# Programación II

Introducción a Eclipse

Versión 1.1

Facultad de Ingeniería y Ciencias Exactas
Universidad Argentina de la Empresa


UADE

# Índice General

# Unidad 1

# Instalación de Eclipse

Eclipse es una plataforma de desarrollo open source basada en Java, que puede ser utilizada para desarrollar aplicaciones en Java como así también en otros lenguajes. Para poder ejecutar el programa Eclipse vamos a necesitar una JVM (Java Virtual Machine, o Máquina Virtual de Java) para lo cual instalaremos un JDK (Java Development Kit, o herramientas de desarrollo para Java.

## 1.1 JDK

Se puede descargar desde **java.sun.com**. Tener en cuenta que hay distintas versiones y que, por razones de compatibilidad con la versión instalada en los Laboratorios, recomendamos obtener la versión JDK 1.5, también denomina JDK 5.

## 1.2 Eclipse

Se descarga el programa Eclipse desde **www.eclipse.org** en forma de archivo ZIP y sólo tenemos que descomprimir este archivo en la carpeta donde queramos tenerlo instalado. Para ejecutarlo solo hay que ejecutar el programa Eclipse.exe.

# Unidad 2

# Utilización de Eclipse

## 2.1 Abriendo Eclipse

En Windows se puede invocar al programa Eclipse con sólo hacer doble clic en el ícono en forma de sol si tiene creado el acceso directo en el escritorio, de lo contrario invocándolo desde: **Inicio → Mi PC → Disco Local (C:) → Archivos de Programas → Eclipse-SDK 3....**

## 2.2 Espacio de trabajo (Workspace)

Eclipse almacena todos los archivos de código fuentes que se vayan a crear más algunos archivos propios en lo que denomina **workspace** o espacio de trabajo. Esto es un espacio físico en el disco rígido de la computadora. Por este motivo al iniciar una sesión con Eclipse deberemos indicar la dirección del espacio de trabajo a emplear (Fig. 2.1).

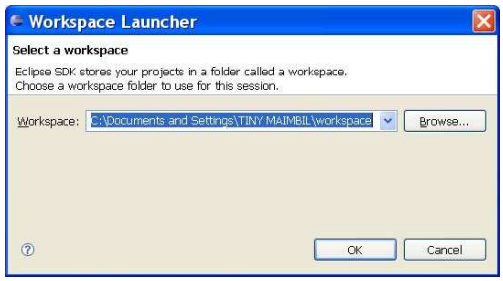


Figura 2.1: Especificación de workspace

Posteriormente, en este espacio de trabajo se incluirán todos los archivos de trabajos que se realicen en Eclipse. La carpeta workspace suele ubicarse en el directorio raíz de Eclipse. Sin embargo es recomendable instalar el workspace en una ubicación independiente de los archivos del programa. Esto hace más fácil la posterior actualización de las versiones de Eclipse y la realización de copias de seguridad del espacio de trabajo. Es posible cambiar a un espacio de trabajo diferente con solo volver a esta ventana con la opción: **File → Switch Workspace...**

## 2.3 Pantalla de bienvenida

Una vez confirmado el espacio de trabajo a ser empleado se accede a la pantalla de bienvenida, donde se ofrecen distintas opciones de información como páginas de ayuda, cursos prácticos, programas de ejemplo, etc, Dependiendo de la configuración de inicio de Eclipse, es posible que esta ventana no aparezca pasando directamente el espacio de trabajo seleccionado. En caso de querer visualizar esta pantalla de bienvenida para poder acceder a sus opciones podrá hacerlo desde: **Help** → **Welcome** Con lo que se visualizará una pantalla con la forma de la Fig. 2.2.

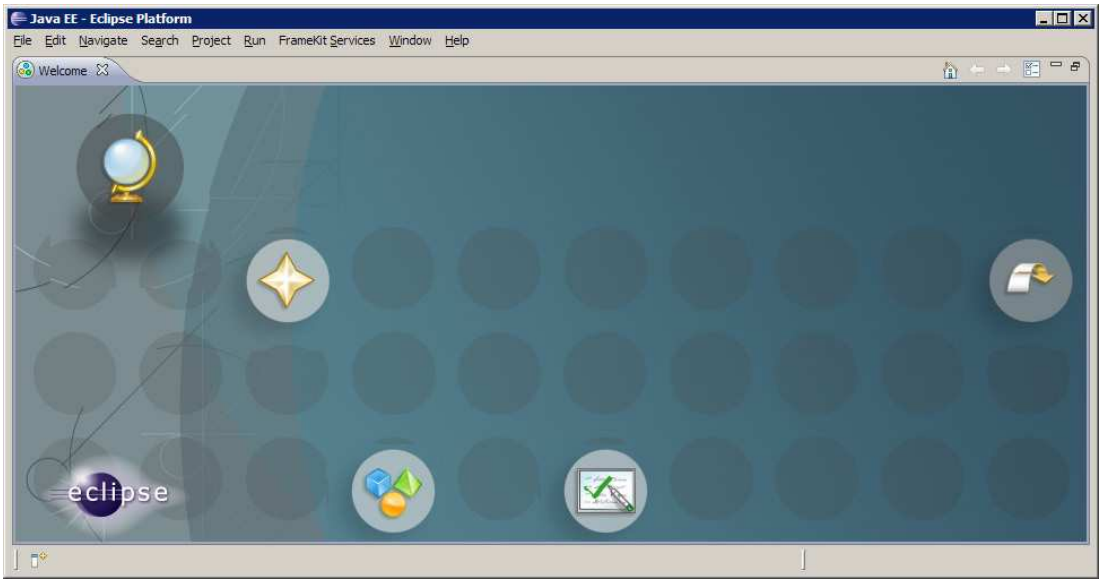


Figura 2.2: Pantalla inicial

En la sección **Overview** encontrará varios capítulos de distintas guías de usuario del sistema de ayuda de Eclipse.

## 2.4 Perspectiva Java

Para acceder a la mesa de trabajo (workbench) desde esta pantalla, sólo debe seleccionar el ícono con la opción **Workbench** (o bien cerrar la pestaña welcome).
Eclipse provee distintas perspectivas para la mesa de trabajo. Una **perspectiva** es una combinación de ventanas y herramientas orientadas a tareas concretas. Dado que estamos interesados en desarrollar en Java, podemos debemos seleccionar la perspectiva Java, para ello haga clic en: **Window** → **Open Perspective** → **Java**
A continuación se accederá a una pantalla con aspecto similar al de la Fig. 2.3. En esta perspectiva de uso de la mesa de trabajo verá las siguientes ventanas:

- Package Explorer: La vista del explorador de paquetes muestra la estructura lógica de paquetes y clases Java almacenados en los distintos proyectos.
- Hierarchy: La vista de jerarquía muestra las relaciones de herencia presentes entre distintos elementos de Java. Haciendo clic derecho en el nombre de una clase Java

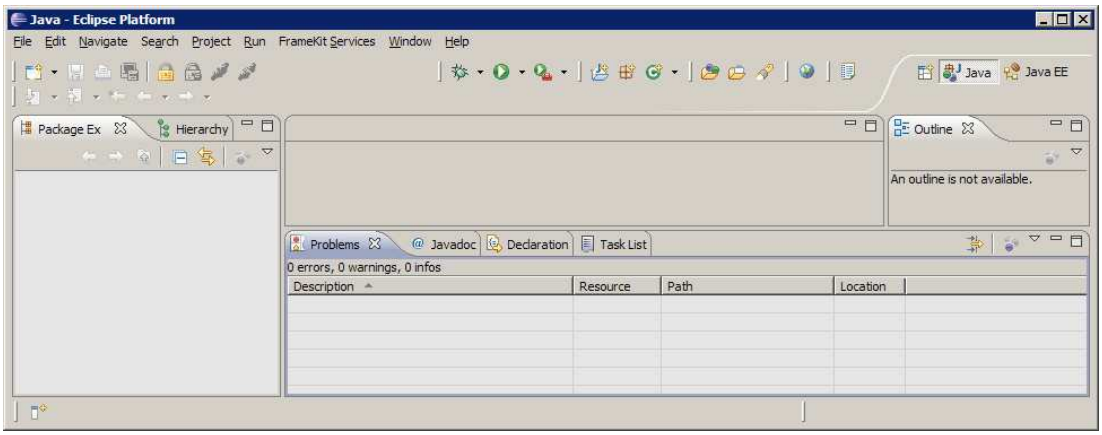


Figura 2.3: Perspectiva Java

en el editor de código y seleccionando "Open Type Hierarchy" abrirá esta vista de jerarquía. La tecla rápida asociada es "F4".

- Los elementos del Menú y los íconos de herramientas habituales
- El ícono que representa la perspectiva activa (ángulo superior derecho)
- La ventana Outline: La vista de resumen es una forma rápida de ver qué métodos y atributos se encuentran definidos dentro de una clase de Java. Los iconos asociados proporcionan información adicional de acuerdo con la visibilidad del atributo o método en cuestión. Y sólo con hacer clic en cualquiera de estos iconos conducirá a la línea de código exacta en que dicho atributo o método está definido.
- La ventana inferior derecha donde se puede visualizar, entre otros, la ventana con salida por Consola y la ventana con los problemas dectectados a nivel de compilación.

En caso de cerrar alguna de estas ventanas y querer visualizarla nuevamente se puede hacer desde: **Window** → **show view** → **Console/...**. También es posible restaurar la configuración inicial de la perspectiva con la opción: **Window** → **reset perspective**.

# Unidad 3

# Desarrollando en Eclipse

Dentro del espacio de trabajo, los programas o archivos fuentes se almacenan en proyectos. Un **proyecto** es un conjunto de archivos relacionados entre si que sirven a un mismo fin. Como los archivos dentro de un proyecto pueden ser muchos, Java nos propone una opción para separar los archivos dentro de un mismo proyecto, estos son los **paquetes**. Para nuestros ejemplos utilizaremos un único workspace, con un único proyecto y uno o más paquetes según se considere necesario.

A continuación, se detallan los pasos para poder escribir un programa en java utilizando Eclipse.

## 3.1 Proyecto nuevo

Antes de crear un nuevo programa, primero debemos crear un proyecto de Java donde va a residir este programa. En la barra de herramientas haga clic en el ícono **New Java Project** tras lo cual visualizará la pantalla de la Fig. 3.1

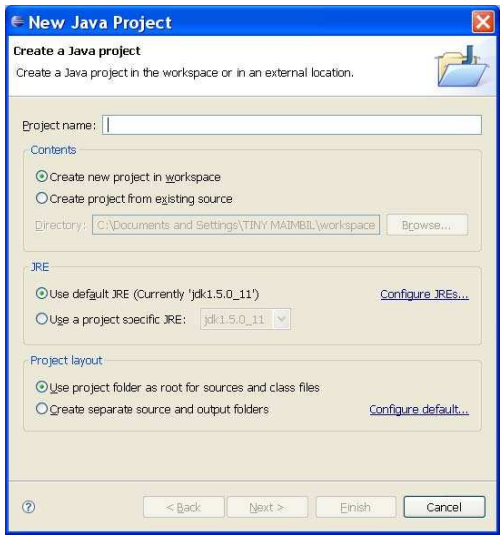


Figura 3.1: Nuevo Proyecto Java

Introducir el **nombre** con el que desea identificar al nuevo proyecto, por ejemplo "ProgII", y luego haga clic en **Finish**, tras lo cual en el **Explorer Package** aparece el símbolo del nuevo proyecto con el nombre correspondiente que será la entrada para el proyecto.

## 3.2 Creación de un paquete

Ahora es necesario crear un paquete donde se incluirán los archivos fuentes (clases, interfaces, etc) que conformarán el proyecto. Para ello haremos lo siguiente: seleccionado el símbolo del nuevo proyecto recién creado (ProgII), hacer clic con el botón derecho y seleccionar la opción: **New** → **Package** de la manera en que se muestra en la pantalla Fig. 3.2.

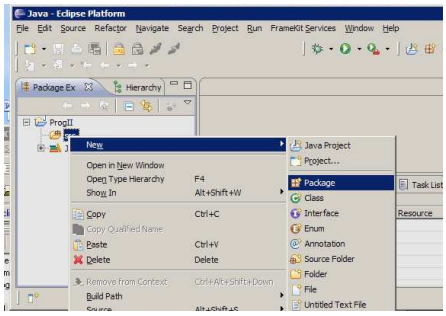


Figura 3.2: Nuevo paquete

Inmediatamente aparece la pantalla de creación del nuevo paquete como se muestra en la Fig. 3.3.

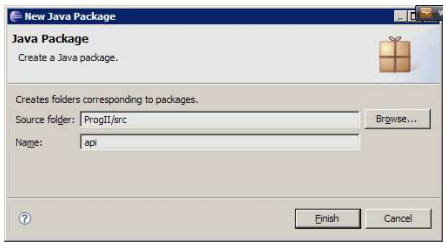


Figura 3.3: Definición de nuevo paquete

Donde se debe incorporar el nombre para identificar al nuevo paquete, que se recomienda sea escrito en letras minúsculas, por ejemplo "api". Luego: **Finish**.

## 3.3 Creación de una interface

Para crear una interface, parados sobre el nombre del paquete recién creado, con el botón derecho seleccionamos: **New** → **Interface** como se visualiza en la Fig. 3.4
En el recuadro que aparece se deberá escribir el nombre de la interface a crear (la interface representa la definición del tipo de datos abstractos a desarrollar, como por

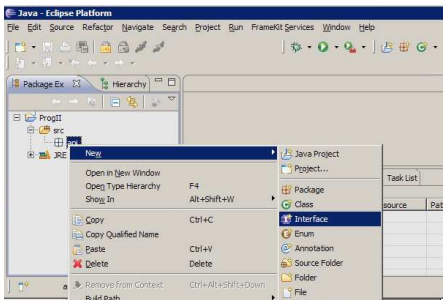


Figura 3.4: Nueva interface

ejemplo "PilaTDA") y presionar **Finish**.

En la **ventana central** se incluye el editor de Java que contiene el código pregenerado del archivo fuente para la clase o interface. En la **ventana Outline** a la derecha se muestra la definición del archivo actual con los métodos y atributos que contiene (si los hubiera). De este modo, en un programa extenso, podemos acceder a un método específico haciendo clic en su nombre en esta ventana.
A continuación se observa el código pregenerado de la nueva interface:

```
package api;

public interface PilaTDA {

}
```

Una vez finalizada la codificación del archivo fuente se deberán salvar el archivo, haciendo clic en el ícono **Save**. Con esta acción automáticamente se "compilará" todo el código generado hasta el momento y se marcarán los errores en caso de que los hubiera en el cuadro **Problems** ubicado en la parte inferior de la pantalla.

## 3.4 Creación de una clase

Para crear una nueva clase, parado sobre el nombre del paquete en el cual queremos que resida, se deberá hacer clic con el botón derecho y seleccionar: **New** → **Class**.
En el recuadro que aparece se deberá escribir el nombre de la clase a crear (por ejemplo "Nodo") y presionar **Finish**.

### 3.4.1 Implementación de Interfaces

Si quisiéramos escribir una clase que implementa una interface previamente definida (es decir, realizar una implementación concreta de un TDA definido), es muy útil, en la misma pantalla donde se escribe el nombre de la clase indicar la interface que estaremos implementando.
Entonces, por ejemplo, creamos el paquete "Impl", dentro de este creamos una clase denominada "PilaArr" que implementará la interface "PilaTDA" (Fig. 3.5). Para

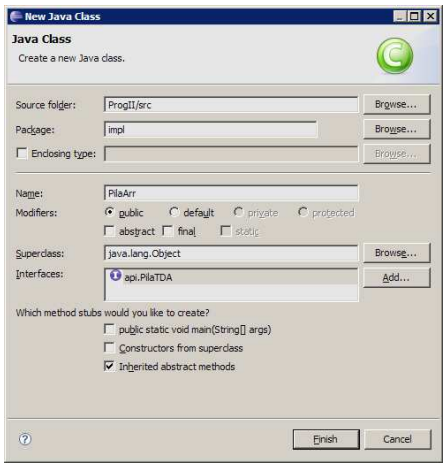


Figura 3.5: Definición de nueva clase

indicar la interface a implementar se presiona el botón **Add** que se encuentra al lado del cuadro **Interfaces** (Fig. 3.6).

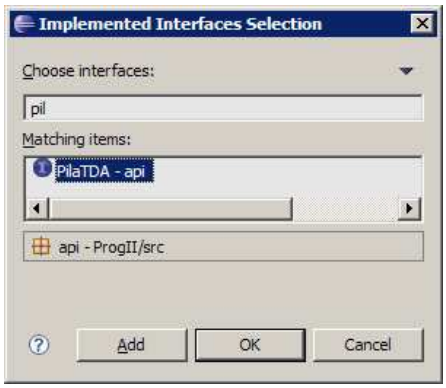


Figura 3.6: Selección de la Interface que implementa

En la Fig. 3.5, la opción **inherited abstract methods** (que en forma predeterminada esta seleccionada) hace que Eclipse complete automáticamente el esqueleto de los métodos definidos en la interface para los cuales esta clase deberá proveer una implementación. A continuación se observa el código pregenerado de la nueva clase:

```
package impl;

import api.PilaTDA;

public class PilaArr implements PilaTDA {

    public void Apilar(int x) {
        // TODO Auto-generated method stub
    }
    public void Desapilar() {
        // TODO Auto-generated method stub
    }
    ...
```

### 3.4.2 Programa Principal

Para crear una clase que se comporte como programa principal, podemos seleccionar la opción **public static void main (String [] args)** (Fig. 3.5) con lo que se creará automáticamente dicho método dentro del cuerpo de la nueva clase definida.
Entonces, por ejemplo, creamos el paquete "ejercicios", dentro de este creamos una clase denominada "TP1".

```
package ejercicios;

public class TP1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

## 3.5 Resolviendo un ejercicio

Supongamos que ahora queremos realizar un ejercicio que consiste en "Pasar el contenido de una Pila a otra Pila".

Vamos a asumir que tenemos:

- en el paquete **api**, la definición del **PilaTDA** (ver 3.3),
- en el paquete **impl**, una implementación de este TDA denominada **PilaArr** (ver 3.4.1),
- y en el paquete **ejercicios**, una clase como programa principal denominada **TP1** (ver 3.4.2).

Lo primero que vamos a hacer es, en la clase **TP1**, declarar un método que va a resolver el ejercicio propuesto. Es importante que este método tenga el modificador **static** para que no sea necesario instanciar (con un new) la clase que contiene al método para invocarlo, por ejemplo:

```
public static void PasarPilaAPila(PilaTDA origen, PilaTDA destino) {

}
```

Posteriormente, sobre el margen izquierdo de esta declaración se visualizará una marca de error en rojo (ver Fig. 3.7). Al poner el puntero sobre la marca podrá leerse "PilaTDA cannot be resolved to a type", lo cual ocurre porque la definición PilaTDA se encuentra en otro paquete por lo que no es visible en esta clase.

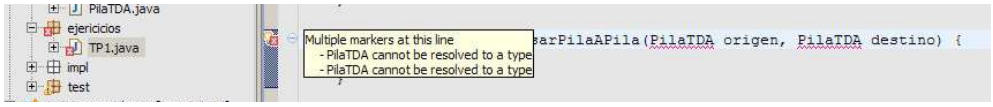


Figura 3.7: Visualización de errores

Para solucionarlo hay que declarar la **importación de paquete** "api" (de la misma forma que aparece en el código de la clase PilaArr).
Haciendo clic con el botón derecho en la marca de error, se accede a la opción **Quick Fix** de Eclipse que permite resolver rápidamente, entre otros, este tipo de problemas (ver la Fig. 3.8).

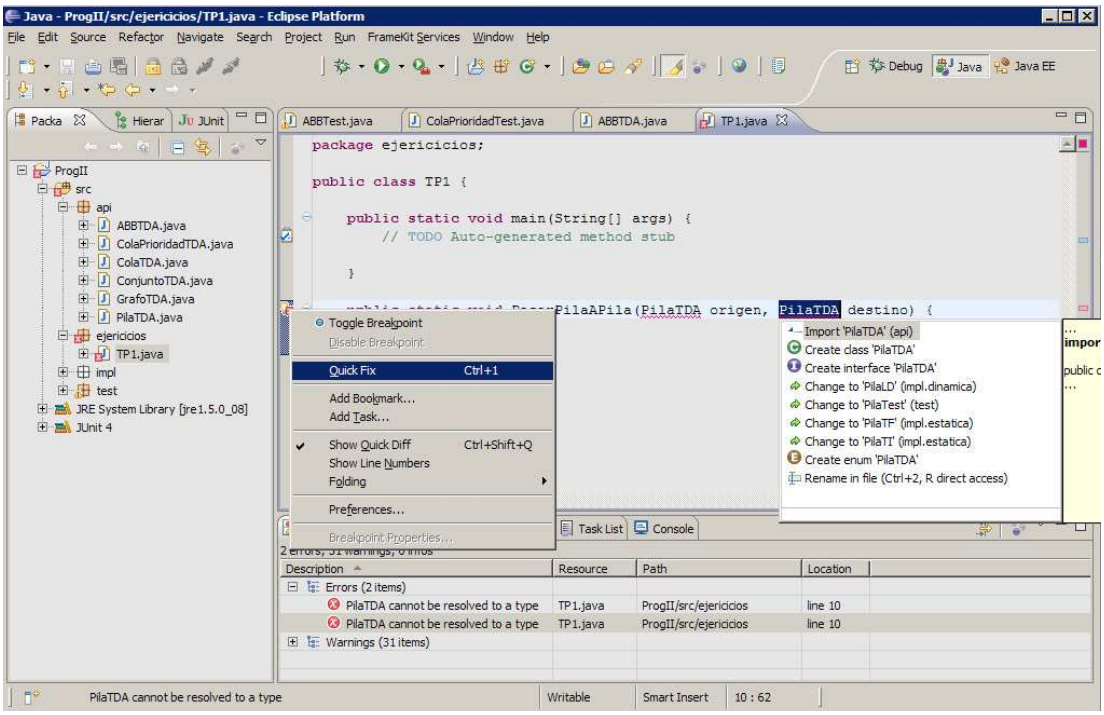


Figura 3.8: Quick Fix para resolver la importación de paquetes

Luego de definir el contenido del método "PasarPilaAPila", en el programa principal lo podemos invocar de la siguiente forma:

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

    PilaTDA p1= new PilaArr();
    p1.InicializarPila();
    PilaTDA p2= new PilaArr();
    p2.InicializarPila();

    p1.Apilar(1);
    p1.Apilar(2);
    p1.Apilar(3);

    PasarPilaAPila(p1, p2);
}
```

En caso que hubiésemos decidido definir el método "PasarPilaAPila" en otra clase, por ejemplo en la clase MetodosPila, habrá que cambiar la última instrucción por

```
    MetodosPila.PasarPilaAPila(p1, p2);
```

## 3.6 Cerrando Eclipse

Para salir de eclipse sólo hay que cerrar el espacio de trabajo o por medio de: **File** → **Exit**. Lo que ocasiona que Eclipse guarde el último estado salvado en el Workbench y cierre la aplicación.

# Unidad 4

# Ejecutando con Eclipse

## 4.1 Compilación

Los archivos fuentes de Java (interfaces y clases) tienen extensión **.java**. Cada vez que se salva uno de estos archivos, automáticamente se "compilará" todo el código generado hasta el momento generando los fuentes compilados que tienen extensión **.class**. En caso de detectarse algún error como resultado del proceso de compilación, estos se visualizarán en el cuadro "problems" ubicado en la parte inferior de la pantalla.

## 4.2 Ejecución de un programa principal

Para ejecutar un programa principal por primera vez, abrimos la clase que tiene definido el método **main()**, y seleccionamos la opción de menú: **Run** → **Run** (ver Fig. 4.1). Otra opción es en, en el Package Explorer, ubicar la clase del programa y haciendo clic con el botón derecho, seleccionar la opción: **Run As...** → **Java application**.


Figura 4.1: Ejecución del Programa Principal

Para la primera ejecución de un programa se crea automáticamente una nueva configuración de ejecución con el nombre de la clase asociada al programa, de manera que para correr nuevamente el programa basta luego con hacer clic en el ícono **Run** de la barra de herramientas (es la flecha blanca en un círculo verde). Expandiendo la Flecha de Run, se pueden observar todas las ejecuciones recientes. El ícono se asocia con la configuración de ejecución más reciente del programa, por lo tanto para volver a ejecutar el programa sólo hay que hacer clic en este ícono.

La **ventana de consola** (ver Fig. 4.2), situada en la parte inferior de la pantalla: se abre automáticamente cuando un programa escribe en **System.out** o en **System.err**, por cuanto:

- En ella se muestra el mensaje de salida esperado

- Las alertas a observar en el código del programa
- O el mensaje de error si lo hubiere.
- Adicionalmente cuando un programa está en ejecución se puede finalizar el mismo haciendo clic en la opción "Terminate" de esta ventana.

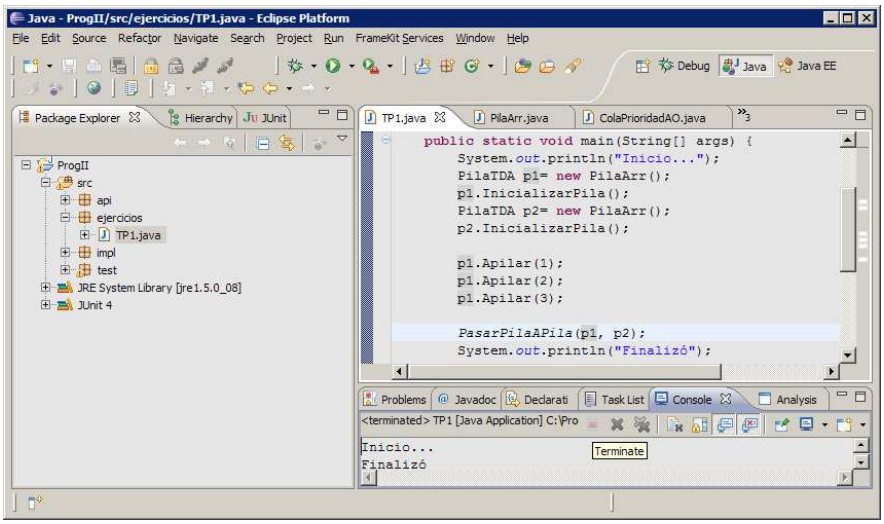


Figura 4.2: Consola de ejecución

## 4.3 Usando el Debugger

Eclipse permite hacer depuración de código de una manera muy simple. Primero se deben establecer aquellos **puntos de interrupción** (breakpoints) en el código fuente donde queremos detener la ejecución temporariamente permitiéndonos inspeccionar los valores de las variables . Entonces, para establecer un punto de interrupción en una instrucción, en el margen izquierdo de la misma hacemos clic derecho para indicar la opción "Toggle Breakpoint" (ver Fig. 4.3).

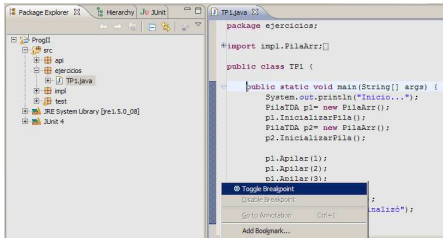

Figura 4.3: Estableciendo un punto de Interrupción

Posteriormente, hay que ejecutar en modo debug un programa principal, para lo cual abrimos la clase que tiene definido el método **main()**, y seleccionamos la opción de menú: **Run** → **Debug**. A continuación, si la ejecución del programa principal alcanza

una instrucción que tiene un punto de interrupción, entonces se activa la **Perspectiva de Debug** como se visualiza en la Fig. 4.4).

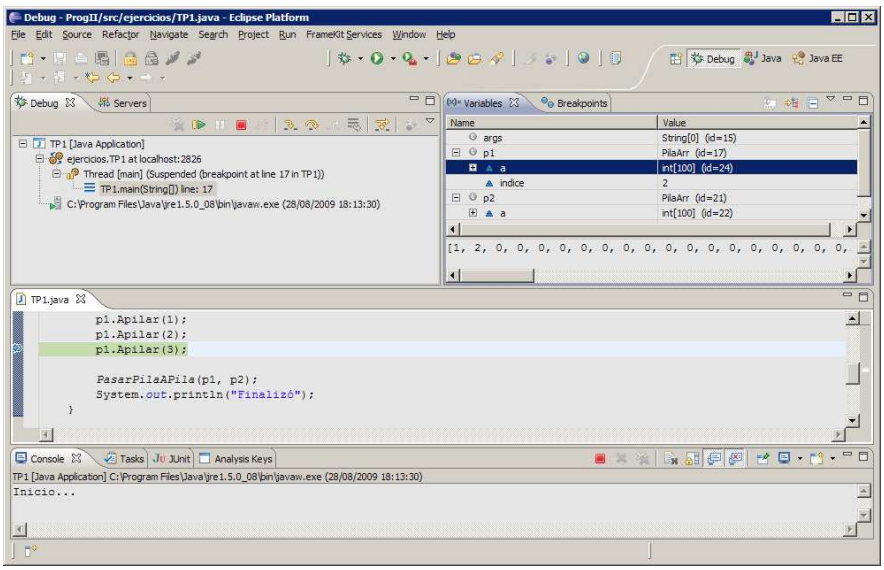


Figura 4.4: Perspectiva de Debug

En la perspectiva Debug aparecen las siguientes siguientes ventanas:

- En la parte superior izquierda la **pila de llamadas** (call stack).
- La **consola** que nos indica si la ejecución aún esta activa (observar el cuadrado en rojo).
- En la parte superior derecha las **variables**, muestra los tipos y valores que tienen actualmente las variables cuando la ejecución se detiene en un punto de interrupción. Por ejemplo, en la Fig. 4.4, se observa que la variable "p1" referencia a un objeto de tipo PilaArr, a su vez, este objeto contiene dos variables "indice" y "a". La variable "indice" contiene el valor 2 y la variable "a" es un arreglo de enteros de 100 posiciones, donde observamos en la parte inferior de esta ventana, que contiene los valores 1, 2, 0, 0, etc.
- En la parte superior, la **barra de herramientas de Debug** donde se encuentran las operaciones: Resume, Step Over y Step Into que nos permiten continuar la ejecución del programa con distintas variantes.

## 4.4 Casos de prueba con JUnit

JUnit es un conjunto de clases que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que

el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

### 4.4.1 Asociando la librería JUnit

La librería JUnit viene con la distribución de Eclipse, pero para usarla hay que incluirla en el proyecto. Para ello seleccionamos la opción de menú: **Project** → **Properties** y en esta pantalla seleccionamos la opción **Java Buid Path** y aquí la pestaña **Libraries** (Fig. 4.5). Posteriormente, con la opción **add Library**, seleccionamos JUnit (Fig. 4.6), y al hacer **Next** la opción "JUnit 4". La librería añadida será visible en la ventana Package Explorer.

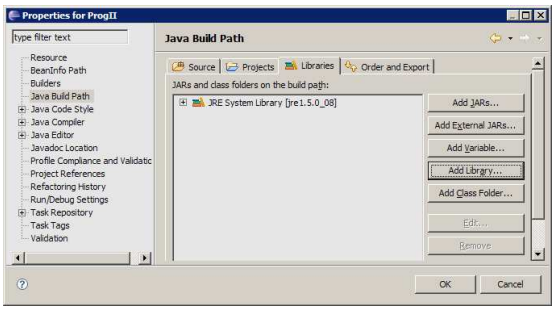


Figura 4.5: Librerías asociadas al proyecto

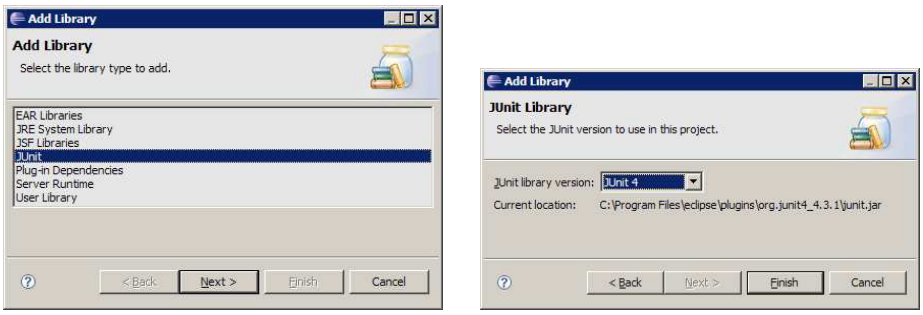


Figura 4.6: Asociando la librería JUnit

### 4.4.2 Escribiendo las pruebas

A modo de ejemplo, supongamos que queremos armar pruebas para el método "PasarPilaAPila" (ver 3.5). Primero abrimos la clase "TP1", y aquí seleccionamos la opción de menu: **File** → **New JUnit Test Case**. En la pantalla que aparece, similar a la primera de la Fig. 4.7, se elige **Next**, en esta segunda pantalla seleccionamos aquellos métodos para los cuales queremos definir casos de prueba.

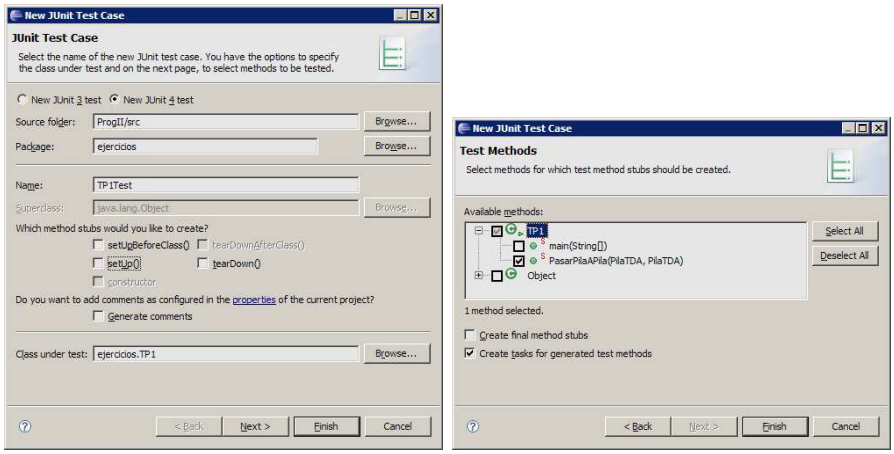


Figura 4.7: Definición de nuevos casos de prueba JUnit

A continuación se observa el código pregenerado para la nueva clase de "TP1Test":

```
package ejercicios;

import static org.junit.Assert.*;
import impl.PilaArr;
import org.junit.Test;
import api.PilaTDA;

public class TP1Test {
    @Test
    public void testPasarPilaAPila() {
        fail("Not yet implemented"); // TODO
    }
}
```

En este código hay que completar los casos de prueba, por ejemplo de la siguiente forma:

```
    @Test
    public void testPasarPilaAPila() {
        PilaTDA p1= new PilaArr();
        p1.InicializarPila();
        PilaTDA p2= new PilaArr();
        p2.InicializarPila();

        p1.Apilar(1);
        p1.Apilar(2);
        p1.Apilar(3);
        //p1 tiene en el tope 3, luego 2, luego 1.
        TP1.PasarPilaAPila(p1, p2);
        //p2 deberia tener en el tope 1, luego 2, luego 3.

        assertEquals("Se espera Tope: ", p2.Tope(), 1);
        p2.Desapilar();
        assertEquals("Se espera Tope: ", p2.Tope(), 2);
```

```
        p2.Desapilar();
        assertEquals("Se␣espera␣Tope:␣", p2.Tope(), 3);
        p2.Desapilar();
        assertTrue("La␣pila␣debe␣quedar␣vacía", p2.PilaVacia());
    }
```

Observar que se utiliza el método "assertEquals" y "assertTrue" para establecer aquellas condiciones que suponemos que deben satisfacerse para cumplir con el caso de prueba.

### 4.4.3 Ejecución de pruebas

Para ejecutar los casos de pruebas definidos en "TP1Test", seleccionamos la opción de menu: **Run → Run As... → JUnit Test** (ver Fig. 4.8).

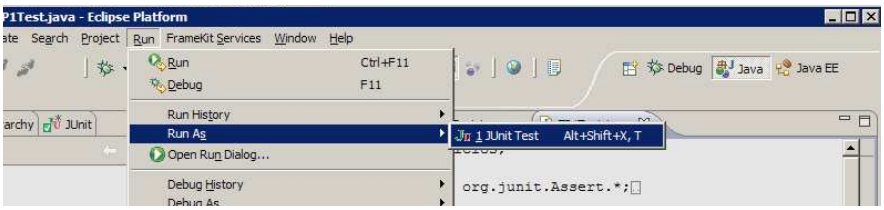


Figura 4.8: Invocación para ejecutar un JUnit

El resultado de la resultado de la ejecución se presenta en la parte izquierda de la pantalla, en la **ventana JUnit**, como aparece en la Fig. 4.9. En este ejemplo todos los casos de pruebas se verificaron exitosamente, en caso contrario, en la parte inferior izquierda de esta pantalla se visualizarán aquellos situaciones de error.

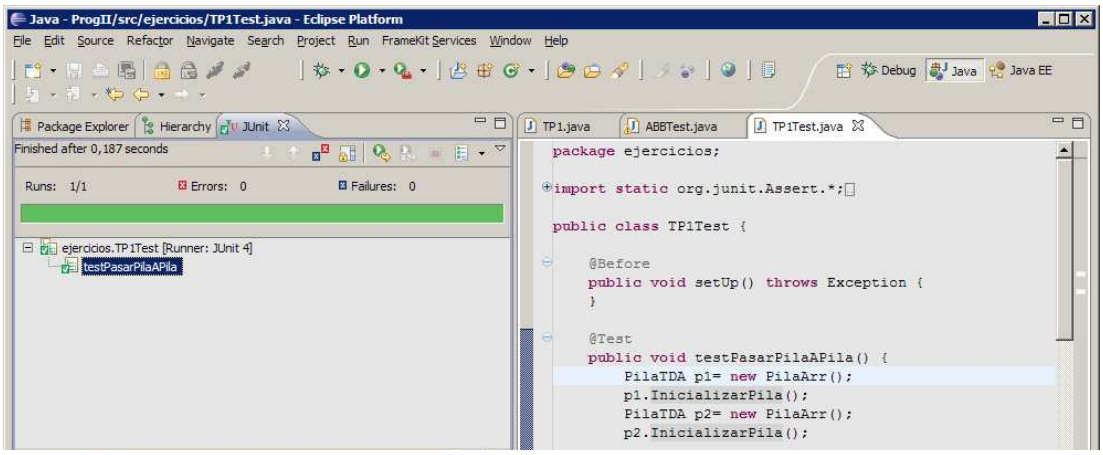


Figura 4.9: Resultado de Ejecución del JUnit

# Unidad 5

# Adicionales

## 5.1 Exportación de Proyecto

Para exportar los archivos de un proyecto se debe seleccionar la opción de menu: **File** → **Export**. A continuación, aparece la primer pantalla de la Fig. 5.1 donde seleccionamos "Archive File". Al confirmar esta opción aparece la segunda pantalla donde debemos determinar los archivos a Exportar, para lo cual marcamos el proyecto raíz y definir el nombre del archivo destino.

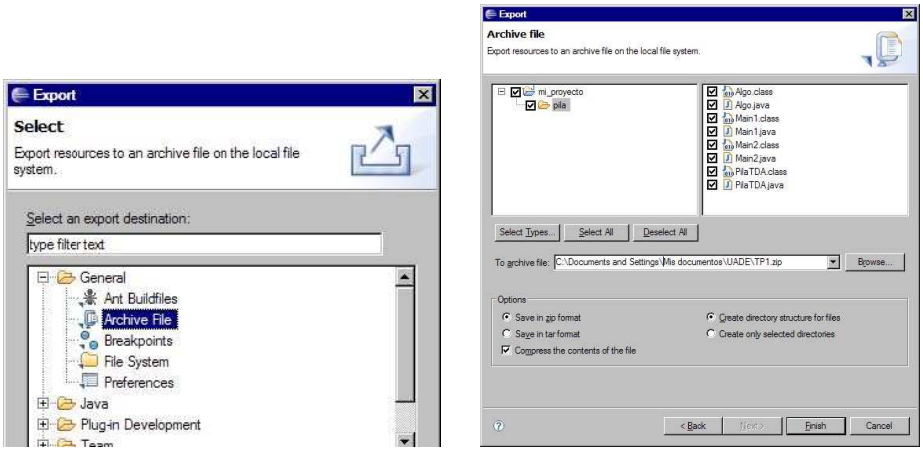


Figura 5.1: Opciones de Exportación

## 5.2 Importación de Proyecto

Para restaurar un proyecto que fue exportado por el mecanismo descripto en la sección anterior primero se descomprime el archivo en una nueva subcarpeta del workspace (ver 2.2). Posteriormente, se crea un nuevo proyecto siguiendo los pasos de la sección 3.1 pero marcando la opción: **Create project from existing source** para la cual indicaremos la carpeta del workspace que acabamos de crear.

## 5.3 Asociando Librerías .JAR

En Java las librería son archivos con extensión .jar. Básicamente, cada archivo .jar es un conjunto de clases e interfaces Java comprimidas con un compresor .ZIP al que se le cambia la extensión.
Para asociar una librería al proyecto se deben seguir los siguientes pasos:

1. copiar esta librería a la carpeta del proyecto que reside en el workspace (ver 2.2).
2. en la ventana del Package Explorer seleccionamos el proyecto y ejecutamos la acción **File** → **Refresh**.
3. seleccionamos la opción de menú: **Project** → **Properties** y en esta pantalla seleccionamos la opción **Java Buid Path** y aquí la pestaña **Libraries** (Fig. 4.5). Finalmente, con la opción **add JARs..**, seleccionamos la nueva librería.

Luego de realizada esta acción, en el Package Explorer, dentro de la carpeta "Referenced Libraries" se observará la nueva librería.