

# ►► Eclipse

Instalación y  
configuración

Entorno  
de  
Proyecto

Creación  
de  
elementos  
Java

Funciones  
útiles

Vistas de  
Eclipse

Ejecutar y  
depurar



01

Instalación y  
configuración



## ¿Qué es Eclipse?

Eclipse es un entorno de desarrollo integrado (IDE) que sirve para muchos lenguajes, incluido Java.

Un IDE como Eclipse es una aplicación que, integra en un único programa todas las herramientas para el desarrollo completo de una aplicación en sus diferentes etapas, desde la edición de código fuente hasta la obtención de un producto final.

Eclipse incorpora un editor con características especiales que facilitan la programación en Java. Además de compilar, depurar y ejecutar el proyecto sin necesidad de salirnos del entorno.



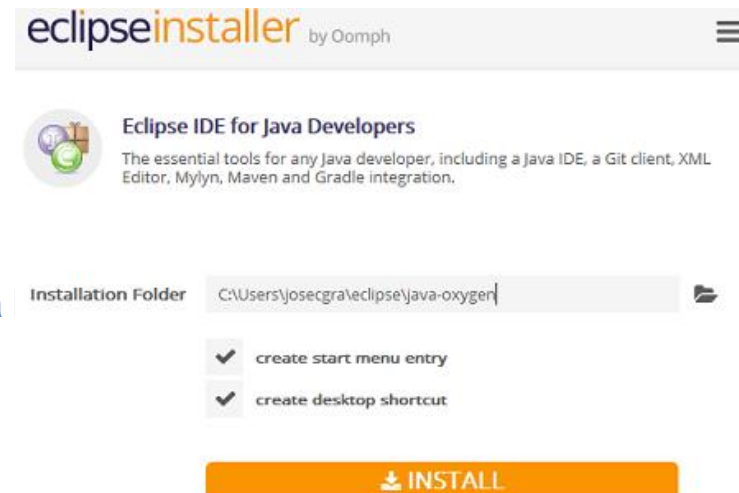


## Instalación

Su instalación es muy sencilla, podemos descargárnoslo de la web oficial [www.eclipse.org](http://www.eclipse.org) a la carpeta que queramos tenerlo instalado.

Arrancaremos el fichero *eclipse-inst-win64.exe* nos pedirá será por defecto donde queramos que eclipse nos vaya guardando los proyectos que creemos.

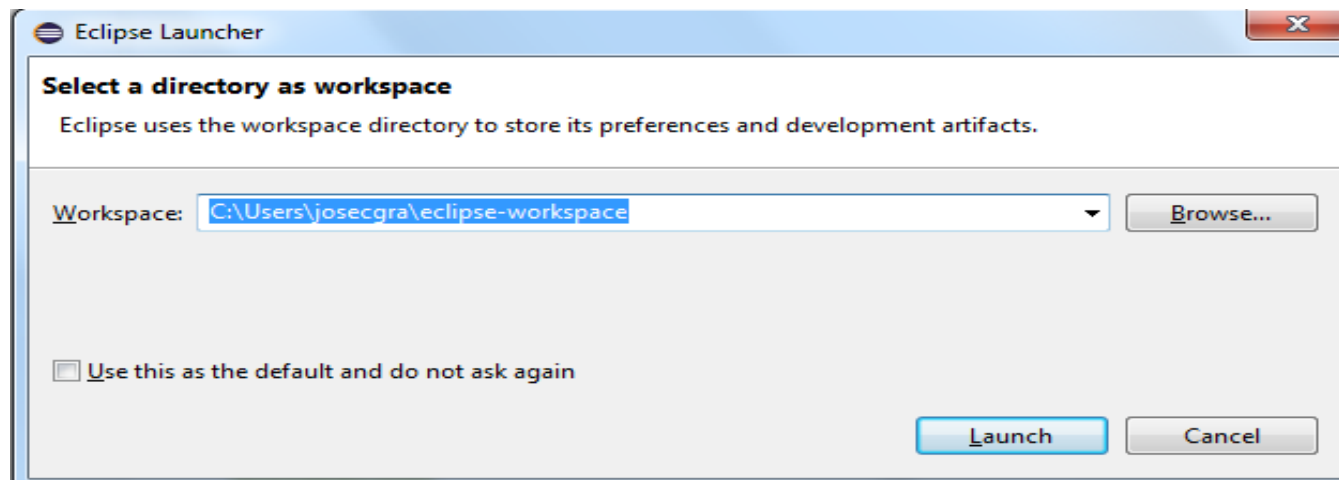
Una vez decidido el directorio de instalación, clicaremos en *Install*, para que comience la instalación del entorno.





## Configuración

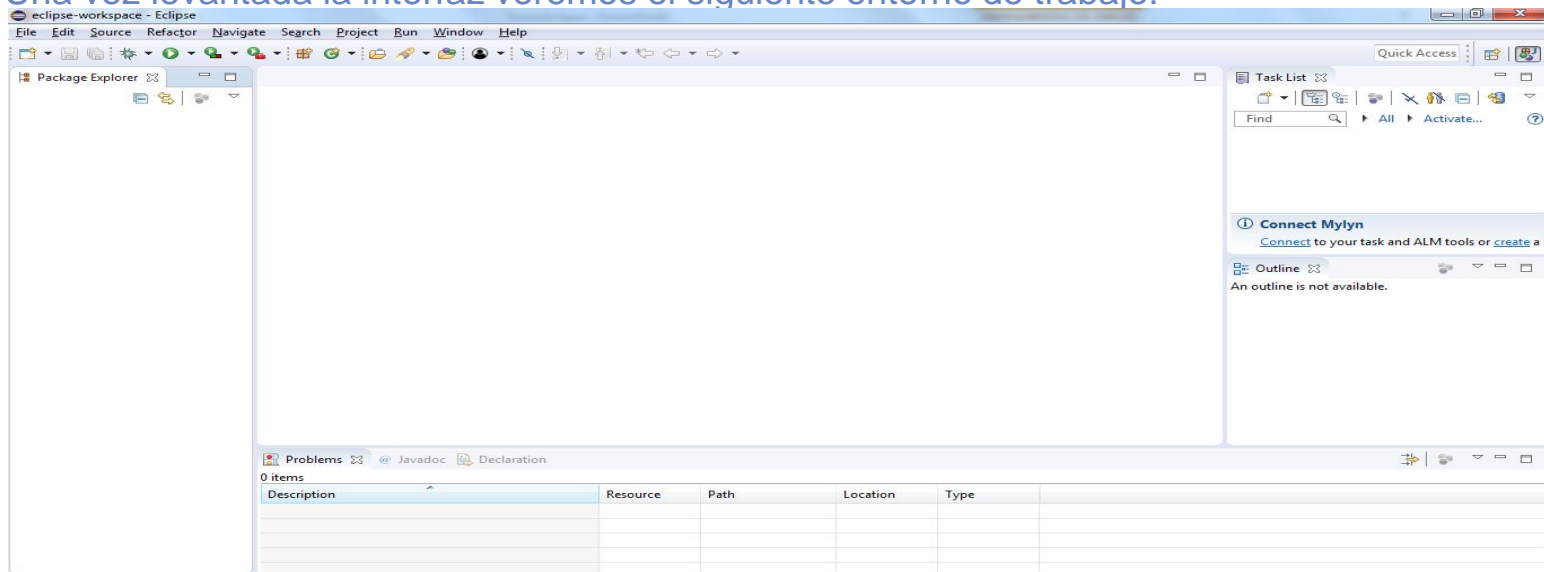
Al arrancar eclipse lo primero que nos pedirá será el directorio *workspace*, esto es el directorio físico donde alojará nuestros proyectos.





## Configuración

Una vez levantada la interfaz veremos el siguiente entorno de trabajo:





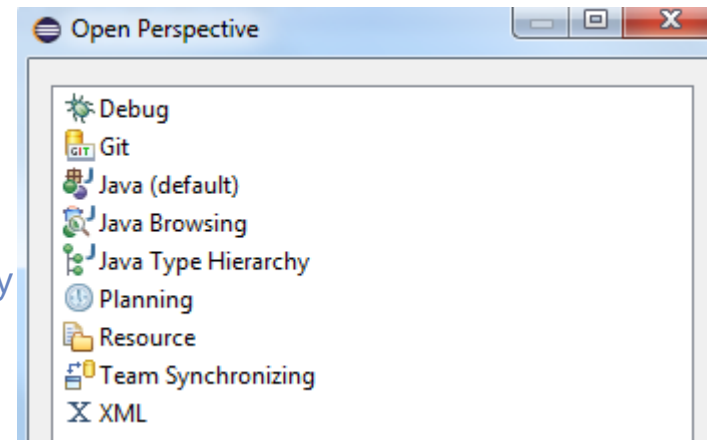


## Espacio de trabajo

El espacio de trabajo puede usar varias perspectivas en su ventana principal dependiendo del tipo de desarrollo que vayamos a realizar.

En el entorno de desarrollo Eclipse todo archivo se almacena dentro de un proyecto. Esto quiere decir que todo documento, carpeta, archivo de código fuente (.java) y código compilado (.class) tiene que estar contenido dentro de un proyecto.

Así pues, el primer paso antes de usar Eclipse es comprender la estructura de proyectos en Eclipse.

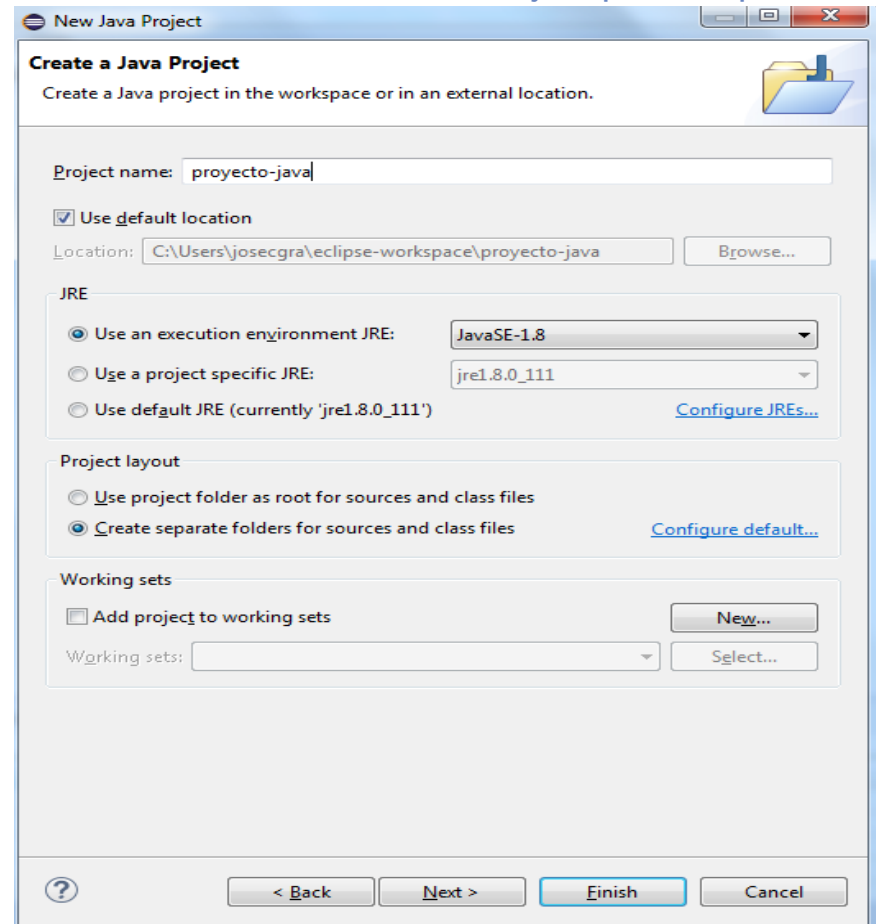
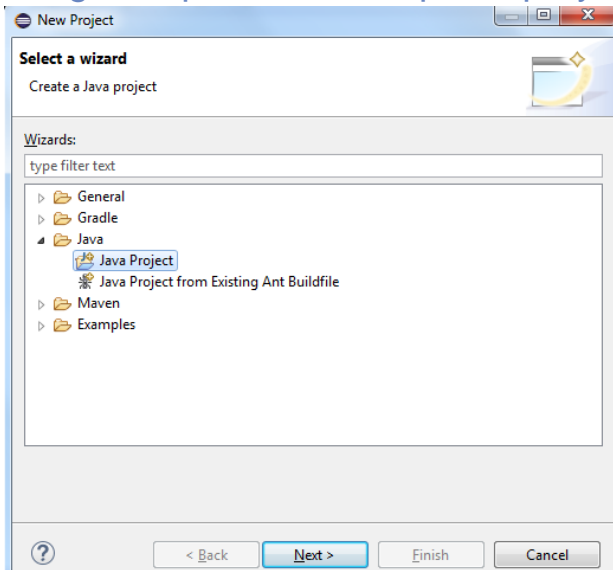


02

Entorno de  
Proyecto



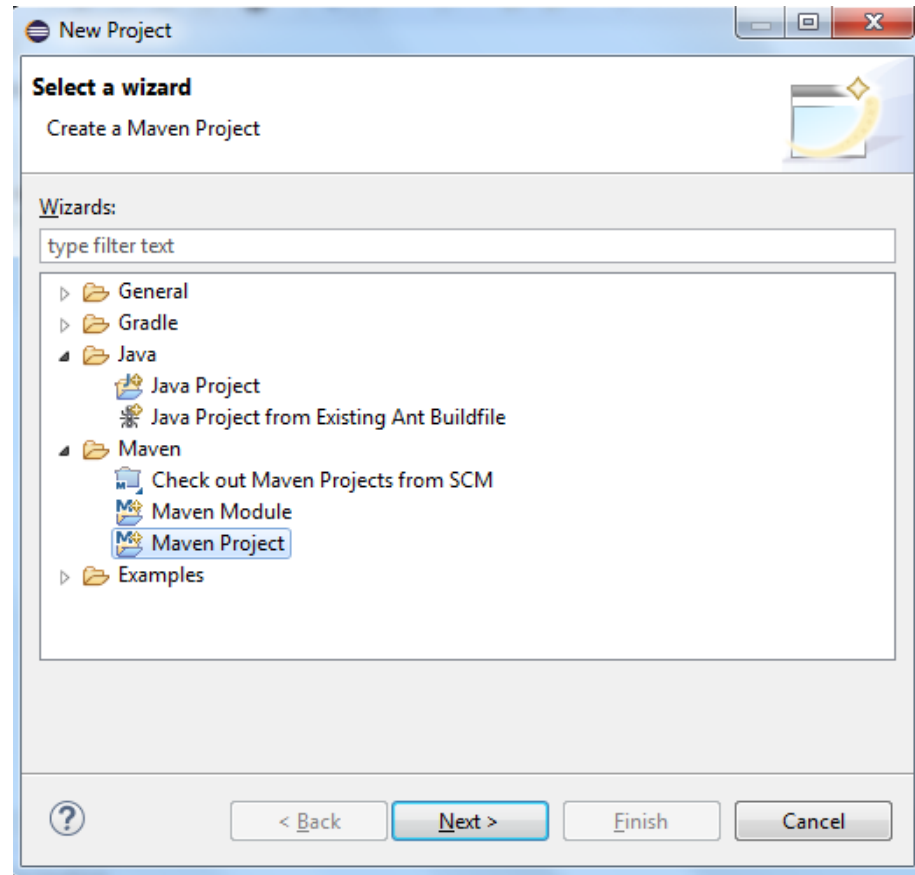
Para crear nuestro primero proyecto iremos a *File -> New -> Project* que nos abrirá el asistente que nos guiará para crear el tipo de proyecto que deseemos. Crearemos un *Java Project* para empezar.



# Eclipse

## Entorno de proyecto

Hay varios tipos de proyectos que pueden ser creados en Eclipse, entre ellos tenemos el tipo Maven que nos crea una estructura predeterminada siguiendo con el estándar Maven. Este tipo de arquitectura la veremos en detalle más adelante en la beca.

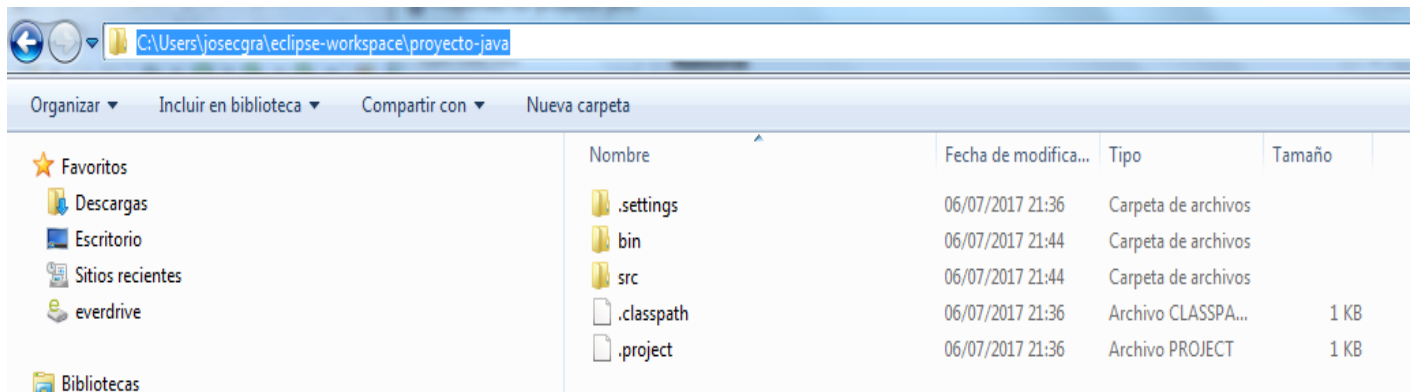


# Eclipse

## Entorno de proyecto

Veamos la estructura de carpetas que nos ha creado en el *workspace*:

- ❑ La carpeta */bin* es la que contiene el código compilado de la aplicación.
- ❑ La carpeta */src* es la que contiene todo el código fuente de la aplicación.



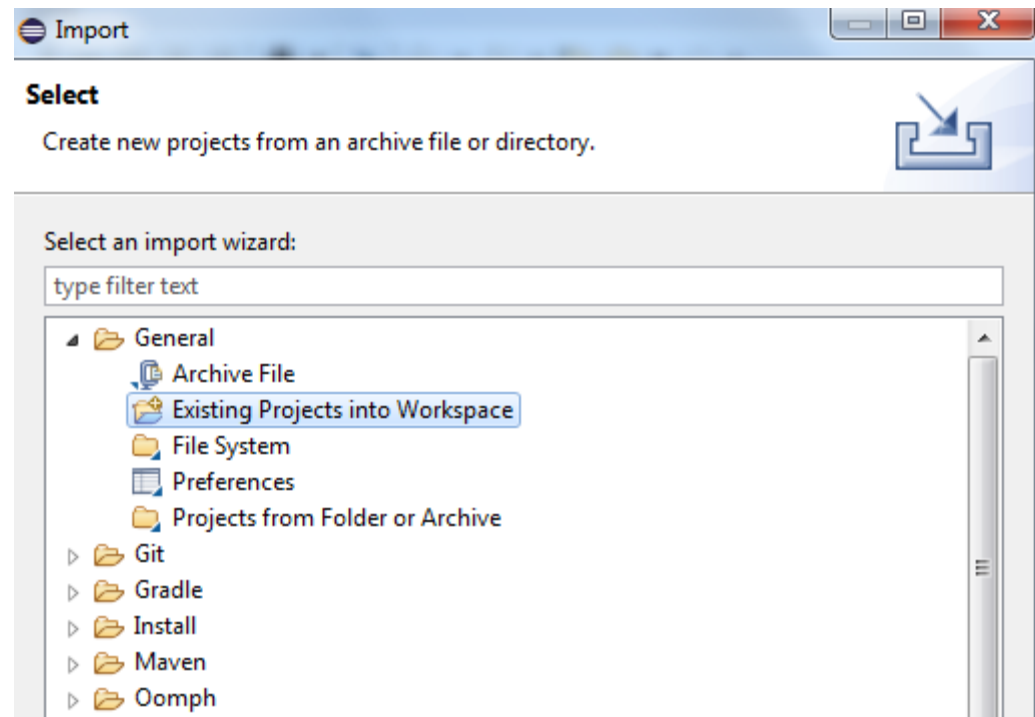


# Eclipse

## Entorno de proyecto

Eclipse es capaz de contener en la carpeta *workspace* varios proyectos (subcarpetas del workspace) a la vez abiertos en el área *Package Explorer*.

Si necesitamos importar un proyecto existente en el *workspace* al entorno de trabajo en Eclipse, iremos a las opciones *File -> Import..*



The background of the slide is a photograph of several pairs of hands clasped together on a wooden table. The image is split vertically: the left side has a green tint, and the right side has a blue tint. White, hand-drawn style lines are overlaid on the hands, suggesting movement or connection.

03

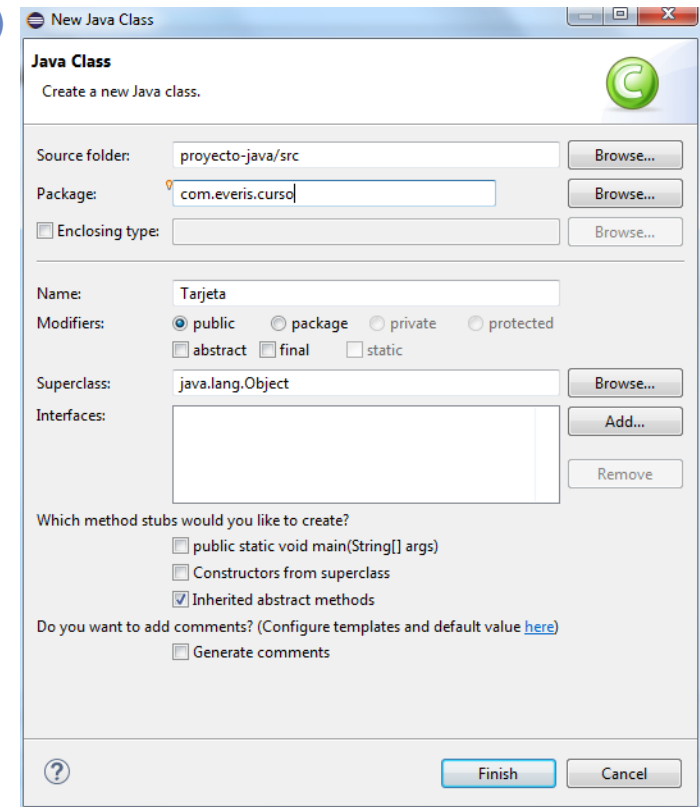
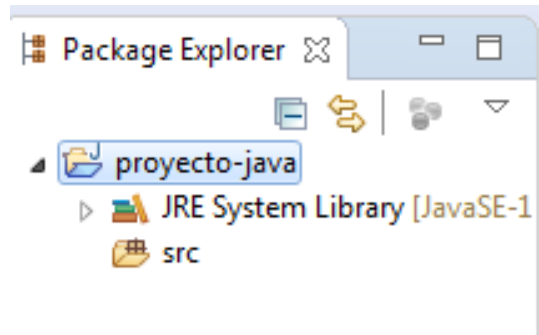
Creación  
de  
elementos  
Java

# Eclipse

## Creación de elementos Java

Definiremos nuestra primera clase *Tarjeta* con los datos de una tarjeta de contactos: *id* (int), *nombre* (String) y *eMail* (String) y una clase *Main* con el programa principal de la aplicación, que crea un par de tarjetas, las guarda en un array y lo recorre para mostrarlas por la salida estándar.

Para crear una clase seleccionamos *New -> Class* y rellenamos el cuadro de diálogo con el nombre y el paquete.





# Eclipse

## Creación de elementos Java

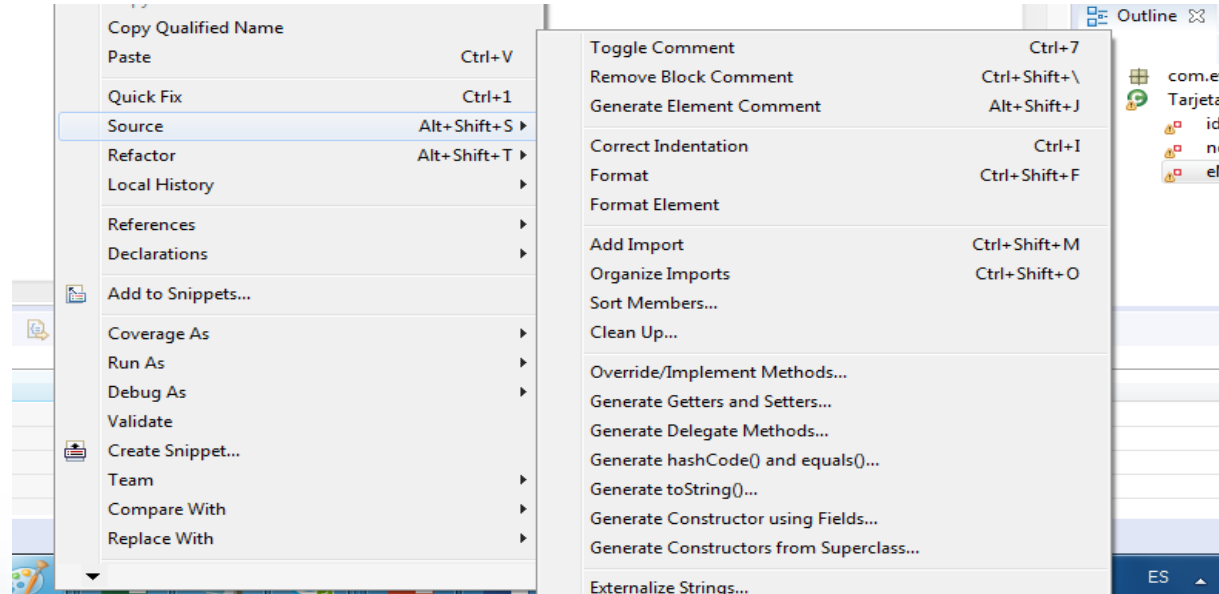
Las clases de Java son los archivos ".java" que contienen el código fuente y que serán posteriormente compilados en archivos ".class"

Veamos como ha quedado nuestra clase *Tarjeta*.  
Le vamos a agregar un constructor, los *getters* y *setters* correspondientes. Esto lo podemos hacer desde la opción *Source -> Generate Constructor using fields*  
*Source -> Generate Getters and Setters*  
Generaremos también de forma automática el método *toString*.

```

Tarjeta.java
1 package com.everis.curso;
2
3 public class Tarjeta {
4     private int id;
5     private String nombre;
6     private String eMail;
7 }
8

```



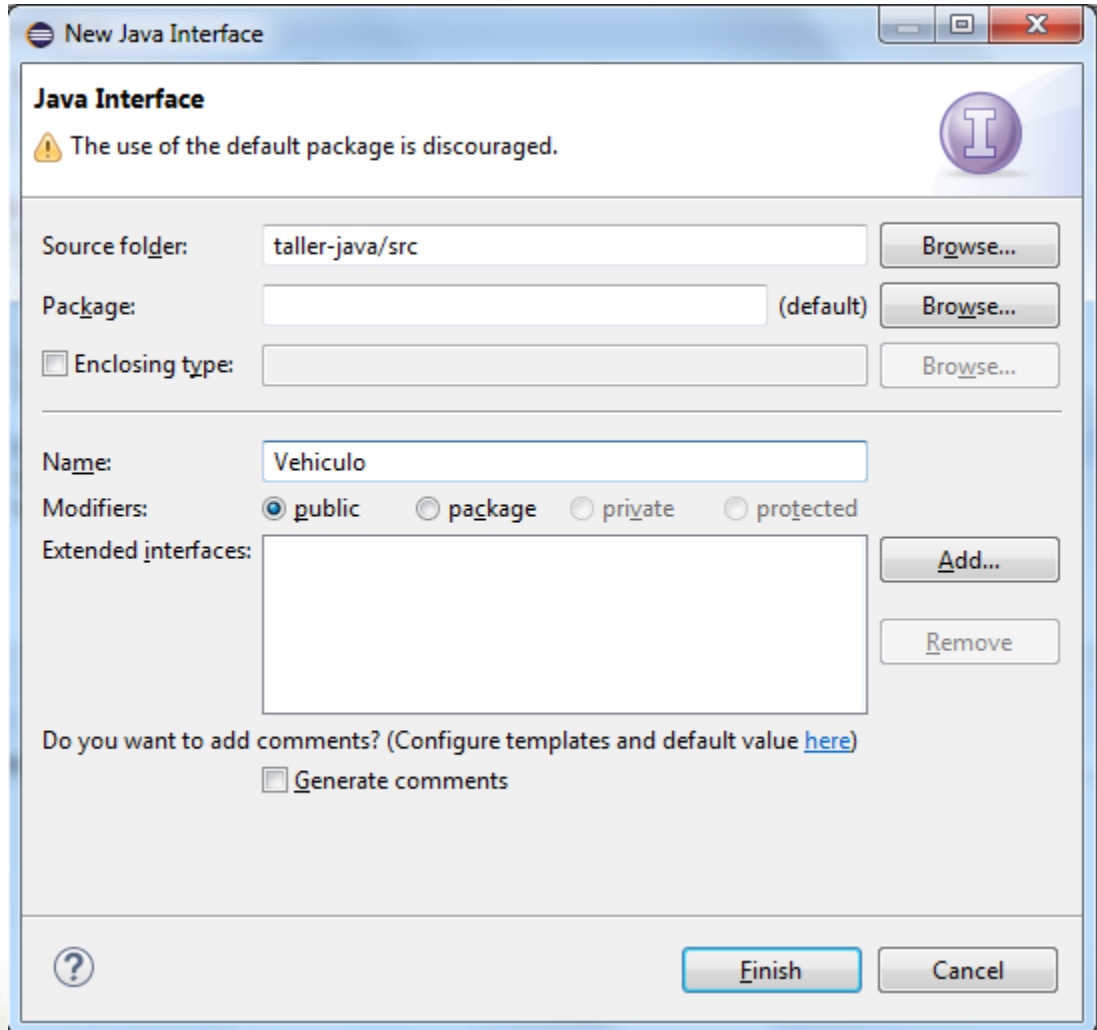
# Eclipse

## Creación de elementos Java

También podemos crear Interfaces, que son entidades abstractas conceptualmente por encima de las Clases, en las que se especifica qué se debe hacer pero no su implementación.

Serán las Clases las que implementen estas interfaces las que describan la lógica del comportamiento de los métodos declarados en la interfaz.

Para crear interfaces nos vamos a botón derecho sobre el proyecto  
*New -> Interface*



**New Java Interface**

**Java Interface**

⚠ The use of the default package is discouraged.

Source folder: taller-java/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: Vehiculo

Modifiers: ☒ public ☐ package ☐ private ☐ protected

Extended interfaces: Add...  
Remove

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Finish Cancel

# Eclipse

Creación de elementos Java

## Ejercicio 1

En el proyecto que hemos creado añadir una clase *Main* en la que declaramos un array de tarjetas, creamos y añadimos un par de tarjetas con los datos que queráis y recorremos el array imprimiendo sus datos en la salida estándar.





04

Funciones  
útiles

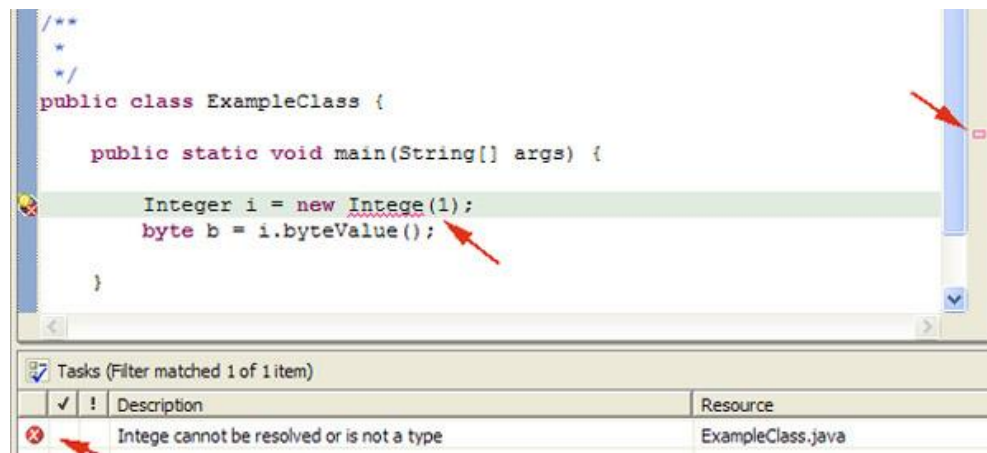


Eclipse presenta una serie de ayudas a la programación que son necesarias conocer dado que ahorra gran cantidad de tiempo y esfuerzo.

### ❑ Compilar y detectar errores

Es importante tener en cuenta que en Eclipse los errores de compilación se muestran en tiempo real subrayando el fragmento de código adecuado con una línea roja. Y además el entorno automáticamente compila los archivos salvados. Así pues, no será necesario pasar por el tedioso y lento proceso de compilar - observar los errores - corregir los errores.

Los errores pueden encontrarse fácilmente porque se muestran además como marcas rojas en el margen derecho del editor de código Java. También los errores y advertencias presentes en archivos ya guardados se muestran dentro de la vista de tareas (Tasks View), como se detallará posteriormente. Haciendo click en cualquiera de los dos tipos de marcadores de error llevará automáticamente hasta la línea en que el error está presente. Las advertencias (warnings) se muestran de la misma manera, pero con marcas amarillas.

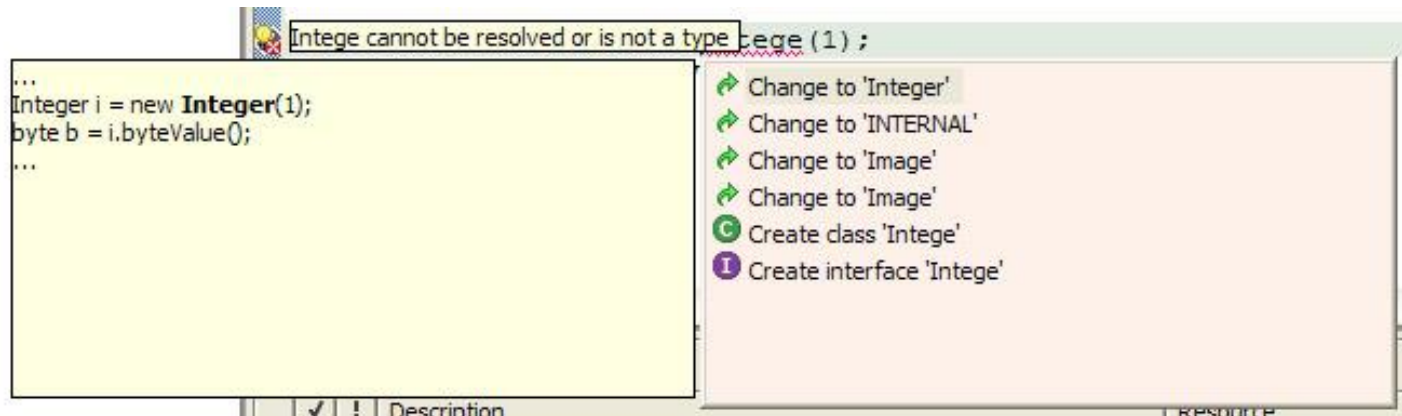


# Eclipse

## Funciones útiles

### Compilar y detectar errores

Hemos visto como Eclipse detecta y marca todo error y advertencia de compilación. Eclipse habitualmente permite autocorregir los posibles errores haciendo clic en el icono de bombilla presente en el margen izquierdo del editor de código. Así pues, aparecerá una ventana mostrando todas las opciones. Seleccionar una opción mediante los cursores del teclado o dejar el punto del ratón sobre dicha opción abrirá una nueva ventana mostrando detalladamente las modificaciones de código que la autocorrección efectuaría. Basta con pulsar la opción seleccionada (o pulsar ENTER) para hacer que Eclipse lleve a cabo la corrección automatizada.



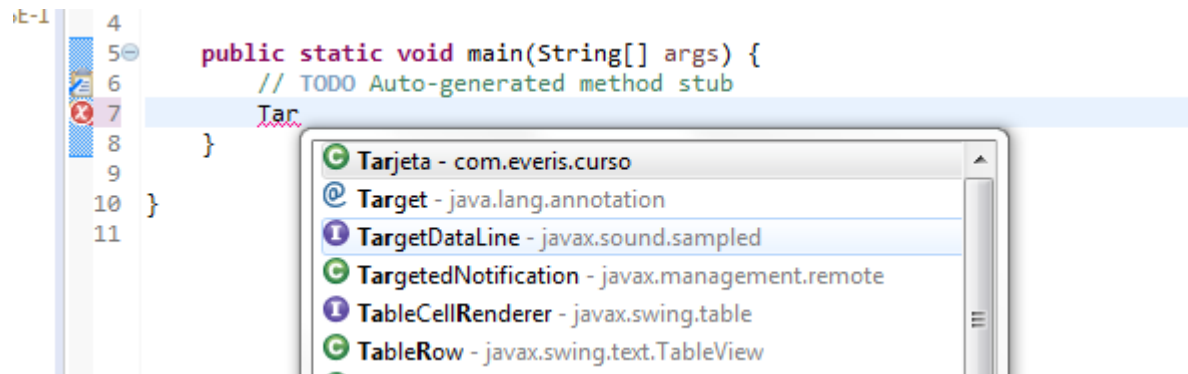
# Eclipse

## Funciones útiles

### CTRL + Espacio = Autocompletar

Esta función permite el autocompletado de código en diferentes aspectos. Lo veremos a través de los siguientes ejemplos prácticos de manera que aprenderemos cuáles son las situaciones más comunes que presenta esta función de ayuda a la programación.

- ☐ Autocompletado de clases.
- ☐ Autocompletado de atributos y variables.
- ☐ Autocompletado de métodos y constructores tras escribir el nombre del objeto.
- ☐ Estructuras lógicas como bucles (for, do, while..)





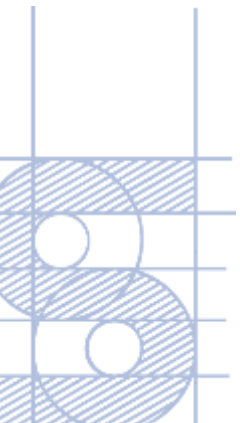
# Eclipse

## Funciones útiles

### Menú *Source*

Con el menú *Source* que hemos utilizado antes, hemos visto que nos ha permitido crear los métodos getter y setter de cada atributo de nuestra clase Tarjeta, un constructor con parámetros y un método toString que retorna nuestro objeto en una cadena formateada sin mucho esfuerzo. También lo podemos usar para:

- ☐ Comment and Uncomment (comentar o descomentar secciones de código)
- ☐ Format (formateo del código)
- ☐ Organise and Add Imports (para eliminar los import no utilizados o añadirlos)
- ☐ Override and Implement Methods (añadirá la signatura de los métodos de la superclase)
- ☐ Surround with try/catch block (seleccionando la parte del código que irá en el try y añade automáticamente el catch)



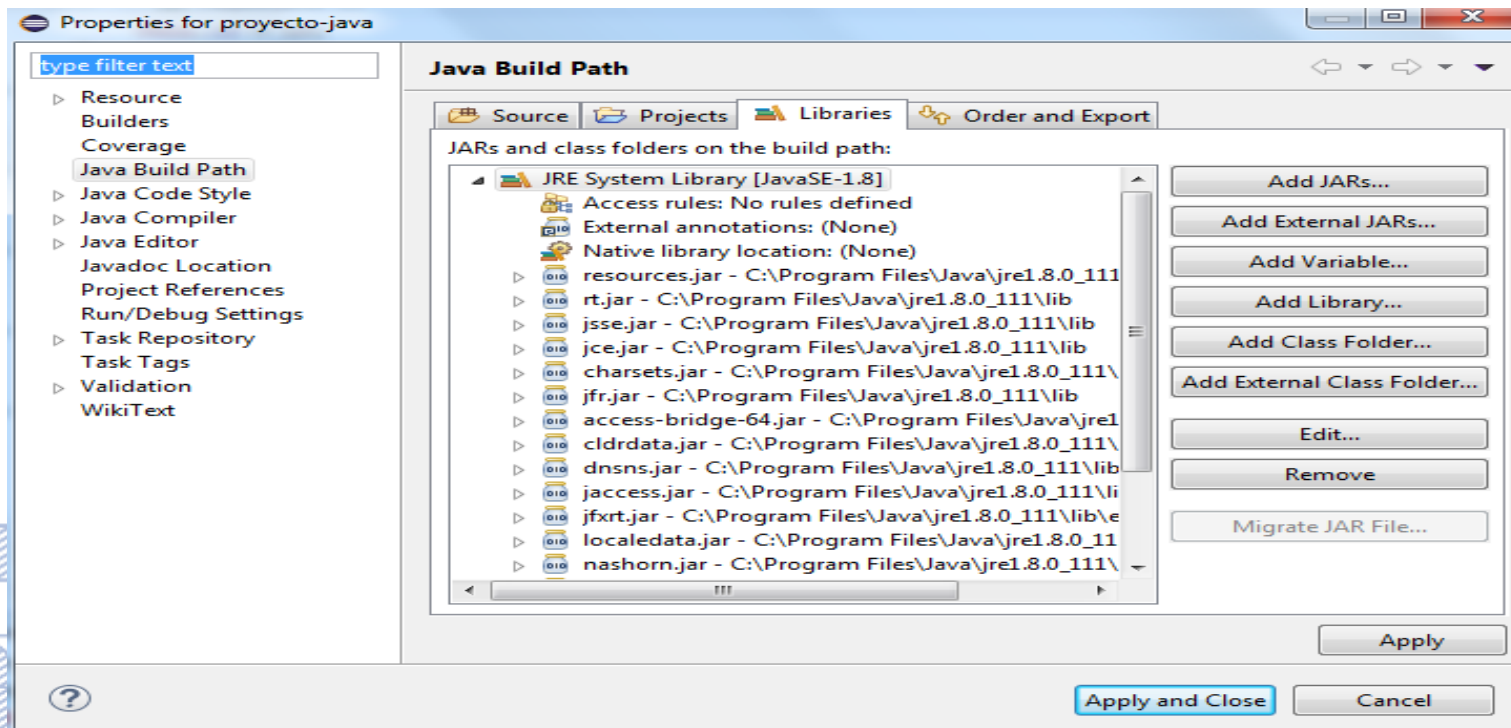
# Eclipse

## Funciones útiles

### Importar Archivos JAR

En ocasiones puede ser necesario importar algunos archivos Jar no incluidos por defecto en el JRE estándar para que el proyecto pueda compilar. Basta con pulsar el

botón derecho del ratón sobre la carpeta adecuada, elegir "Properties > Java Build Path", seleccionar la pestaña "Libraries", pulsar el botón "Add External Jars" y seleccionar el archivo ".jar" o ".zip". El nuevo Jar añadido será visible en la ventana Package Explorer como un pequeño frasco.



# Eclipse

## Vistas de Eclipse

### Refactor

- Refactorización: técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
- Limpiar el código → mejorar la consistencia interna y la claridad.
- Mantenimiento del código
  - No arregla errores
  - No añade funcionalidad
- Se pueden realizar fácilmente cambios en el código.
- Probar trozos de código.
- Código limpio y altamente modularizado.

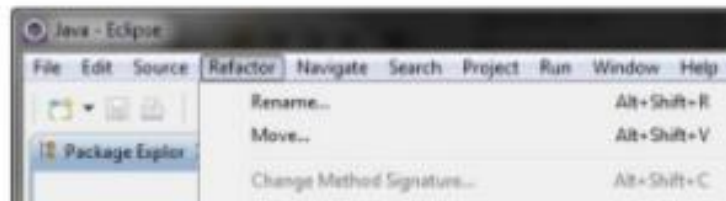
## Opciones de refactorización

- Rename
- Move
- Extract Local Variable
- Extract Constant
- Convert Local Variable to Field
- Convert Anonymous Class to Nested
- Member Type to Top Level
- Extract Interface
- Extract Superclass
- Extract Method
- Inline
- Change Method Signature
- Infer Generic Type Arguments
- Migrate JAR File
- Refactoring Scripts



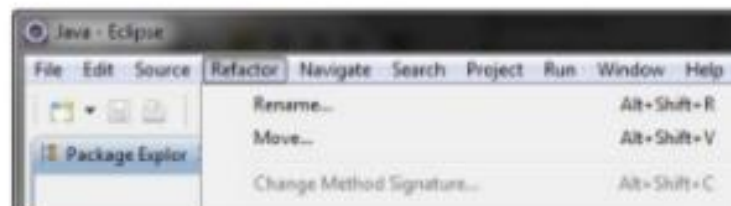
# 1. Rename

- Opción más utilizada.
- Cambia el nombre de variables, clases, métodos, paquetes, directorios y casi cualquier identificador Java.
- Tras la refactorización, se modifican las referencias a ese identificador.
- Refactor > Rename...
- Alt + Shift + R



## 2. Move

- Mover una clase de un paquete a otro.
  - Se mueve el archivo *.java* a la carpeta.
  - Se cambian todas las referencias.
- Arrastrar y soltar una clase a un nuevo paquete.
  - Refactorización automática.
- Refactor > Move...
- Alt + Shift + V



### 3. Extract Local Variable

- Asignar expresión a variable local.
- Tras la refactorización, cualquier referencia a la expresión en el ámbito local se sustituye por la variable.
- La misma expresión en otro método no se modifica.

```
public class ExtractLocalVariable {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```



```
public class ExtractLocalVariable {  
  
    public static void main(String[] args) {  
        String string = "Hello World!";  
        System.out.println(string);  
    }  
}
```

## 4. Extract Constant

- Convierte un número o cadena literal en una constante.
- Tras la refactorización, todos los usos del literal se sustituyen por esa constante.
- Objetivo: Modificar el valor del literal en un único lugar.

```
public class ExtractConstant {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```



```
public class ExtractConstant {  
  
    private static final String HELLO_WORLD = "Hello World!";  
  
    public static void main(String[] args) {  
        System.out.println(HELLO_WORLD);  
    }  
}
```



## 5. Convert Local Variable to Field

- Convierte una variable local en un atributo privado de la clase.
- Tras la refactorización, todos los usos de la variable local se sustituyen por ese atributo.

```
public class ConvertLocalVariable {  
  
    public static void main(String[] args) {  
        String msg = "Hello World!";  
        System.out.println(msg);  
    }  
}
```



```
public class ConvertLocalVariable {  
  
    private static String msg;  
  
    public static void main(String[] args) {  
        msg = "Hello World!";  
        System.out.println(msg);  
    }  
}
```

## 6. Convert Anonymous Class to Nested

- Clase anónima
  - Clase sin nombre de la que sólo se crea un único objeto.
  - No se pueden definir constructores.
- Se utilizan con frecuencia para definir clases y objetos que gestionen los eventos de los distintos componentes de la interfaz de usuario.
- Maneras de definir una clase anónima:
  - Palabra new seguida de la definición de la clase anónima, entre llaves {...}.
  - Palabra new seguida del nombre de la clase de la que hereda (sin extends) y la definición de la clase entre llaves {...}.
  - Palabra new seguida del nombre de la interfaz (sin implements) y la definición de la clase anónima entre llaves {...}.

## 6. Convert Anonymous Class to Nested

- Convierte una clase anónima en una clase anidada de la clase contenedora.

```
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;

public class ConvertAnonymousClass {
    public void createPool() {
        Object threadPool = Executors.newFixedThreadPool(1, new ThreadFactory() {
            @Override
            public Thread newThread(Runnable r) {
                Thread t = new Thread(r);
                t.setName("Worker thread");
                t.setPriority(Thread.MIN_PRIORITY);
                t.setDaemon(true);
                return t;
            }
        });
    }
}
```

## 6. Convert Anonymous Class to Nested

- Asignar nombre de la clase, los modificadores de acceso (*public*, *private*, *protected*) y el tipo (*static*, *final*).
- Refactorización → código más limpio

```
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;

public class ConvertAnonymousClass {

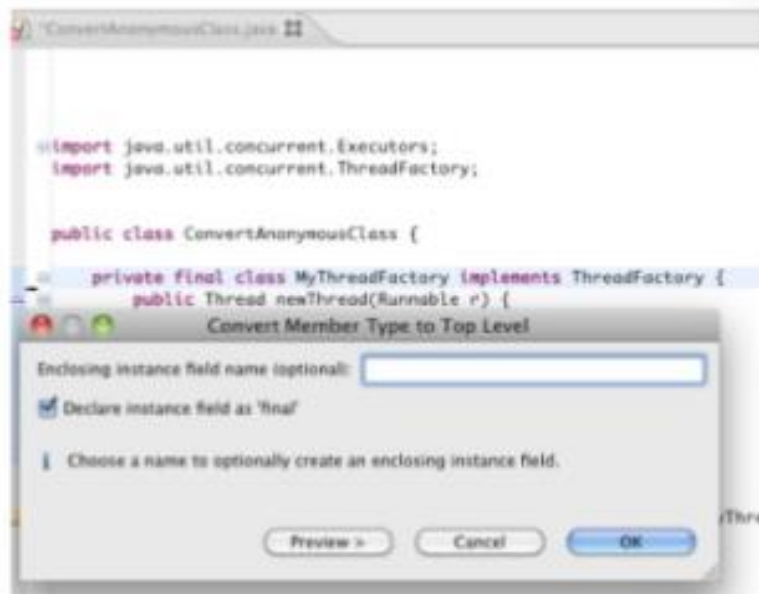
    private final class MyThreadFactory implements ThreadFactory {
        @Override
        public Thread newThread(Runnable r) {
            Thread t = new Thread(r);
            t.setName("Worker thread");
            t.setPriority(Thread.MIN_PRIORITY);
            t.setDaemon(true);
            return t;
        }
    }

    public void createPool() {
        threadPool = Executors.newFixedThreadPool(1, new MyThreadFactory());
    }
}
```



## 7. Member Type to Top Level

- Convierte una clase anidada en una clase de nivel superior con su propio archivo de java.
- Si la clase es estática, la refactorización es inmediata.
- Si no es estática nos pide un nombre para declarar el nombre de la clase que mantendrá la referencia con la clase inicial.



## 8. Extract Interface

- Se crea una interfaz con los métodos de una clase.
- Nos permite escoger que métodos de la clase, formarán parte de la interfaz.



## 9. Extract Superclass

- Extrae una superclase en lugar de una interfaz.
- Si la clase ya utilizaba una superclase, la recién creada pasará a ser su superclase.
- Se pueden seleccionar los métodos que formaran parte de la superclase.
- Diferencia: En la superclase, los métodos están actualmente allí, así que si hay referencias a campos de clase original, habrá fallos de compilación.

## 10. Extract Method

- Nos permite seleccionar un bloque de código y convertirlo en un método.
- Eclipse ajustará automáticamente los parámetros y el retorno de la función.
- Aplicaciones
  - Un método es muy extenso y lo queremos subdividir en diferentes métodos.
  - Un trozo de código se utiliza en diferentes métodos.

## 10. Extract Method

- Seleccionamos el bloque de código que queremos refactorizar.

```
@Override
public Object get(Object key) {
    TimedKey timedKey = new TimedKey(System.currentTimeMillis(), key);
    Object object = map.get(timedKey);
    if (object != null) {
        /**
         * if this was removed after the 'get' call by the worker thread
         * put it back in
         */
        map.put(timedKey, object);
        return object;
    }
    return null;
}
```

- Seleccionamos la opción “Extract Method” (alt+shift+M).
- Configuramos nuestro nuevo método indicando:
  - Nombre
  - Visibilidad (Public, Private, Protected)
  - Tipo de retorno



## 10. Extract Method

- Resultado:

```
@Override
public Object get(Object key) {
    TimedKey timedKey = new TimedKey(System.currentTimeMillis(), key);
    Object object = map.get(timedKey);
    return putIfNotNull(timedKey, object);
}

private Object putIfNotNull(TimedKey timedKey, Object object) {
    if (object != null) {
        /**
         * if this was removed after the 'get' call by the worker thread
         * put it back in
         */
        map.put(timedKey, object);
        return object;
    }
    return null;
}
```

### 11. Inline

- Nos permite ajustar una referencia a una variable o método con la línea en la que se utiliza y conseguir así una única línea de código.
- Cuando se utiliza, se sustituye la referencia a la variable o método con el valor asignado a la variable o la aplicación del método, respectivamente.
- Esto puede ser útil para limpiar nuestro código en diversas situaciones:
  - Cuando un método es llamado una sola vez por otro método, y tiene más sentido como un bloque de código.
  - Cuando una expresión se ve más limpia en una sola línea.

## 11. Inline

- Código a refactorizar con inline:

```
public Object put(Object key, Object value) {
    TimedKey timedKey = new TimedKey(System.currentTimeMillis(), key);
    return map.put(timedKey, value);
}
```

- Posicionamos el cursor en la referencia al método o variable.
- Seleccionamos la opción "Inline" (alt+shift+i).

- Resultado:

```
@Override
public Object put(Object key, Object value) {
    return map.put(new TimedKey(System.currentTimeMillis(), key), value);
}
```

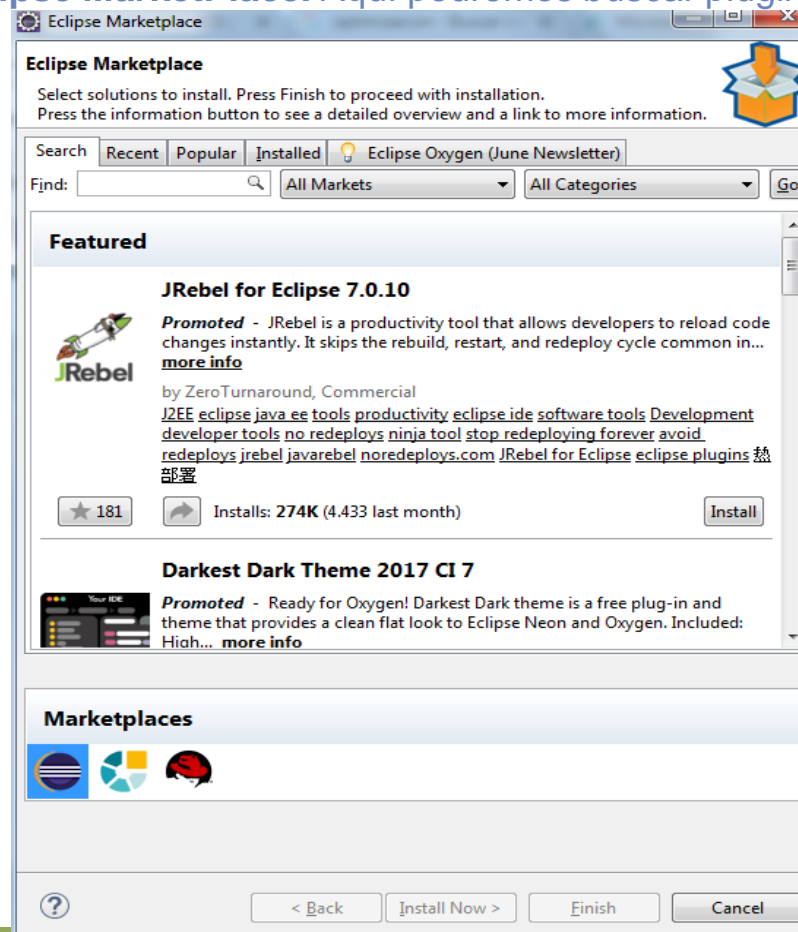
# Eclipse

## Vistas de Eclipse

### Instalación plugins

Para instalar plugins con MarketPlace haremos lo siguiente:

Pinchamos en **Help -> Eclipse Marketplace**. Aquí podremos buscar plugins por categorías o por su nombre.



# Eclipse

## Vistas de Eclipse

### Instalación plugins

- ☐ Pinchamos en **Install**.
- ☐ Nos mostrara lo que contiene el plugin.
- ☐ Nos pide que aceptemos los términos de licencia.
- ☐ Comenzara la instalación.
- ☐ Cuando termine nos pedirá que reiniciemos el programa.
- ☐ El plugin ya estará instalado y se configurara según el plugin.



05

Vistas de  
Eclipse

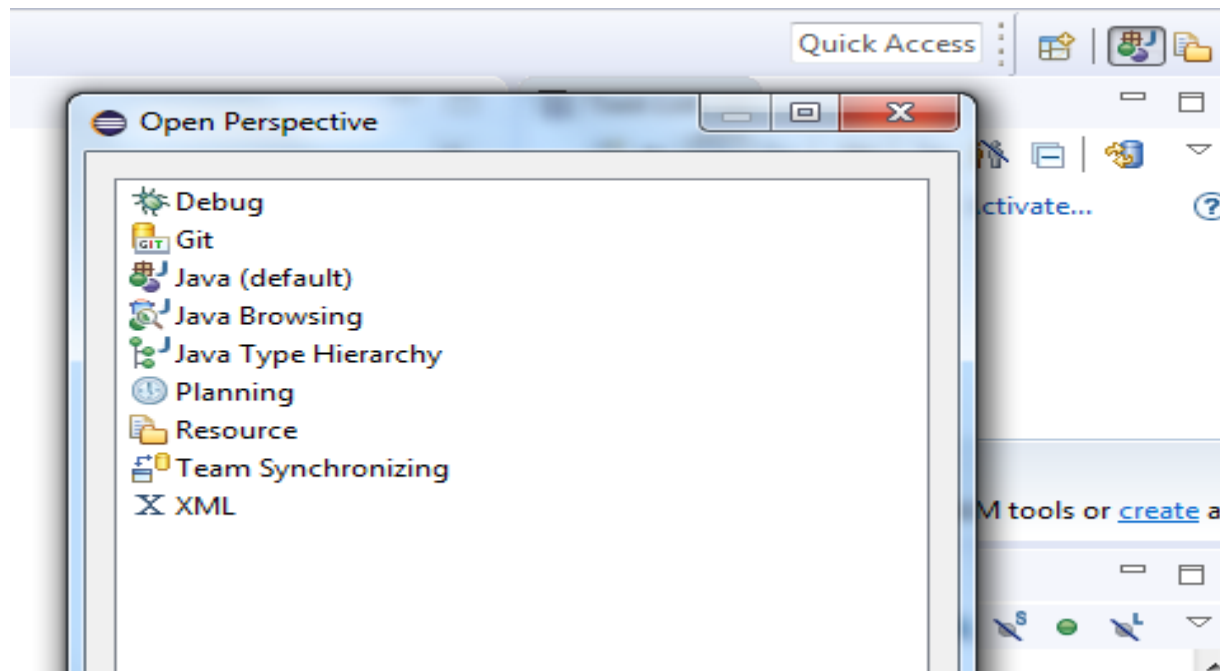


# Eclipse

## Vistas de Eclipse

### Perspectivas

Una perspectiva de Eclipse es una agrupación de vistas y editores de manera que den apoyo a una actividad completa del proceso de desarrollo software. Sin embargo, es posible crear perspectivas propias añadiendo nuevas vistas y cambiando su distribución en la pantalla. Las perspectivas pueden seleccionarse haciendo clic a la esquina superior derecha

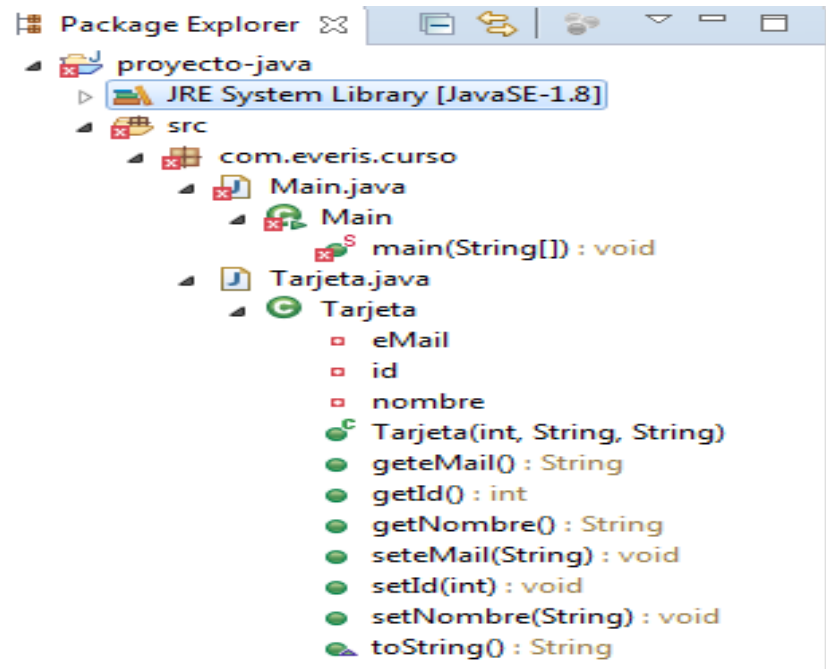


# Eclipse

## Vistas de Eclipse

### Package Explorer

La vista del explorador de paquetes muestra la estructura lógica de paquetes y clases Java almacenados en los distintos proyectos. las carpetas fuente (que deben almacenar los archivos fuente ".java") se muestran aquí decoradas con el icono de un paquete contenido. Los archivos Java también pueden ser expandidos de modo que muestren sus métodos y atributos internos al pulsar la flechita de al lado.



# Eclipse

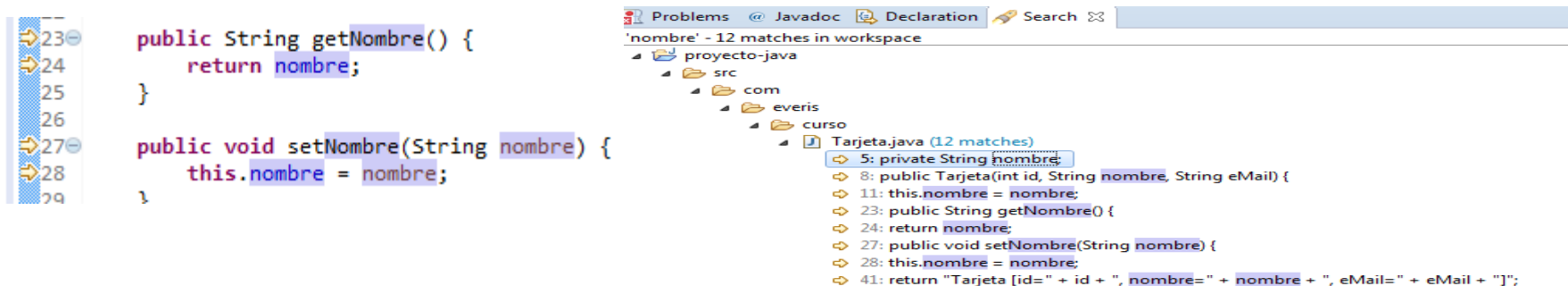
## Vistas de Eclipse

### Search View

Para realizar una búsqueda dentro de Eclipse, el menú "Search" de la barra superior de menús debería ser seleccionado. También se pueden lanzar búsquedas pulsando el icono de linterna.

- ☐ La búsqueda de archivos "File Search" es una búsqueda textual que puede ser ejecutada sobre archivos de todo tipo. Es equivalente a una búsqueda tradicional.
- ☐ La búsqueda de ayuda "Help Search" efectúa búsquedas dentro de la ayuda de Eclipse.
- ☐ La búsqueda de Java "Java Search" es similar a la búsqueda de archivos pero proporciona funciones adicionales para buscar en archivos Java. Así pues, permite buscar explícitamente por tipos, métodos, paquetes, constructores y campos, usando restricciones de búsqueda adicionales (como por ejemplo, buscar sólo el punto del código en que se declararon los elementos coincidentes).

Los resultados de la búsqueda aparecen en la vista "Search". También se subrayan en gris dentro del editor de código, con una flecha amarilla en el margen izquierdo y con una marca gris en el margen derecho. Haciendo clic en cualquiera de estos elementos seremos conducidos al punto en que la cadena buscada se encontró.



The screenshot shows the Eclipse IDE with a search for the word "nombre". The search results are displayed in the "Search" view on the right, showing 12 matches in the file "Tarjeta.java". The code editor on the left shows the corresponding code with the search results highlighted in grey and yellow arrows pointing to the matches.

```

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

```

The search results in the "Search" view are as follows:

- 5: private String nombre;
- 8: public Tarjeta(int id, String nombre, String eMail) {
- 11: this.nombre = nombre;
- 23: public String getNombre() {
- 24: return nombre;
- 27: public void setNombre(String nombre) {
- 28: this.nombre = nombre;
- 41: return "Tarjeta [id=" + id + ", nombre=" + nombre + ", eMail=" + eMail + "];

06

Ejecutar y  
depurar



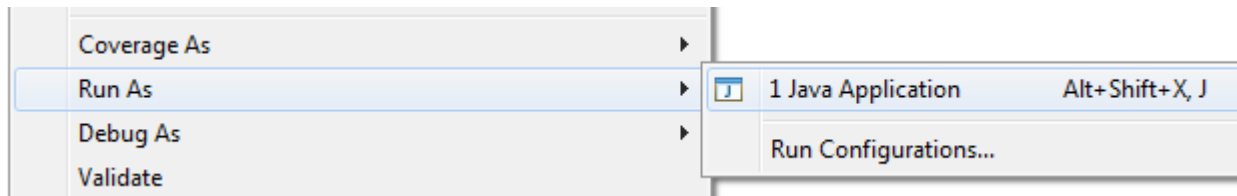


# Eclipse

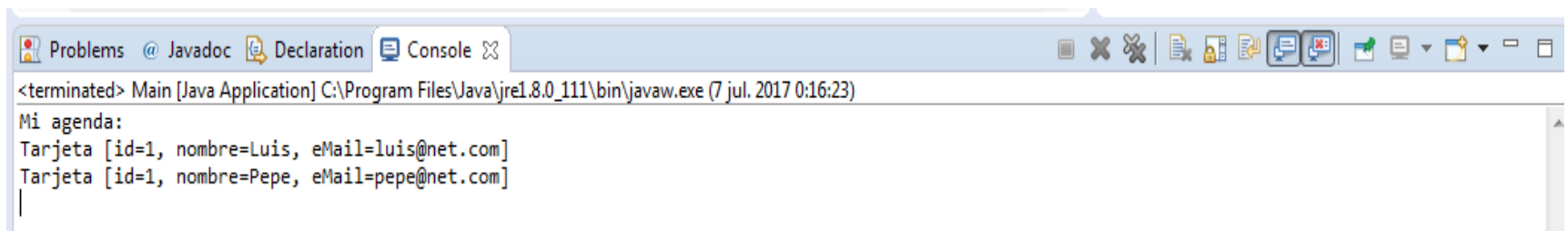
## Ejecutar y depurar

### Ejecución de proyectos

Para ejecutar un proyecto sobre el haremos clic derecho e iremos a *Run As -> Java Application*



Ejecutar el proyecto del ejercicio anterior a ver qué sucede. Los resultados deberían mostrarse en la consola en el margen inferior del espacio de trabajo

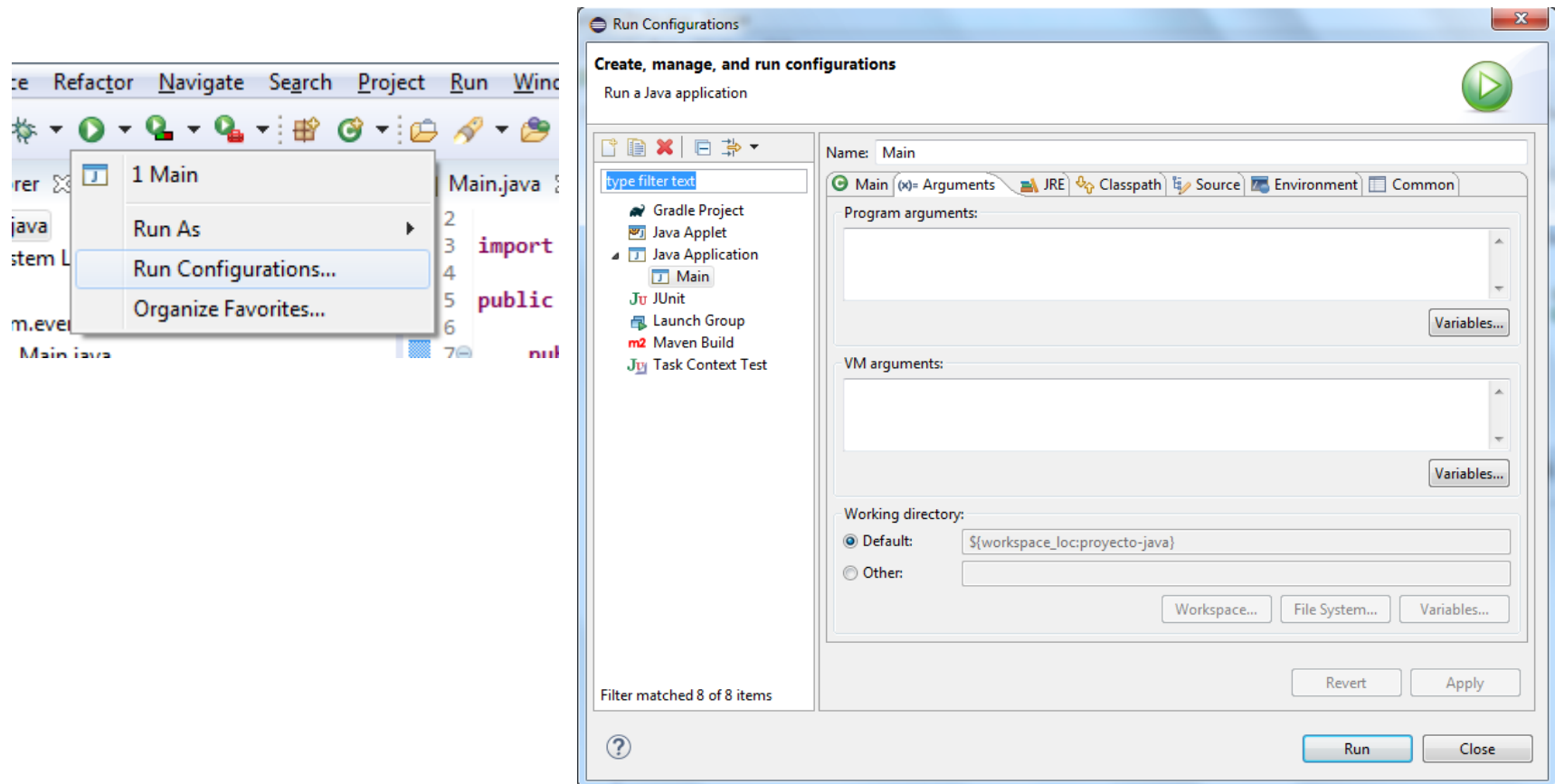


# Eclipse

## Ejecutar y depurar

### Ejecución de proyectos

Desde el menú superior podemos acceder a la configuración de ejecución. Podemos por ejemplo pasar argumentos a la función Main.



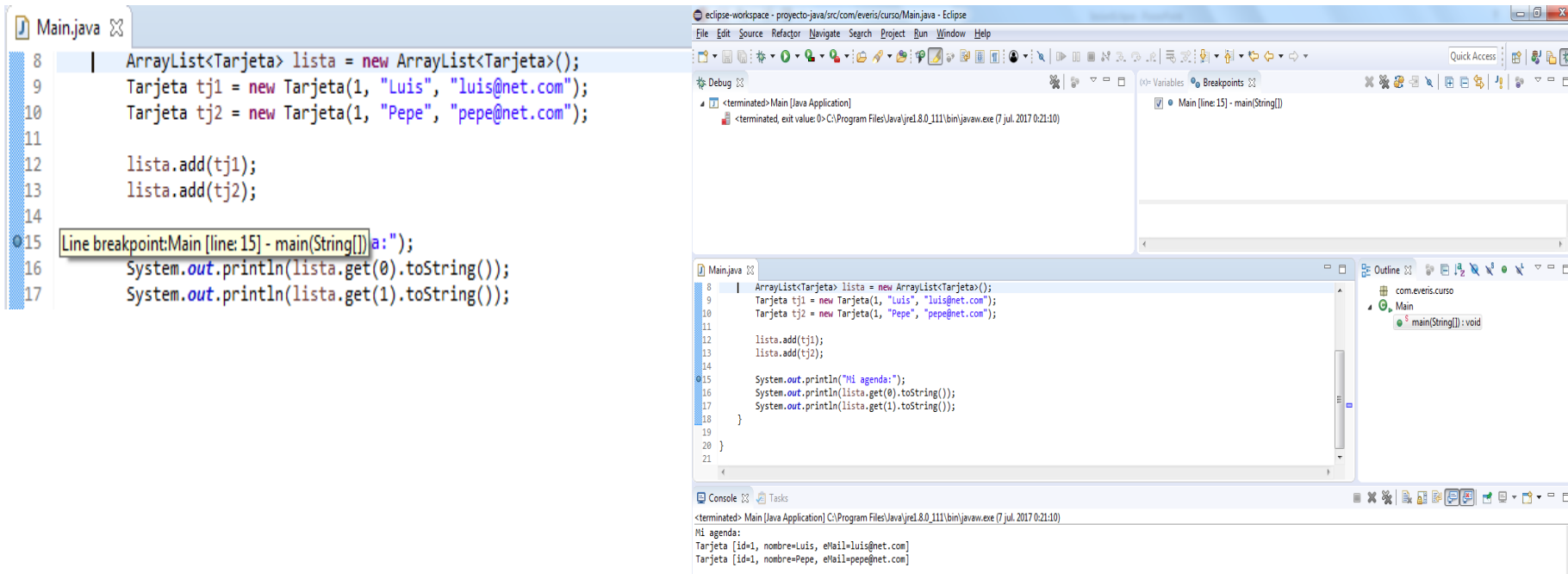
# Eclipse

## Ejecutar y depurar

### Depurar proyectos

Aunque Java no es tan difícil de depurar como otros lenguajes de programación, también es perfectamente posible que surjan complejos problemas de ejecución. Eclipse da apoyo completo a la tarea de depuración a través de su perspectiva "Debug" ("Window > Open Perspective > Debug" o seleccionando el icono del "bicho" en el menú).

Los puntos de ruptura marcan líneas en que la ejecución del programa se detendrá de manera que sea posible comprobar el valor de las variables en ese instante, identificando así posibles errores.



The screenshot displays the Eclipse IDE interface with a Java project named 'proyecto-java'. The main editor shows the file 'Main.java' with the following code:

```

8 | ArrayList<Tarjeta> lista = new ArrayList<Tarjeta>();
9 | Tarjeta tj1 = new Tarjeta(1, "Luis", "luis@net.com");
10 | Tarjeta tj2 = new Tarjeta(1, "Pepe", "pepe@net.com");
11 |
12 | lista.add(tj1);
13 | lista.add(tj2);
14 |
15 | Line breakpoint:Main [line: 15] - main(String[]):a:");
16 | System.out.println(lista.get(0).toString());
17 | System.out.println(lista.get(1).toString());

```

A breakpoint is set at line 15. The 'Debug' perspective is active, showing the 'Main' application running. The console output is as follows:

```

<terminated> Main [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (7 jul. 2017 0:21:10)
Ni agenda:
Tarjeta [id=1, nombre=Luis, email=luis@net.com]
Tarjeta [id=1, nombre=Pepe, email=pepe@net.com]

```

# Eclipse

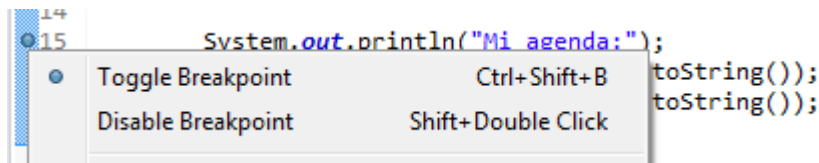
## Ejecutar y depurar

### Depurar proyectos

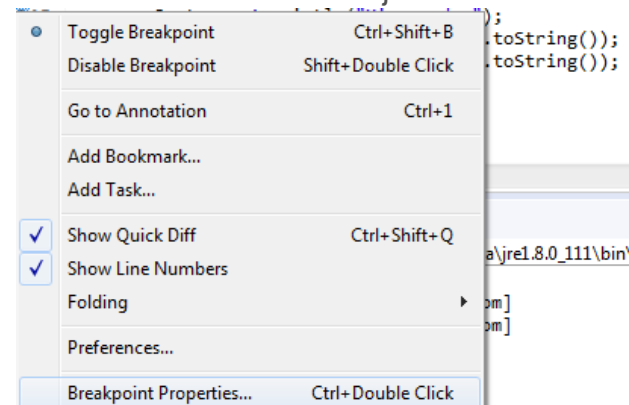
Los puntos de ruptura pueden eliminarse todos desde la sección superior derecha con *Remove All*



O desde el editor podemos eliminarlos clicando sobre ellos con *Disable breakpoint*



Haciendo clic derecho en un punto de ruptura y seleccionando "Breakpoint Properties..." permitirá especificar opciones avanzadas del punto de ruptura. "Hit Count" especifica que la ejecución del programa se detendrá cuando se pase por el punto de ruptura el número especificado de veces. Las condiciones de activación detendrán la ejecución cuando la condición sea cierta o bien cuando el valor de la condición cambie.

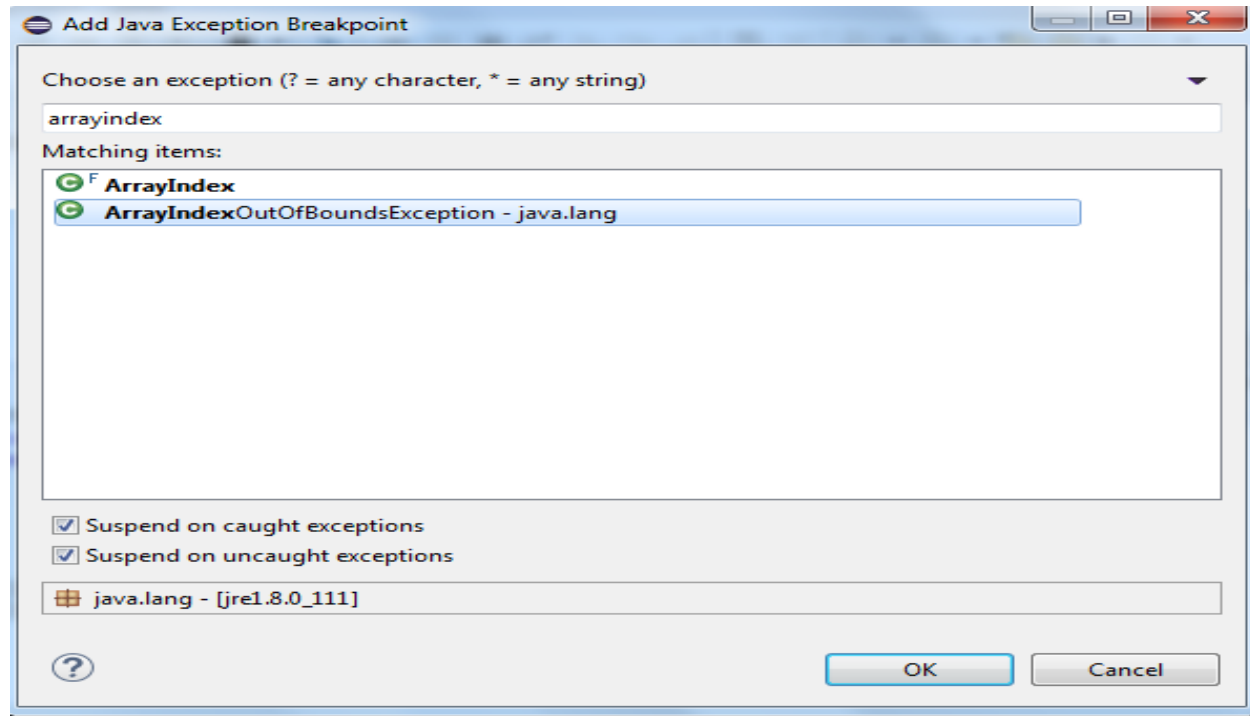


# Eclipse

## Ejecutar y depurar

### Depurar proyectos

Las excepciones son uno de los síntomas más evidentes de errores de ejecución. Los "Java Exception Breakpoints" detienen la ejecución cuando salta una excepción del tipo seleccionado. Estos puntos de ruptura se activan haciendo clic en el icono "J!" de la vista de "Breakpoints" o desde el menú principal "Run". La ejecución puede detenerse cuando la excepción sea capturada, no capturada o ambas. Añadir siempre los puntos de ruptura de excepciones Java de "ArrayIndexOutOfBoundsException" (lanzada cuando el índice de una matriz se sale de sus dimensiones) y "NullPointerException" (lanzada cuando se intenta acceder a una referencia que apunta a null) es una práctica de depuración recomendada.



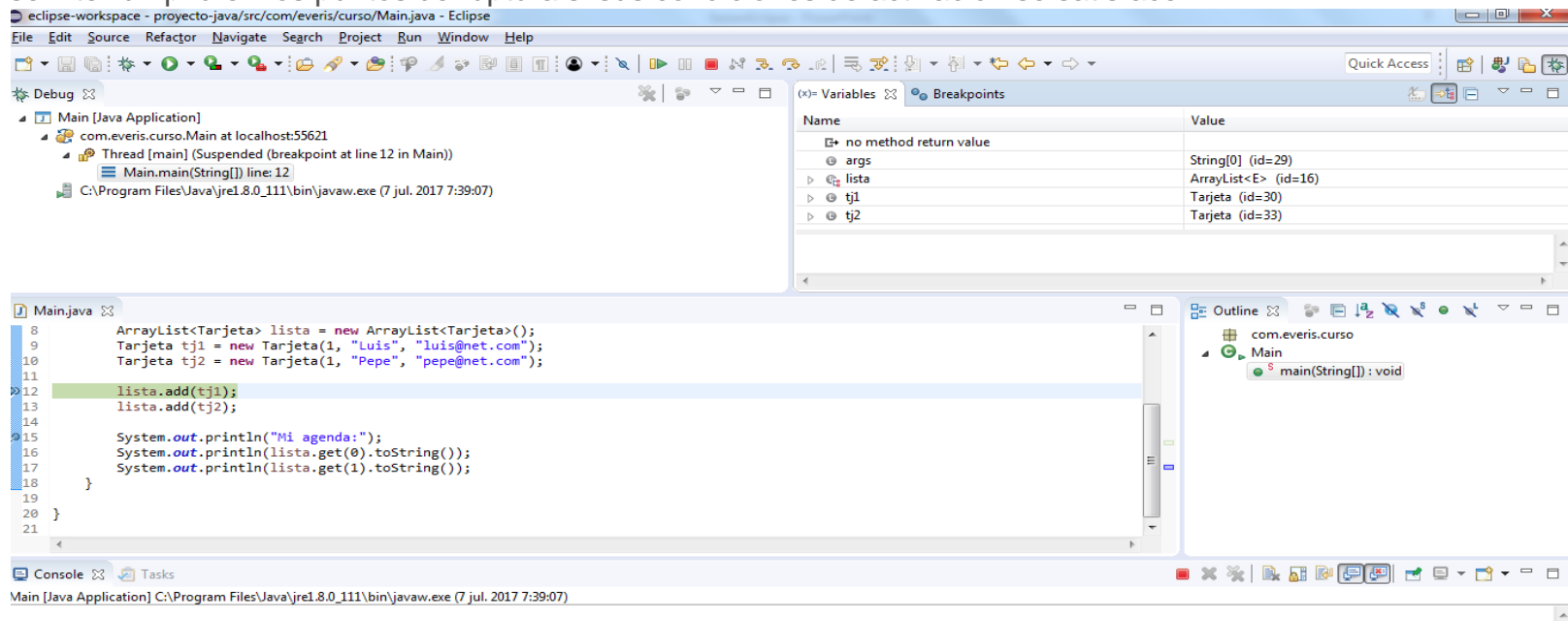


# Eclipse

## Ejecutar y depurar

### Depurar proyectos

Si se desea que el programa se detenga en los puntos de ruptura definidos deberá ser ejecutado en modo depuración ("Run > Debug..."). Tras detenerse en un punto de ruptura la ejecución del programa puede continuar de diversas maneras. Haciendo clic derecho en el editor de código dentro de la perspectiva de depuración aparecerá un menú contextual con estas opciones. "Run to line" reanuda la ejecución del programa hasta que se alcanza la línea en que está el cursor. "Step into selection" continuará la ejecución dentro del método seleccionado siempre y cuando el código fuente del método esté disponible. La ejecución también puede reanudarse mediante un clic derecho en la ventana de "Debug" y seleccionando las opciones adecuadas, o directamente pulsando los iconos de dicha ventana. "Step over" parará en la línea siguiente a la invocación de un método. "Resume" reanudará la ejecución normal del programa y sólo se interrumpirá en los puntos de ruptura si sus condiciones de activación se satisfacen.



The screenshot shows the Eclipse IDE with the following components:

- Debugger View:** Shows the 'Main' thread [main] (Suspended (breakpoint at line 12 in Main)) at line 12 of 'Main.main(String[])'. The execution path is C:\Program Files\Java\jre1.8.0\_111\bin\javaw.exe (7 jul. 2017 7:39:07).
- Variables Window:** Displays the state of the program.
 

Name	Value
no method return value	
args	String[0] (id=29)
lista	ArrayList<E> (id=16)
tj1	Tarjeta (id=30)
tj2	Tarjeta (id=33)
- Outline View:** Shows the package structure 'com.everis.curso' and the 'Main' class with the 'main' method.
- Editor View:** Shows the source code of 'Main.java' with the following content:
 

```

8  ArrayList<Tarjeta> lista = new ArrayList<Tarjeta>();
9  Tarjeta tj1 = new Tarjeta(1, "Luis", "luis@net.com");
10 Tarjeta tj2 = new Tarjeta(1, "Pepe", "pepe@net.com");
11
12 lista.add(tj1);
13 lista.add(tj2);
14
15 System.out.println("Mi agenda:");
16 System.out.println(lista.get(0).toString());
17 System.out.println(lista.get(1).toString());
18 }
19
20 }
21

```
- Console View:** Shows the output of the program:
 

```

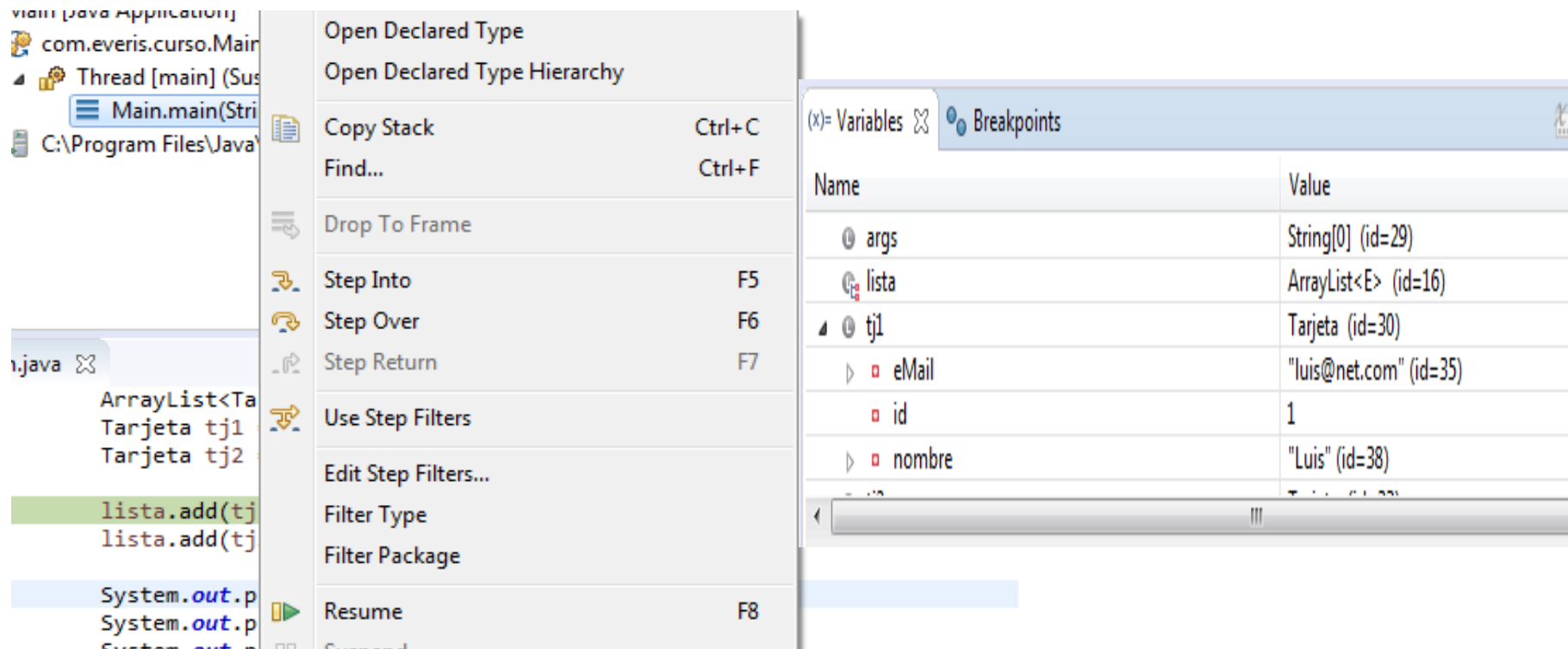
Main [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (7 jul. 2017 7:39:07)

```

# Eclipse

Ejecutar y depurar

## Depurar proyectos



The screenshot shows the Eclipse IDE interface during a debug session. The left sidebar displays the Project Explorer with the package structure `com.everis.curso.Main` and the `Main.main(String[] args)` method. The central editor shows the `Main.java` file with a breakpoint set on the line `lista.add(tj1);`. The right sidebar contains the Debug console and the Variables view.

**Debug Console:**

```

main [Java Application]
com.everis.curso.Main
Thread [main] (Suspended)
Main.main(String[])
C:\Program Files\Java\

```

**Variables View:**

Name	Value
args	String[0] (id=29)
lista	ArrayList<E> (id=16)
tj1	Tarjeta (id=30)
eMail	"luis@net.com" (id=35)
id	1
nombre	"Luis" (id=38)

**Java File (Main.java):**

```

ArrayList<Tarjeta> lista = new ArrayList<Tarjeta>();
Tarjeta tj1 = new Tarjeta(1, "Luis", "luis@net.com");
Tarjeta tj2 = new Tarjeta(2, "Maria", "maria@net.com");

lista.add(tj1);
lista.add(tj2);

System.out.println("Lista de tarjetas:");
System.out.println(lista);

```

# Eclipse

## Ejercicios

### Ejercicio:

En el proyecto *taller-java* creado en el Ejercicio 2.

Crear una clase Coche que implemente la interfaz Vehículo. Tendrá como atributo la *velocidad* (int).

- ☐ Implementaremos la lógica de la función *frenar* de la clase Coche restaremos a la velocidad en base al parámetro de la cantidad que viene por parámetro y retornaremos un mensaje estilo “El coche ya ha frenado y ya va a X km/hora”.
- ☐ Para implementar la función *acelerar* de la clase Coche sumaremos a la velocidad en base al parámetro de la cantidad que viene por parámetro y retornaremos un mensaje estilo “El coche ha acelerado y va a X km/hora”. Caso que supere la velocidad máxima deberá indicarlo también en el mensaje de retorno.
- ☐ Añadiremos una función *plazas()*, que retornará un entero de 5.

Crear una clase Moto que implemente la interfaz Vehículo. Similar que la clase Coche pero sustituyendo los mensajes de retorno propios para moto. Y el número de plazas que devolverá la función *plazas()* será de 2.

Finalmente crear una clase Main en la que crearemos un array de Vehículos, añadiéndole 6 de ellos entre Coches y Motos y utilizaremos las funciones propias de acelerar y frenar para actualizar sus velocidades. Recorrer el array mostrando las velocidades de los vehículos creados.