Breve introducción a Eclipse

Eclipse[1] es un entorno de programación que permite el desarrollo de aplicaciones en diferentes lenguajes. Consta de un entorno de desarrollo integrado (*IDE*) y de un conjunto de complementos (*plugin*) que permite extender su funcionalidad. Eclipse es un proyecto de código abierto (*open source*) y, como tal, cualquiera que lo desee puede desarrollar e integrar en él sus propios complementos.

El entorno clásico de Eclipse[2] está preparado para el desarrollo de código Java. Para ejecutar Eclipse en Linux abriremos el menú `Aplicaciones` y, normalmente, lo encontraremos dentro del submenú `Programación` si no encontramos un acceso directo desde el Escritorio.

La ventana de trabajo

Cuando se ejecuta Eclipse, lo primero que aparece es una ventana de diálogo (Figura 1) que permite indicar cuál va a ser nuestro espacio de trabajo (*workspace*). El espacio de trabajo es la carpeta donde se almacenará el trabajo que vayamos realizando (proyectos creados, código fuente, ejecutables, etc.).

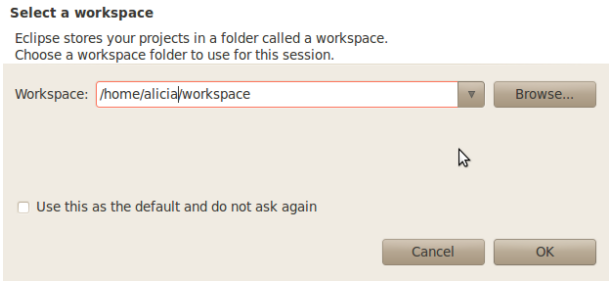

Figura 1: Ventana de diálogo para indicar el espacio de trabajo.

Es muy recomendable cambiar el *workspace*, ya que por defecto en el laboratorio se asigna uno que es accesible por cualquier usuario, por tanto se recomienda cambiarlo a uno que esté situado dentro de nuestro usuario.

Una vez definido el espacio de trabajo aparece el entorno de trabajo (*workbench*), donde se realiza todo el proceso de desarrollo del programa. La primera vez será como muestra la Figura 2, donde podemos elegir ver tutoriales, ejemplos, etc. Para cerrar esta ventana inicial, pinchar en la pestaña arriba a la izquierda (junto a Welcome).

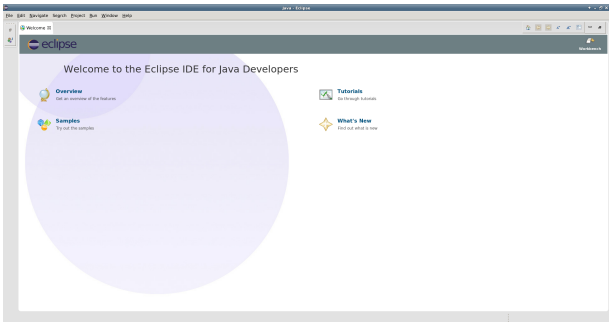

Figura 2: Ventana de trabajo inicial.

Una vez dentro del entorno de trabajo de Eclipse, éste contiene una o más perspectivas (*perspectives*). Las perspectivas contienen vistas (*views*) y editores (*editors*), y determinan qué opciones se muestran en ciertos menús y barras de herramientas, adaptando el entorno de Eclipse a la tarea que vayamos a desarrollar. Si tenemos instalados complementos en Eclipse para desarrollar programas con distintos lenguajes, tendremos perspectivas diferentes para trabajar con cada uno de ellos. La Figura 3 muestra la perspectiva por defecto para el desarrollo de código en Java. Los principales elementos que componen esta perspectiva son los siguientes:

[1] `http://www.eclipse.org/`.
[2] `https://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr1`.

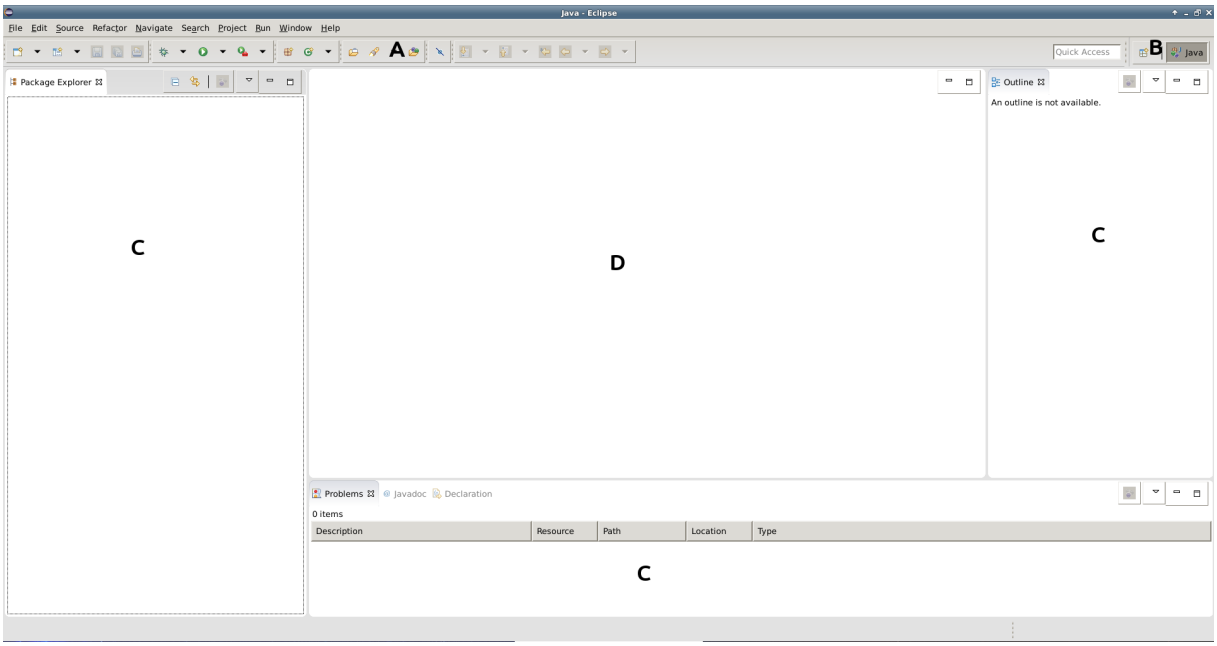

Figura 3: Perspectiva de Eclipse para trabajar en Java.

(A) *Barra de herramientas*: contiene accesos directos a las opciones más habituales.

(B) *Cambio de perspectiva*: contiene los botones para cambiar de perspectiva. También se puede hacer a través del menú `Window > Open Perspective`.

(C) *Vistas*: son componentes visuales de la ventana de trabajo que ayudan a llevar a cabo determinadas tareas. Una vista puede aparecer por sí sola o junto a otras vistas en una barra de pestañas.

(D) *Editor*: permite introducir y modificar el código fuente.

Podemos modificar el conjunto de vistas que se muestran en pantalla añadiendo o eliminando vistas en función de nuestras necesidades. Para añadir vistas a una perspectiva elegimos el menú `Window > Show View` y seleccionamos la vista que queramos incorporar. En `Other...` podemos encontrar vistas que no están asociadas por defecto a la perspectiva de trabajo actual.

Las vistas se usan típicamente para navegar por una jerarquía de información (como el conjunto de ficheros y carpetas de nuestro proyecto), o mostrar propiedades para el editor activo. Las vistas más utilizadas en la perspectiva de desarrollo de Java son las siguientes:

- `Project Explorer`: se abre por defecto a la izquierda de la pantalla en una nueva pestaña y proporciona una visión jerárquica de los recursos (carpetas y ficheros) de los proyectos Java con los que trabajamos.
- `Navigator`: también se abre a la izquierda de la pantalla y proporciona una visión jerárquica de los recursos (carpetas y ficheros) de nuestro espacio de trabajo (directorio).
- `Console`: se abre en la parte inferior de la pantalla, muestra la salida del programa y del compilador. También permite la entrada de texto por teclado al programa. Dependiendo del tipo de texto (salida estándar, salida de error o entrada estándar), el color del texto que se muestra es diferente.
- `Outline`: según la versión y configuración de Eclipse puede aparecer a la izquierda o derecha de la pantalla y muestra los elementos del código fuente que estemos editando (clases, métodos, etc.).
- `Problems`: situada en la parte inferior de la pantalla, muestra los errores generados por el sistema, las advertencias (*warnings*) y la información asociada con un recurso. Esta información la proporciona típicamente el compilador. Si guardamos un programa que contiene errores de sintaxis, éstos se mostrarán automáticamente en esta vista.

Mientras se trabaja en Eclipse es habitual abrir, mover, redimensionar y cerrar vistas. Podemos volver al aspecto original de la perspectiva seleccionando `Window > Reset Perspective`.

Crear un proyecto

Antes de empezar a escribir el código de nuestra aplicación debemos crear un proyecto para almacenar el código fuente, los binarios y otros ficheros relacionados que necesitemos. Para crear un nuevo proyecto en Eclipse, seleccionamos `File > New > Project`; a continuación se abre una ventana que nos permitirá seleccionar el tipo de proyecto que queremos crear. En nuestro caso será un `Java Project`. A partir de aquí el asistente de creación de proyectos nos permitirá crear uno. La Figura 4 muestra el primer paso de este asistente.

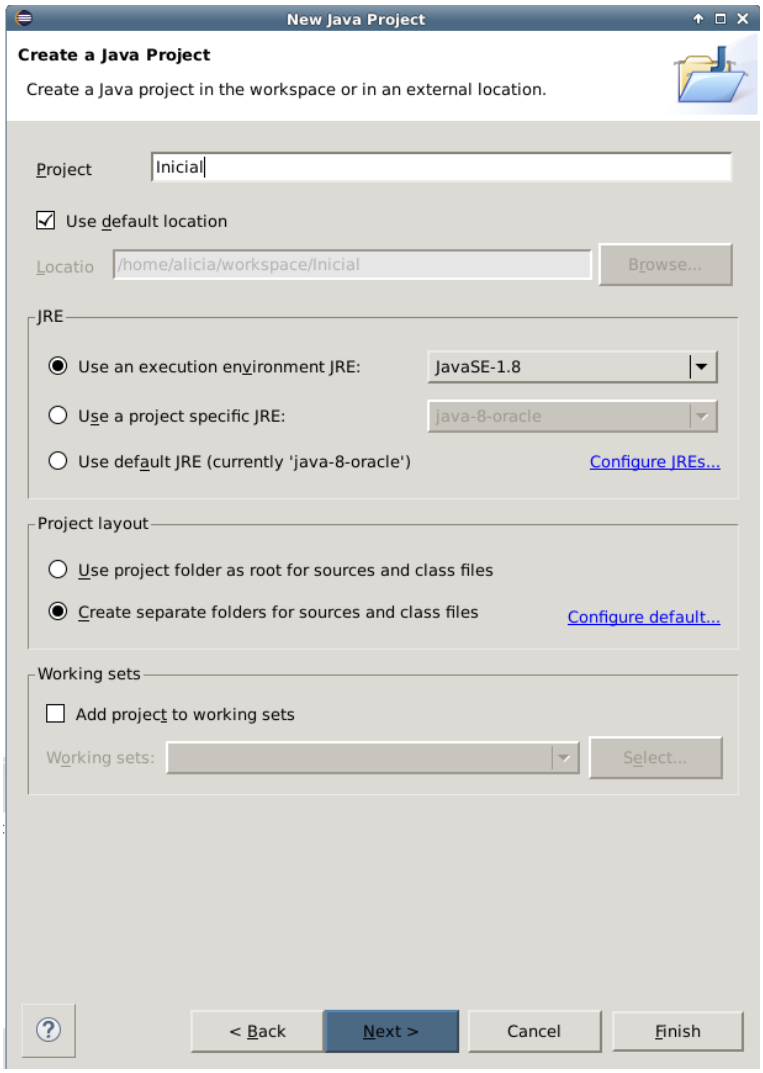

Figura 4: Asistente para la creación de proyectos Java. Primer paso.

Damos nombre a nuestro proyecto, en nuestro ejemplo será **`Inicial`**, dejamos las opciones marcadas por defecto y pinchamos el botón `Finish`. Con esto Eclipse crea dentro de la carpeta `workspace` de nuestro `home` una carpeta llamada **`Inicial`**, que a su vez contiene las carpetas `bin` (para almacenar los ficheros compilados con extensión `.class`) y `src` (para almacenar los ficheros con código fuente con extensión `.java`).

Crear un fichero para el código fuente

Para crear un nuevo fichero de código fuente seleccionamos `File > New > Class`. Aparecerá una ventana como la mostrada en la Figura 5. Aquí debemos indicar la carpeta donde se guardará el fichero (en nuestro caso `Inicial/src`), el nombre del fichero sin extensión (por ejemplo `Hola`) y le indicamos que nos incluya el método `main`. Por último pinchamos el botón `Finish`.

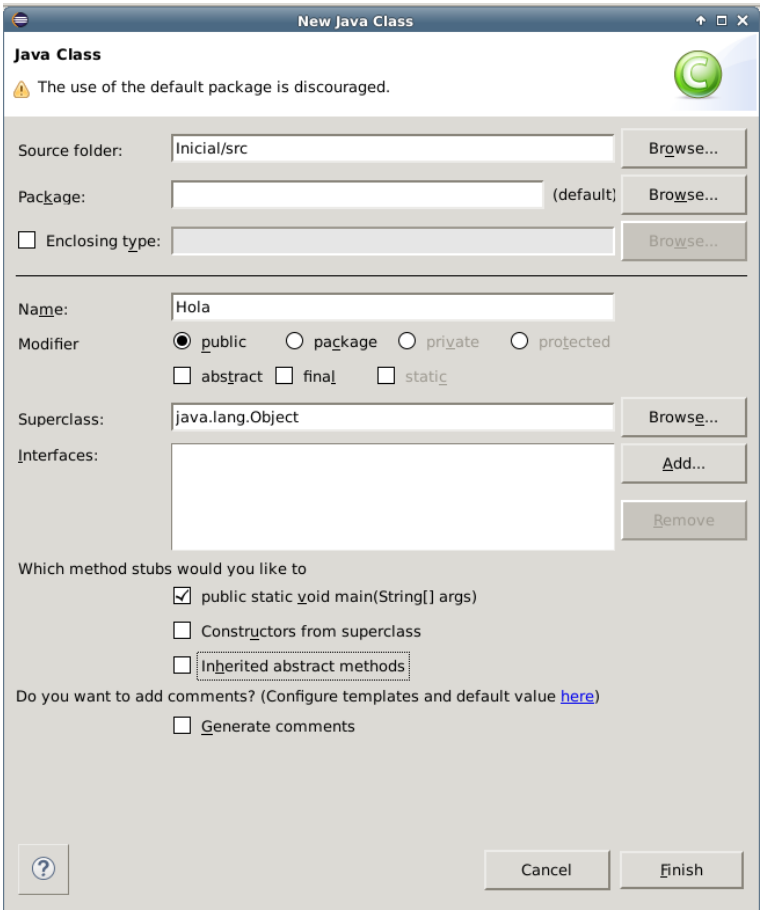

Figura 5: Creación de un fichero de código fuente.

El código se escribe en el editor Java localizado en el centro de la pantalla.

En un fichero con código fuente en Java puede haber implementada más de una clase pero sólo una de ellas puede ser pública y además esta debe llamarse igual que el fichero que la contiene; por este motivo Eclipse ya nos abre el fichero con el nombre de la clase como se puede ver en la Figura 6.

El método `main`

Cuando ejecutamos un programa, el sistema localiza y ejecuta el método `main` ya que es el punto de comienzo de la ejecución.

En Java este método debe ser **`public static void`** y aceptar un único argumento de tipo **`String []`**[3], que será donde estarán los argumentos del programa, si los tiene.

Por ejemplo, si queremos hacer un programa sin argumentos que muestre por pantalla el mensaje `Hello world! Learning java!`, el código que podemos escribir dentro de este método para conseguir nuestro objetivo es:

```
System.out.print("Hello world! ");
// Muestra por pantalla la cadena literal que se le pasa por parámetro
// equivalente a printf("Hello world! ");

System.out.println("Learning java!");
// Muestra por pantalla lo que se le pasa por parámetro y termina
// con una nueva linea.
// Equivalente a printf("Learning java!\n");
```

[3]Es un array de cadenas de tamaño variable; su tamaño se determina cada vez que se ejecuta el programa.

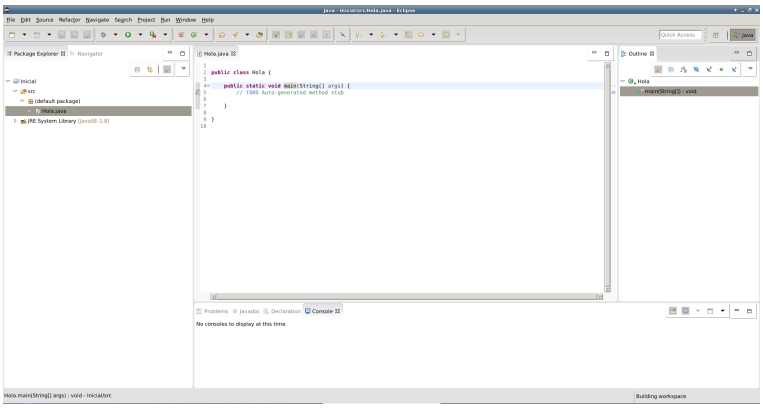

Figura 6: Creación de una clase en Java.

Paquetes en Java

Los lenguajes de programación suelen proporcionarse con un conjunto de bibliotecas estándar en las que se definen funciones de uso común. En Java, al ser un lenguaje orientado a objetos, se proporcionan bibliotecas de clases e interfaces, cada una de las cuales declara o define un conjunto de métodos.

Los paquetes (*packages*) definen unidades de software que se pueden distribuir independientemente y combinar con otros paquetes para formar aplicaciones. Los paquetes pueden contener:

- Clases
- Interfaces
- Otros paquetes

Para poder utilizar una clase definida en un paquete necesitamos, o bien importarla:

```
import java.util.Date;
...
Date mifecha;
```

O bien al declarar la variable se debe calificar el nombre de la clase incluyendo el nombre del paquete en que se encuentra:

```
java.util.Date mifecha;
```

Un programa Java está formado por las clases definidas en el programa más otras ya predefinidas en el ***API*** (*Application Programming Interface*).

Para importar todos los elementos de un paquete utilizamos el "`*`". Por ejemplo:

```
import java.util.*;
```

Esta instrucción nos permite disponer de todas las clases e interfaces creadas en el paquete `util` de java.

Esto no implica que el compilador lea toda la información relativa al paquete, sino que le indica dónde puede mirar si no encuentra a su alcance algún elemento declarado. Además sólo los elementos declarados como públicos (`public`) son visibles mediante la importación de paquetes.

Las clases e interfaces que pertenecen al paquete **`java.lang`** están disponibles por defecto, por tanto no es necesario importarlas.

Es importante tener en cuenta que cuando importamos un paquete obtenemos acceso a las clases e interfaces públicas que contiene directamente dicho paquete, pero no a las que contienen sus subpaquetes (véase Figura 7) ya que el anidamiento de paquetes es únicamente un agrupamiento lógico, sin más funcionalidad.

Los paquetes de uso más común en Java son:

Paquete de Java	Descripción
java.io	Clases para gestión flujos de datos entrada/salida
java.lang	Clases e interfaces básicas. Este paquete se importa automáticamente
java.util	Clases de utilidad para la gestión de estructuras de datos, fechas, etc.
java.math	Clases para aritmética de alta precisión

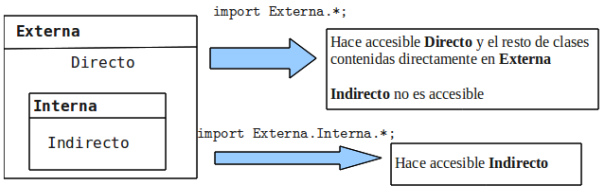

Figura 7: Acceso a clases en paquetes importados.

Compilación de un programa fuente

El compilador es un programa (o conjunto de programas) que se encarga de traducir un programa fuente en un programa ejecutable; el programa fuente está escrito en un lenguaje de alto nivel (C/C++, Pascal, Java, . . .), y el programa ejecutable está escrito en el lenguaje que entiende la máquina, el código máquina. En Java esto no es así exactamente, ya que Java es lo que se conoce como lenguaje interpretado. Realmente el compilador de Java traduce el código fuente a un tipo de código intermedio (entre el lenguaje de alto nivel que programamos y el lenguaje máquina que entiende el ordenador) que guarda en ficheros `.class` (uno por cada `.java`) y que deben ser "*interpretados*"[4] para ejecutarse.

Si durante el proceso de traducción el compilador encuentra algún error, produce un mensaje (indicando la fila en la que está el error) y no genera el fichero `.class`. A veces, el compilador produce un *aviso* (*warning* en inglés), indicando que hay algo raro, pero que no es incorrecto; en este caso sí se genera el fichero `.class`, pero lo más probable es que no funcione correctamente. Un buen programador (tenga más o menos experiencia) debe considerar errores todos los avisos que produce el compilador, y corregirlos como si fueran errores.

En Java (y en otros muchos lenguajes) un programa fuente puede (y suele) estar dividido en varios ficheros con código (acabados en "`.java`").

La compilación de los fuentes y la ejecución del programa resultante se puede realizar desde un terminal o desde Eclipse. Veremos en primer lugar cómo hacerlo desde terminal y a continuación cómo hacerlo desde Eclipse.

Compilación y ejecución desde terminal

El comando que debes utilizar para compilar un programa desde un terminal en Linux es:

```
javac programaFuente.java
```

Por ejemplo, para compilar el programa fuente que has escrito antes debes introducir el siguiente comando:

```
javac Hola.java
```

Este comando compila el programa fuente "`Hola.java`" y, si no hay errores, genera el fichero con código intermedio "`Hola.class`"; si el programa tiene errores hay que volverlo a editar, corregir los errores y volver a compilar.

Una vez hayas corregido todos los errores y *warnings* puedes ejecutar el programa generado por el compilador, para lo cual tienes que escribir lo siguiente en línea de comando:

```
java Hola
```

o bien:

```
java Hola >salida.txt
```

de manera que redirigimos la salida estándar (`>`, lo que muestra por pantalla) de nuestro programa a un fichero, en este caso "`salida.txt`". Este fichero se puede consultar con comandos (less, more, cat) o bien editar con cualquier editor de textos.

[4]Deben ser leídos por un intérprete de Java; normalmente se llama `java`.

Compilación y ejecución desde Eclipse

Eclipse realiza la compilación mientras se está editando el código, de manera que va marcando con iconos en el margen si hay errores o *warnings*.

Por ejemplo, vamos a introducir un error en nuestra aplicación. En Java no podemos declarar variables y utilizarlas sin inicializarlas con algún valor, esto produce un error de compilación.

Por tanto en nuestro programa vamos a declarar dos variables de tipo `String` (tipo cadena)[5]; una la inicializamos con un valor, la otra no y las mostramos por pantalla. El código a incluir sería:

```
String cadena1;
String cadena2;
// reservo memoria y doy valor
cadena1=new String("pruebas con cadenas ");
System.out.println(cadena1+cadena2);
// Muestra por pantalla el contenido de cadena1 concatenado
// con el contenido de cadena2. El operador + aplicado a String
// realiza la concatenación
```

Cuando grabamos, Eclipse nos marca la línea que contiene el error con un símbolo y una X, y además la vista `Problems` nos indica que existe un error en el fuente y nos indica cuál puede ser la causa al desplegar el error, como se puede ver en la Figura 8.

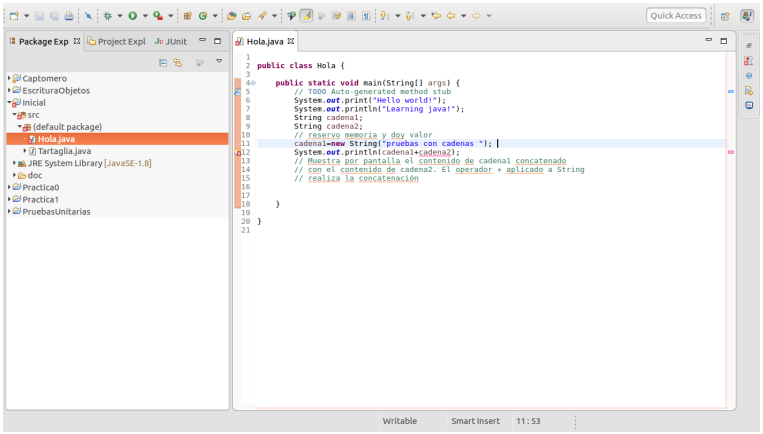

Figura 8: Detección de errores en el código fuente.

Corregimos el error asignándole algún valor a la segunda variable, guardamos (por ejemplo con `Ctrl+s`) y desaparecen las marcas de errores.

Para ejecutar hacemos:

- Entramos en el menú `Run > Run Configurations`.
- Aparece una nueva ventana para crear una configuración de ejecución para nuestra aplicación.
- Seleccionamos `Java Application` y pinchamos en el botón `New`, como se muestra en la Figura 9.

[5]Es un tipo de java que no es primitivo, por tanto para las variables de este tipo es necesario reservar memoria dinámica, o bien asignar un literal.

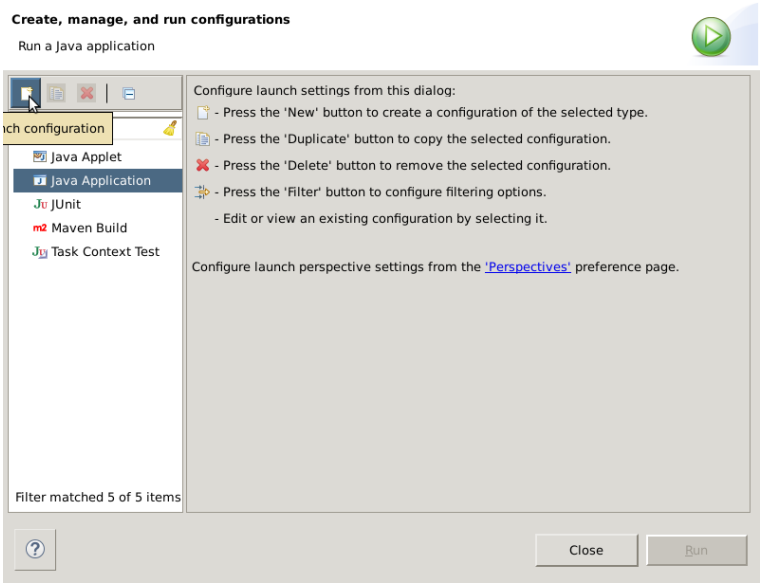

Figura 9: Cómo crear un perfil de ejecución para una aplicación.

- Aparece entonces una pestaña en la que hay varias pestañas para poder indicar las características particulares de cada ejecución cuando sea necesario. Cómo nuestra aplicación no tiene argumentos de entrada[6] y no usamos paquetes externos, pinchamos el botón `Run`, tal como se ve en la Figura 10.

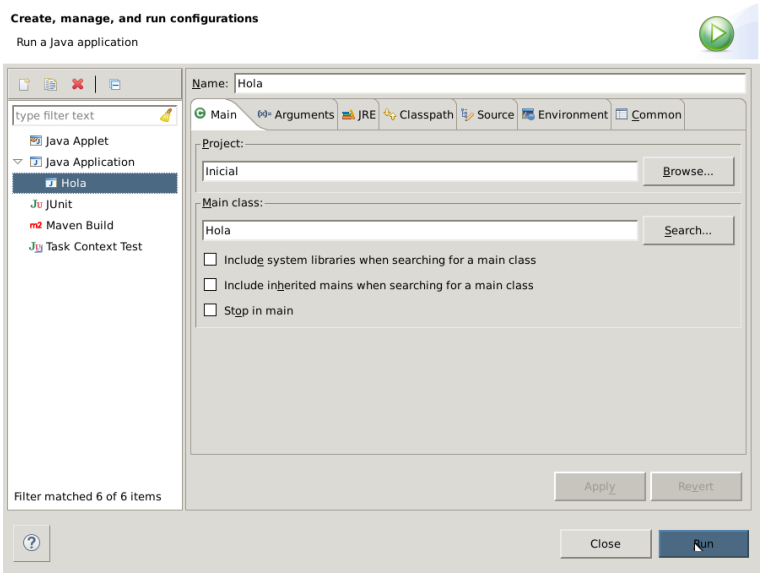

Figura 10: Ventana de opciones de ejecución.

En la vista `Console` podemos ver la salida del programa, como se puede ver en la Figura 11

[6]Se pondrían en la pestaña *Arguments*.

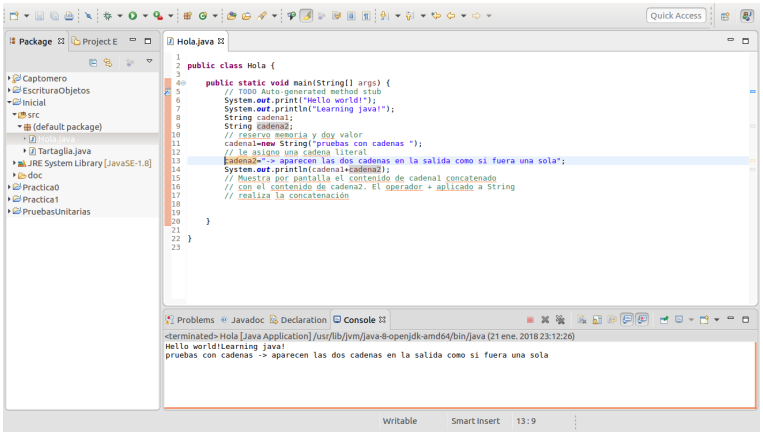

Figura 11: Ejecución de un programa desde Eclipse.

¿Qué me llevo?

Cuando trabajamos con Eclipse se genera toda una estructura de directorios/ficheros con la información del proyecto que hemos creado. Si trabajamos con un mismo ordenador (por ejemplo, el portátil), cuando volvemos a abrir Eclipse, éste abre por si sólo los proyectos que se han creado en el espacio de trabajo seleccionado. Si trabajamos en distintos ordenadores tendremos que grabar y llevarnos la información de uno a otro, por tanto, cuando acabemos de trabajar lo que tenemos que grabar para llevarnos es el directorio del proyecto en el que estemos trabajando (si es más de uno, todos los directorios de los proyectos modificados).

Ahora bien, cuando volvamos a trabajar con Eclipse en los laboratorios, tendremos nuestro directorio del proyecto pero hay que hacer que Eclipse lo reconozca.

Los pasos a seguir serán:

1. Copiar el directorio con nuestro proyecto al directorio `workspace` del ordenador del laboratorio.
2. Abrir Eclipse.
3. Crear un proyecto nuevo con el mismo nombre que nuestro proyecto.

Eclipse lo reconoce y lo abre como tal.

Sistema de entrega de prácticas: entrega de prueba

Vamos a realizar una entrega para probar el sistema de entrega que se va a utilizar para las prácticas a través de la web del DLSI. Para ello debes seguir los siguientes pasos:

1. Descargar del UACloud el fichero Ejercicio.java.
2. Abrir un terminal.
3. Situarse en el directorio donde se encuentran el fichero fuente (`.java`) de manera que al ejecutar `ls` se muestre el fichero y ningún directorio.
4. Ejecutar:

   ```
   tar cfvz  prueba.tgz   *.java
   ```

5. Entra en la web del DLSI, elige el idioma que prefieras.
6. Entra en "`Entrega de prácticas`".
7. Pincha en el enlace de la asignatura "`Programación 2 - Ingeniería Multimedia`".
8. Pincha en el enlace que pone "`EntregaPrueba`"
9. Lee las instrucciones, y rellena los datos del formulario. La contraseña que se te pide es la del campus virtual de la UA. Al lado del campo que pone "`Fichero prueba.tgz:`" tienes un botón que te permite buscar en el sistema de archivos el fichero que quieres entregar.
10. Una vez hayas rellenado los campos del correo electrónico, la contraseña y el campo del fichero a entregar, pincha en el botón "`Entregar`".