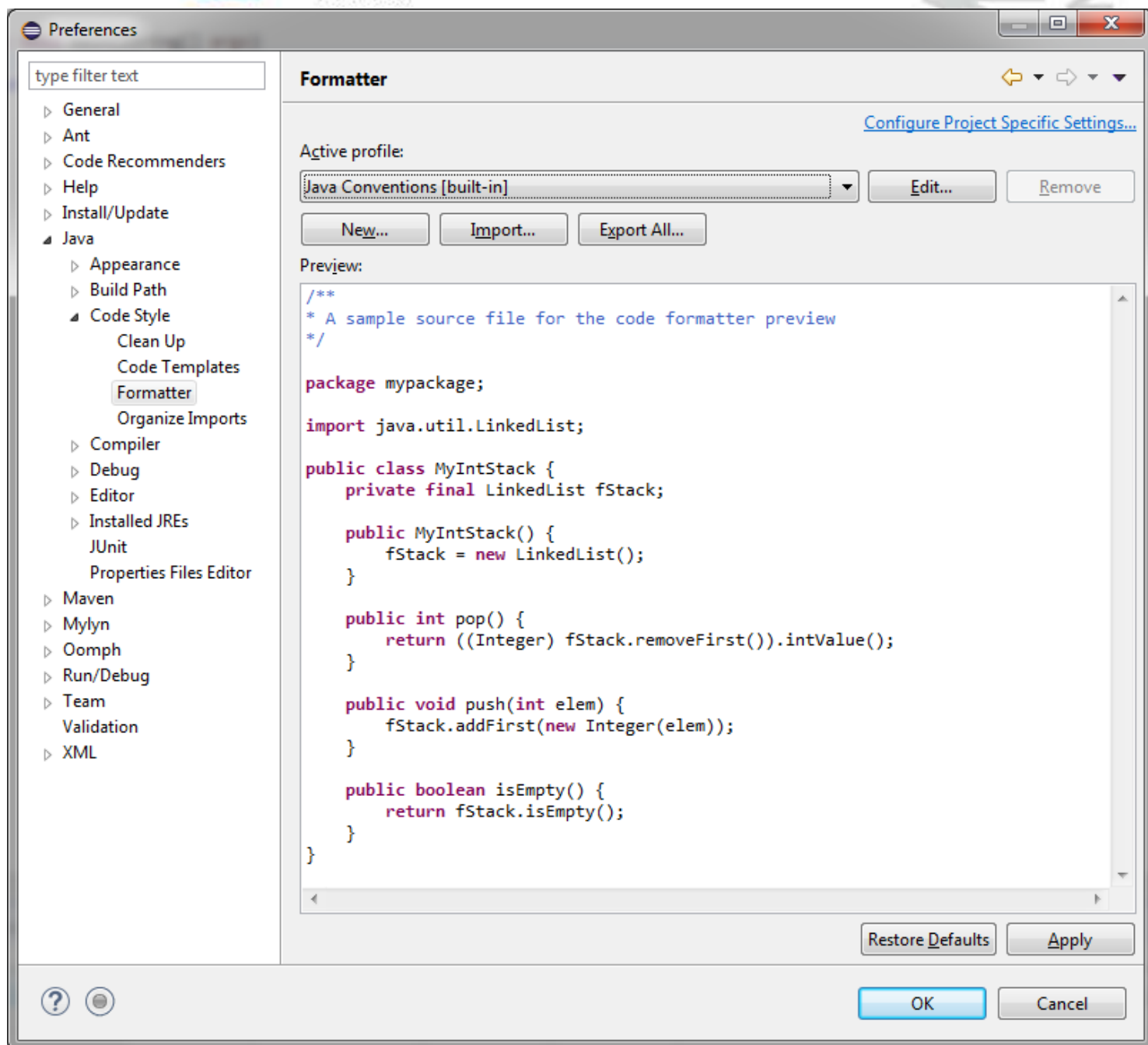
Una de las primeras tareas que debemos afrontar al programar en Eclipse es configurar el editor para que el código que escribamos tenga el formato deseado.

Si miramos por internet, nos podemos encontrar multitud de formas de formatear nuestro código: Indentación, saltos de línea, corchetes,...

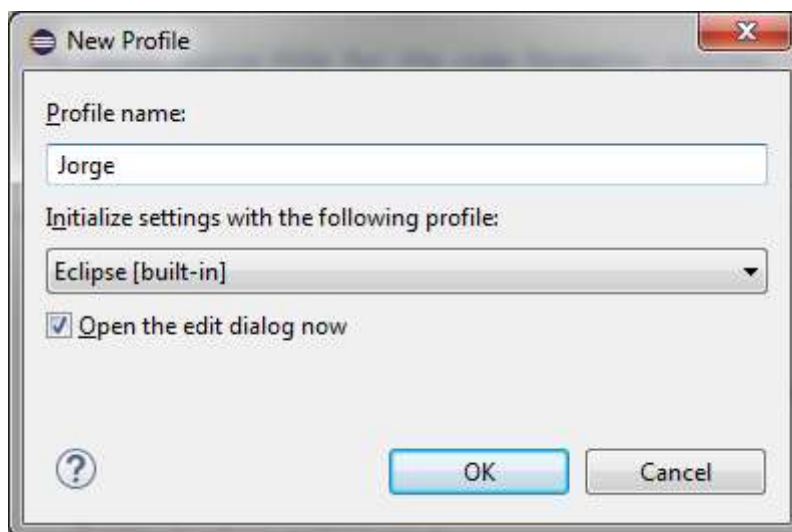
A lo largo de los años de trabajo con código JAVA, por experiencia propia, a continuación os presentaré la mejor manera de escribir el código en Eclipse. En programas con pocas líneas tampoco se nota la “mejoría”. Pero cuando nuestros programas tengan cientos de líneas, la forma de movernos por el código será más fluida si lo hacemos de la siguiente forma. Es el denominado “código limpio”.

Veamos en Eclipse dónde configurar este aspecto. Podemos hacerlo para un código en concreto que tengamos en pantalla, o bien hacerlo por defecto, para todos nuestros proyectos que montemos a partir de este momento. Será esta la opción elegida.

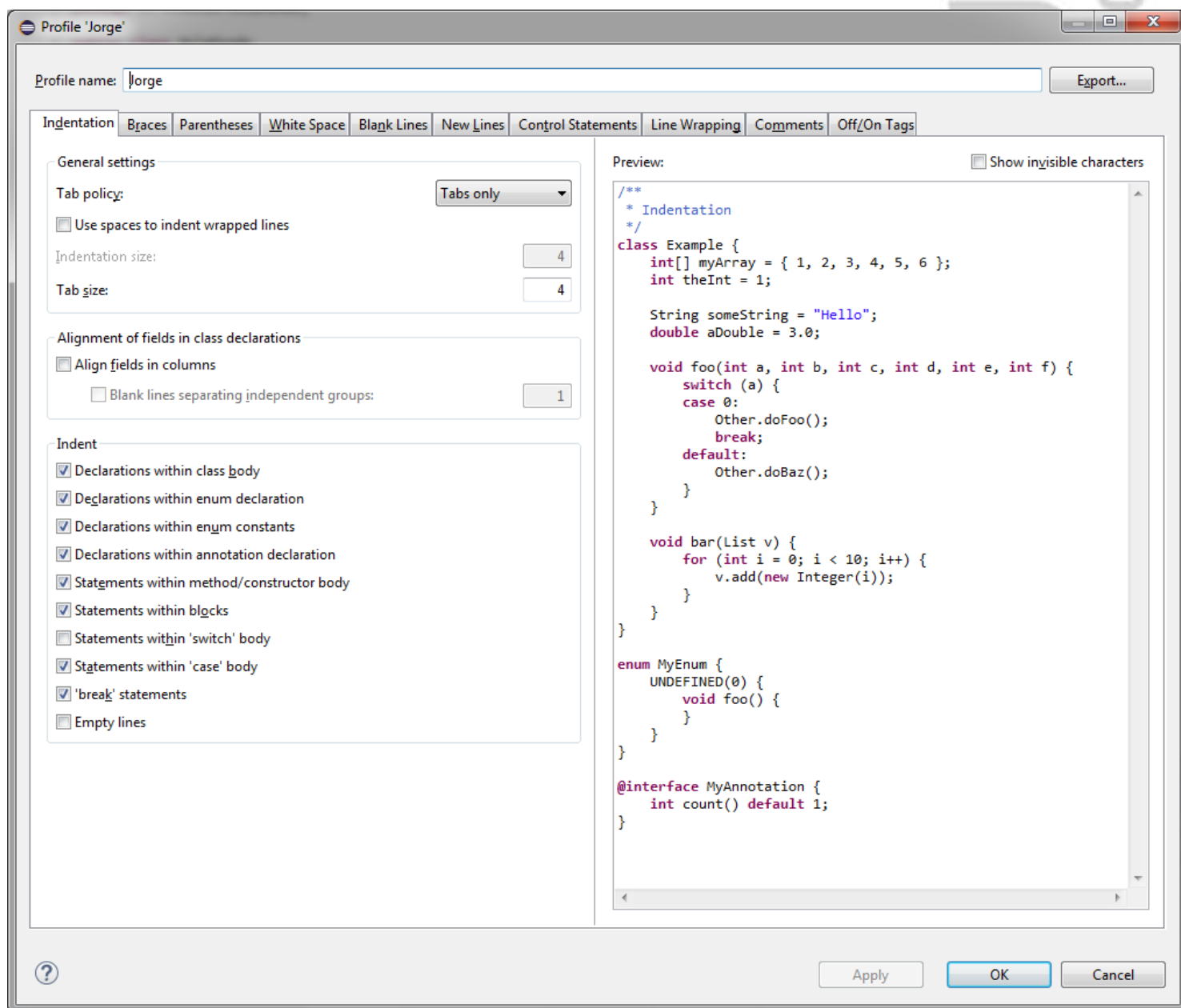
Debemos navegar a “Window”, “Java”, “Code Style” y “Formatter”. Llegados a esta pantalla tenemos una vista previa del código formateado. Podemos editar formatos ya preestablecidos, o bien crear el nuestro propio. Se pueden editar, guardar, exportar, importar, ...



Pulsaremos en “New” para crear nuestro estilo propio:



Daremos un nombre y elegiremos la plantilla a partir de la cual montaremos nuestro estilo. Al pulsar “OK” se abrirá el editor del estilo.



A partir de esta pantalla podemos ir personalizando nuestro estilo de código. Por ejemplo en la pestaña “Indentation” podemos poner si queremos el formato en espacios en blanco o en tabulaciones; si elegimos tabulaciones, el número de espacios en blanco por tabulación, etc.

Como podéis ver la ventana de edición de estilo se divide en dos partes, una parte izquierda en la que decimos el estilo y otra parte derecha en la que vemos

cómo quedaría el código al aplicarle el estilo. Navegando por cada una de las pestañas podréis ir editando vuestro estilo de código. A continuación os presento algunas de las pestañas más interesantes:

En la pestaña “Braces”, podéis poner como quedarían los “corchetes” en las clases, métodos, etc:

Brace positions

Class or interface declaration:	Next line
Anonymous class declaration:	Next line
Constructor declaration:	Next line
Method declaration:	Next line
Enum declaration:	Next line
Enum constant body:	Next line
Annotation type declaration:	Next line
Blocks:	Next line
Blocks in case statement:	Next line
'switch' statement:	Next line
Array initializer:	Next line
<input type="checkbox"/> Keep empty array initializer on one line	
Lambda body:	Next line

También podremos configurar el comportamiento para “Parentheses”, “White Space”, “Blank lines”, “New Lines”, “Control Statements”, “Line Wrapping”, “Comments” y “Off/On Tags”.

Parentheses positions

Method declaration:	Same line as content
Method invocation:	Same line as content
Enum constant declaration:	Same line as content
Annotation:	Same line as content
Lambda declaration:	Same line as content
'if', 'while' and 'do while' statement:	Same line as content
'for' statement:	Same line as content
'switch' statement:	Same line as content
'try' clause:	Same line as content
'catch' clause:	Same line as content

Más ejemplos

Vamos con el segundo ejercicio del boletín de ejercicios del Tema 1 que introduce nuevas herramientas de JAVA.

Empezamos creando un proyecto nuevo llamado “Suma” y le agregamos una nueva clase llamada también “Suma” dentro del paquete es.studium.Suma. El código sería de la siguiente forma:

```
package es.studium.Suma;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Suma
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader lectura = new BufferedReader(new
        InputStreamReader(System.in));
        int num1,num2, suma;
        System.out.println("Ingresa número: ");
        num1 = Integer.parseInt(lectura.readLine());
        System.out.println("Deme otro número: ");
        num2 = Integer.parseInt(lectura.readLine());
        suma = num1 + num2;
        System.out.println("La suma es "+suma);
    }
}
```

Veamos qué hay de nuevo con respecto al anterior.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

Con estas instrucciones indicamos al programa que vamos a usar instrucciones que no vienen en la librería estándar. Usamos la palabra reservada **import** y a continuación la librería que importamos. **TRUCO:** Si escribiendo código se nos olvida importar las librerías necesarias para nuestras instrucciones, pulsando **CTRL+MAY+O** se nos “escriben” los **imports** necesarios.

```
public static void main(String[] args) throws IOException
```

Se ha añadido a la función principal **main** la partícula **throws IOException** porque en su interior vamos a utilizar instrucciones que pueden elevar excepciones del tipo **IOException**.

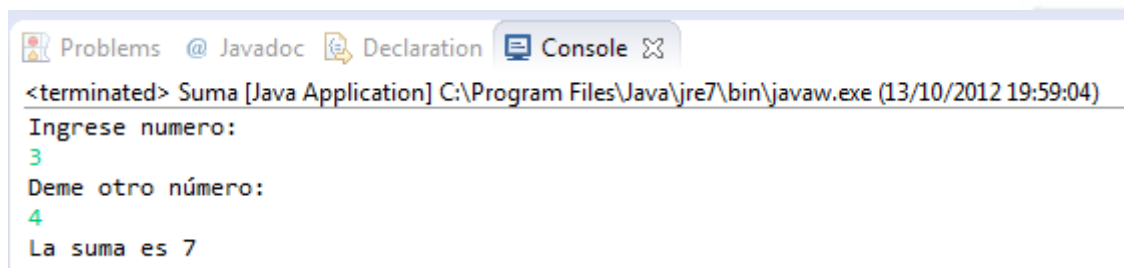
```
BufferedReader lectura = new BufferedReader(new InputStreamReader(System.in));
```

Creamos una memoria intermedia llamada **lectura** de tipo **BufferedReader** que nos permitirá interactuar con la **consola**, de modo que lo que escribamos en dicha consola se puede recoger y tratar dentro de mi programa.

```
num1 = Integer.parseInt(lectura.readLine());
```

Con esta instrucción metemos en la variable **num1**, declarada anteriormente de tipo entero, el contenido del **BufferedReader**, que no es más que lo que hayamos escrito en la **consola**. Todo lo que leamos de la consola es de tipo **cadena**. Como **num1** es **entero**, debemos hacer una transformación de cadena a entero con la instrucción **Integer.parseInt**.

Al ejecutar el programa, debemos tener en cuenta que hay que pinchar con el ratón en la consola para poder escribir. Un resultado posible podría ser:



```
<terminated> Suma [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (13/10/2012 19:59:04)
Ingrese numero:
3
Deme otro número:
4
La suma es 7
```


El código del tercer ejercicio sería:

```
package es.studium.ParImpar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class ParImpar
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader lectura = new BufferedReader(new
        InputStreamReader(System.in));
        int numero;
        System.out.println("Ingrese número: ");
        numero = Integer.parseInt(lectura.readLine());
        if(numero%2==0)
        {
            System.out.println("El número es PAR");
        }
        else
        {
            System.out.println("El número es IMPAR");
        }
    }
}
```

El siguiente ejercicio, llamado Pares, quedaría de la siguiente forma:

```
package es.studium.Pares;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Pares
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader lectura = new BufferedReader(new
        InputStreamReader(System.in));
        int num1,num2,i;
        System.out.println("Ingrese número: ");
        num1 = Integer.parseInt(lectura.readLine());
        System.out.println("Deme otro número: ");
        num2 = Integer.parseInt(lectura.readLine());
        for(i=num1;i<=num2;i++)
        {
            if(i%2==0)
            {
                System.out.println(i);
            }
        }
    }
}
```

Ahora vamos a por el Suma Pares – Producto Impares:

```
package es.studium.SumaParesProductoImpares;

public class SumaParesProductoImpares
{
    public static void main(String[] args)
    {
        int suma, producto, i;
        suma = 0;
        producto = 1;
        for(i=1;i<=40;i++)
        {
            if(i%2==0)
            {
                suma=suma+i;
            }
            else
            {
                producto=producto*i;
            }
        }
        System.out.println("La suma de los pares es "+suma);
        System.out.println("El producto de los impares es "+producto);
    }
}
```

Por último, para introducir un nuevo concepto como son las funciones realizaremos el ejercicio 6 Dividir. Quedaría de la siguiente forma:

```
package es.studium.Dividir;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Dividir
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader lectura = new BufferedReader(new InputStreamReader(System.in));
        int num1,num2;
        float division;
        System.out.println("Ingrese número: ");
        num1 = Integer.parseInt(lectura.readLine());
        System.out.println("Deme otro número: ");
        num2 = Integer.parseInt(lectura.readLine());
        division = dividir(num1,num2);
        System.out.println("La división es "+division);
    }
    public static float dividir(int a, int b)
    {
        float resultado;
        resultado = (float) a /(float) b;
        return(resultado);
    }
}
```

Comentar simplemente que en la función dividir, se ha tenido que hacer un **casting** de a y de b al realizar la operación porque de otra forma se haría la división entera. Es decir, si tenemos $9 / 2$ como enteros, nos daría 4.0. Sin embargo, al hacer el **casting** estamos realizando la división $9.0 / 2.0$ y esto sí resulta 4.5.

Números aleatorios

Para generar números aleatorios utilizaremos la librería java.Math. La clase Math define una función denominada *random* que devuelve un número pseudoaleatorio comprendido en el intervalo [0.0, 1.0):

```
package es.studium.Aleatorios;

public class Aleatorios
{
    public static void main(String[] args)
    {
        System.out.println("Número aleatorio: " + Math.random());
        System.out.println("Otro número aleatorio: " + Math.random());
    }
}
```

Pero la secuencia que se obtenga es siempre la misma y tiene muchas limitaciones. Existe otra posibilidad: la clase Random.

La clase Random proporciona un generador de números aleatorios que es más flexible que la función estática random de la clase Math.

Para crear una secuencia de números aleatorios tenemos que seguir los siguientes pasos:

- Proporcionar a nuestro programa información acerca de la clase Random. Al principio del programa escribiremos la siguiente sentencia:

```
import java.util.Random;
```

- Crear un objeto de la clase Random
- Llamar a una de las funciones miembro que generan un número aleatorio y
- Usar el número aleatorio.

Constructores

La clase dispone de dos constructores, el primero crea un generador de números aleatorios cuya semilla es inicializada en base al instante de tiempo actual:

```
Random rnd = new Random();
```

El segundo, inicializa la semilla con un número del tipo long.

```
Random rnd = new Random(3816L);
```

El sufijo L no es necesario, ya que aunque 3816 es un número int por defecto, es promocionado automáticamente a long.

Aunque no podemos predecir que números se generarán con una semilla particular, podemos sin embargo, duplicar una serie de números aleatorios usando la misma semilla. Es decir, cada vez que creamos un objeto de la clase Random con la misma semilla obtendremos la misma secuencia de números aleatorios.

Por ejemplo, el siguiente código mostraría por pantalla 10 números aleatorios entre 0 y 999:

```
package es.studium.Aleatorios2;
import java.util.Random;
public class Aleatorios2
{
    public static void main(String[] args)
    {
        Random rnd=new Random();
        int i,x;
        for(i=0;i<10;i++)
        {
            x=rnd.nextInt(1000);
            System.out.println(x);
        }
    }
}
```

Funciones miembro

Podemos cambiar la semilla de los números aleatorios en cualquier momento, llamando a la función miembro setSeed.

```
rnd.setSeed(3816);
```

Podemos generar números aleatorios en cuatro formas diferentes:

```
rnd.nextInt();
```

genera un número aleatorio entero de tipo int

```
rnd.nextLong();
```

genera un número aleatorio entero de tipo long

```
rnd.nextFloat();
```

genera un número aleatorio de tipo float entre 0.0 y 1.0, aunque siempre menor que 1.0

```
rnd.nextDouble();
```

genera un número aleatorio de tipo double entre 0.0 y 1.0, aunque siempre menor que 1.0

Casi siempre usaremos esta última versión. Por ejemplo, para generar una secuencia de 10 números aleatorios entre 0.0 y 1.0 escribimos:

```
package es.studium.Aleatorios3;
import java.util.Random;
public class Aleatorios3
{
    public static void main(String[] args)
    {
        Random rnd=new Random();
        for (int i = 0; i < 10; i++)
        {
            System.out.println(rnd.nextDouble());
        }
    }
}
```

Para crear una secuencia de 10 números aleatorios enteros comprendidos entre 0 y 9 ambos incluidos escribimos:

```
package es.studium.Aleatorios4;
import java.util.Random;
public class Aleatorios4
{
    public static void main(String[] args)
    {
        Random rnd=new Random();
        int x;
```



```

        for (int i = 0; i < 10; i++)
        {
            x = (int)(rnd.nextDouble() * 10.0);
            System.out.println(x);
        }
    }
}

```

(int) transforma un número decimal double en entero int eliminando la parte decimal.

Secuencias de números aleatorios

En la siguiente porción de código, se imprime dos secuencias de cinco números aleatorios uniformemente distribuidos entre [0, 1), separando los números de cada una de las secuencias por un carácter tabulador.

```

package es.studium.Aleatorios5;
import java.util.Random;
public class Aleatorios5
{
    public static void main(String[] args)
    {
        Random rnd=new Random();
        System.out.println("Primera secuencia");
        for (int i = 0; i < 5; i++)
        {
            System.out.print("\t"+rnd.nextDouble());
        }
        System.out.println("");
        System.out.println("Segunda secuencia");
        for (int i = 0; i < 5; i++)
        {
            System.out.print("\t"+rnd.nextDouble());
        }
        System.out.println("");
    }
}

```

Comprobaremos que los números que aparecen en las dos secuencias son distintos. En la siguiente porción de código, se imprime dos secuencias iguales de

números aleatorios uniformemente distribuidos entre [0, 1). Se establece la semilla de los números aleatorios con la función miembro setSeed.

```
package es.studium.Aleatorios6;
import java.util.Random;
public class Aleatorios6
{
    public static void main(String[] args)
    {
        Random rnd=new Random();
        rnd.setSeed(3816);
        System.out.println("Primera secuencia");
        for (int i = 0; i < 5; i++)
        {
            System.out.print("\t"+rnd.nextDouble());
        }
        System.out.println("");

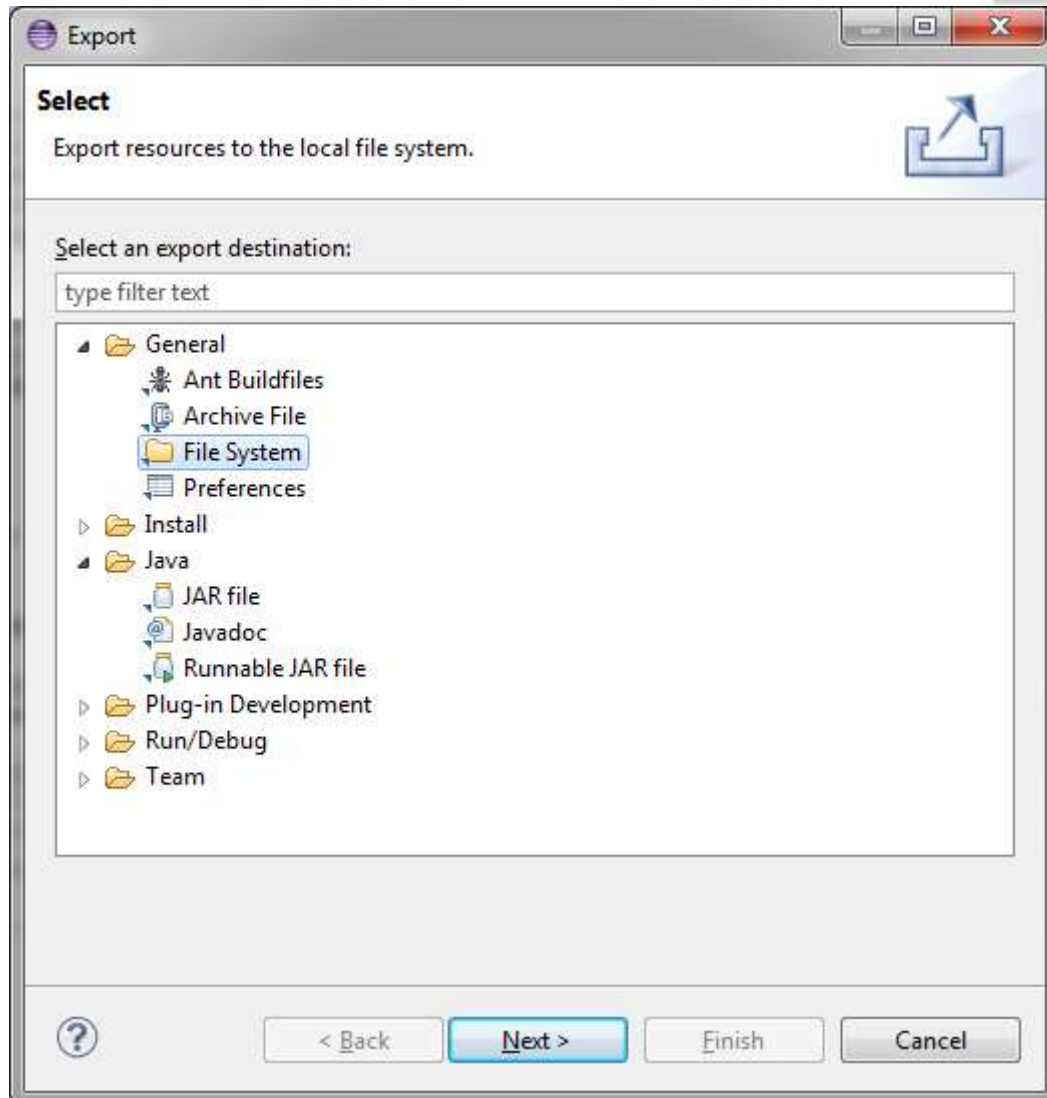
        rnd.setSeed(3816);
        System.out.println("Segunda secuencia");
        for (int i = 0; i < 5; i++)
        {
            System.out.print("\t"+rnd.nextDouble());
        }
        System.out.println("");
    }
}
```

Importar – Exportar proyectos

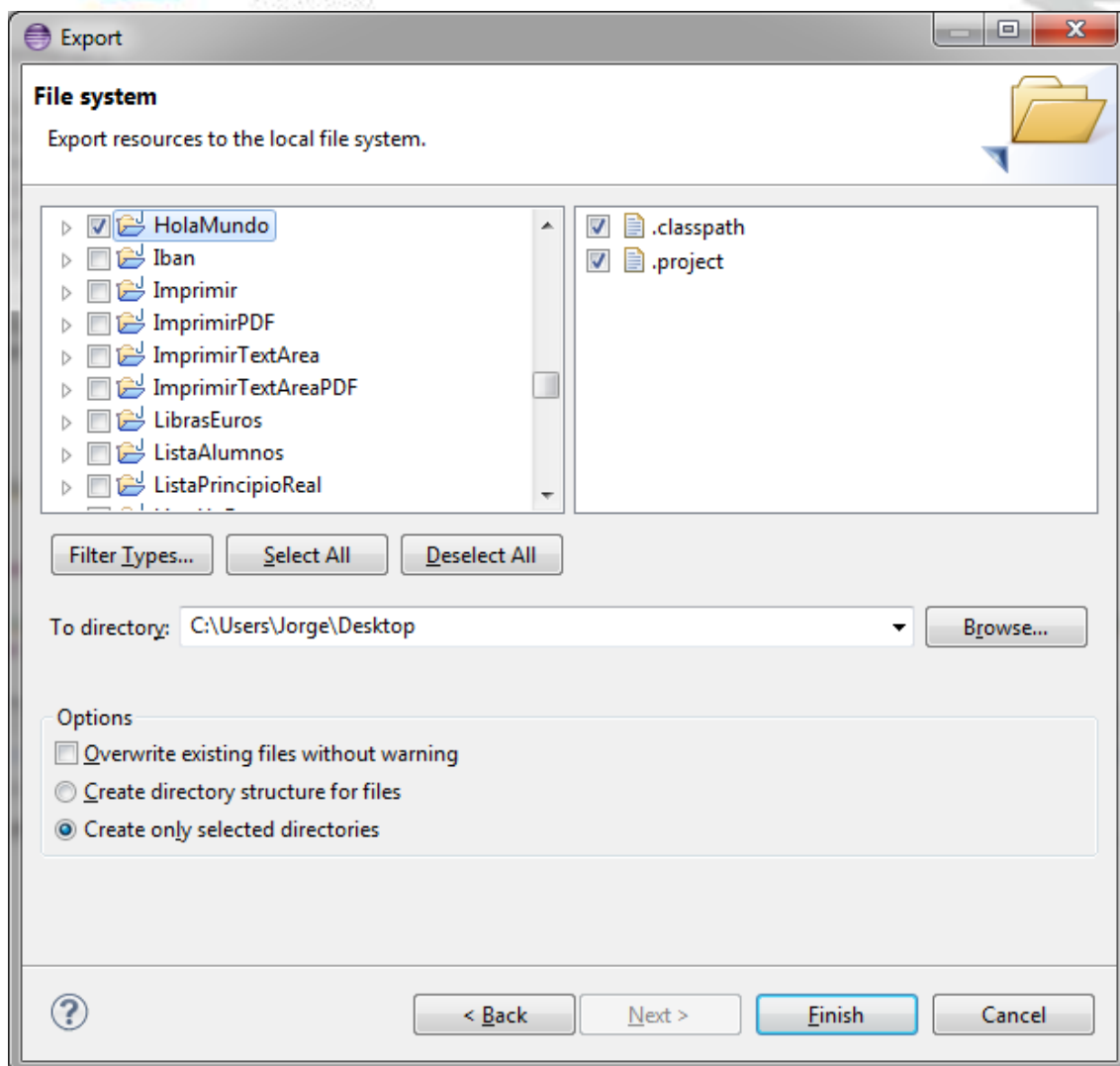
A medida que vayamos trabajando con Eclipse y realizando programas JAVA, a lo mejor tenemos la necesidad de “llevarnos” nuestros programitas de un ordenador a otro. Para este menester, vamos a ver cómo se exportan e importan proyectos en Eclipse.

Exportar

Con nuestro Eclipse abierto, pulsaremos con el botón derecho del ratón sobre el proyecto que queremos exportar.



Elegiremos la forma de exportar el fichero. Lo normal es “File System”. Pulsar en “Next” para continuar...



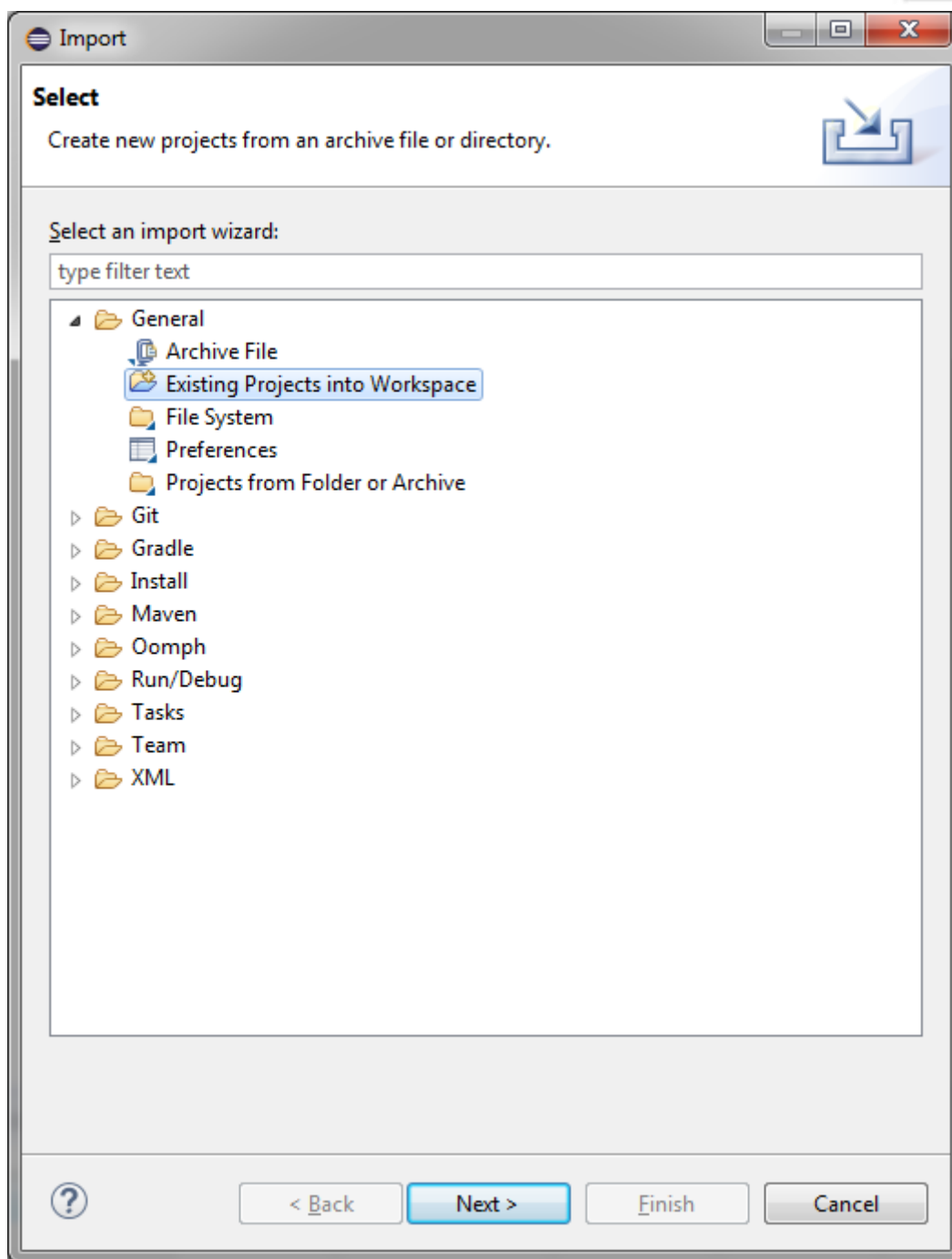
En el cuadro de la izquierda nos aparece el proyecto elegido, pero podemos elegir más de uno. Antes de pulsar en “Finish” para acabar la exportación, debemos elegir el destino del fichero exportado. En este caso sería el **Escritorio**. En esta ubicación, nos aparecerá una carpeta con el proyecto al completo:



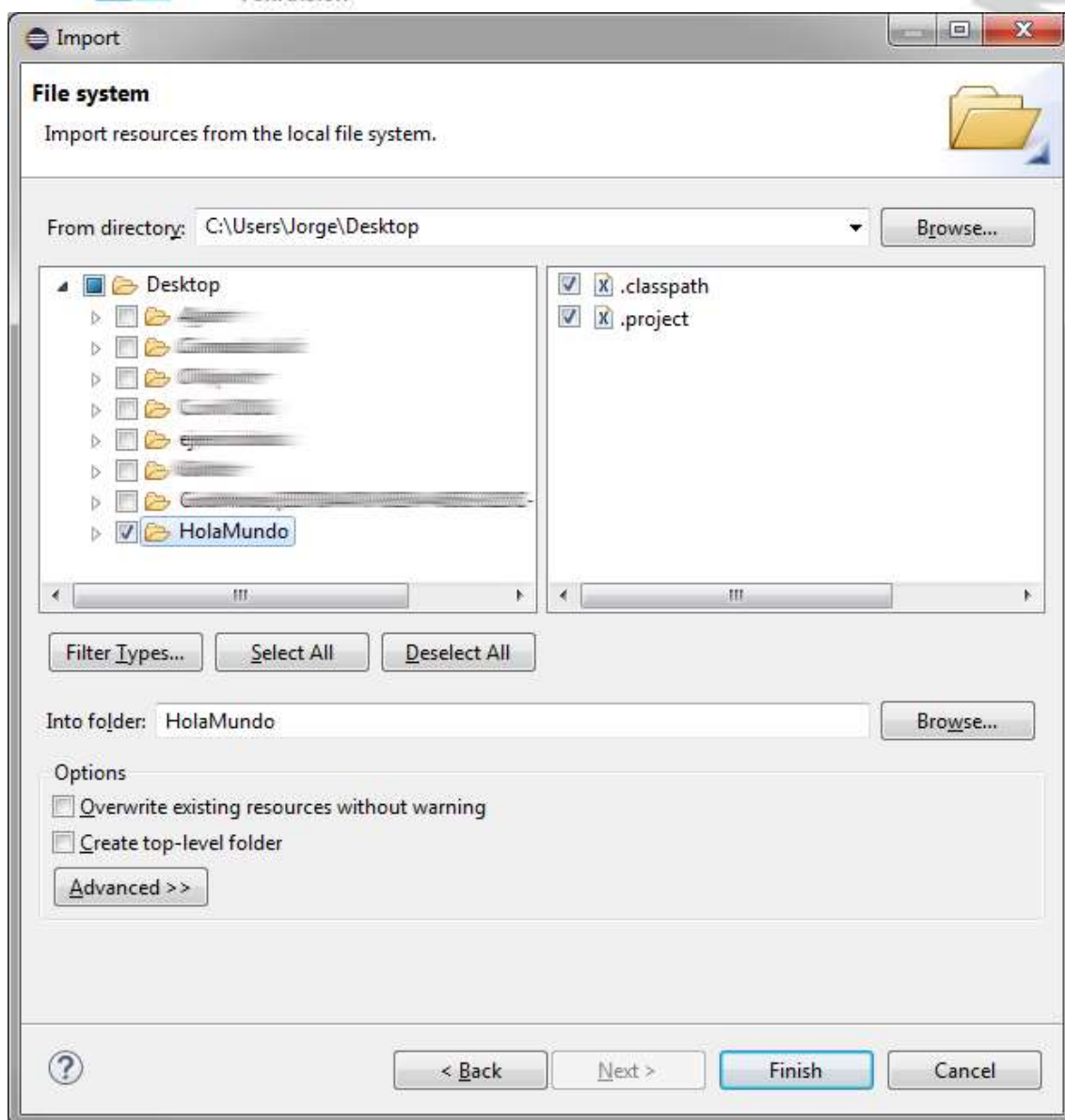
Importar

Ahora el proceso contrario. Dicho proyecto lo vamos a incluir en otro espacio de trabajo o workspace, bien en otra ubicación de nuestro mismo equipo, bien en otro equipo.

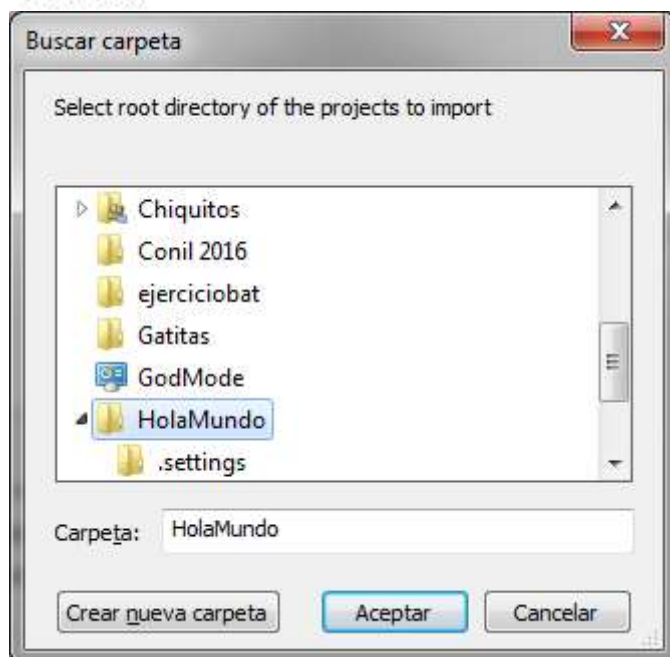
Sobre el “Package Explorer” con el botón derecho del ratón sobre el proyecto recién creado, elegiremos la opción “Import”:



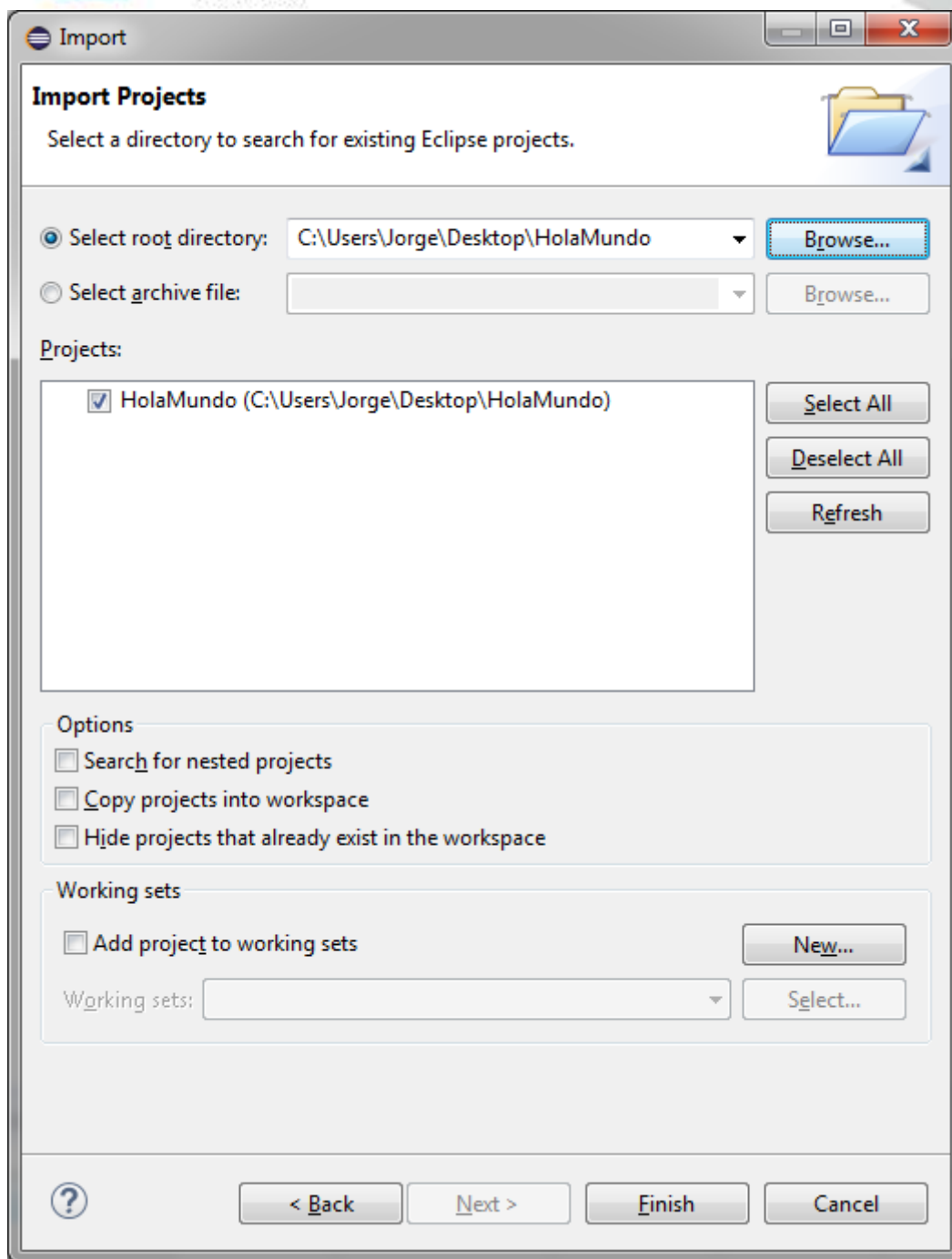
Ahora elegiremos “Existing Projects into Workspace” para la importación. Pulsar en “Next” para continuar ...



En “Browse” buscaremos lo que queremos...



Ahora buscar la carpeta que se creó en la **Exportación** y pulsar en “Aceptar”...



Comprobad que aparece el proyecto correcto y pulsar en “Finish”. Ya dispondremos del proyecto en nuestro nuevo Workspace.

23/07/2016