

Seminario 2

Eclipse

PROGRAMACIÓN 3

David Rizo, Pedro J. Ponce de León

Departamento de Lenguajes y Sistemas Informáticos

Universidad de Alicante



Seminario 2.1

Contenidos

Índice

1. Instalación	1
2. Entorno	1
2.1. Workspace	1
2.2. Interfaz	2
3. Proyectos	2
3.1. Creación	2
4. Clases	2
4.1. Importación clases	2
4.2. Creación de clases	2
5. Ejecución	3
5.1. Depuración	3
6. Pruebas unitarias	3
7. Generación código	4

Seminario 2.2

1. Instalación

Instalación

- Localizado en www.eclipse.org
- Descargar *Eclipse IDE for Java Developers*
- Descomprimir y arrancar el ejecutable `eclipse`

Seminario 2.3

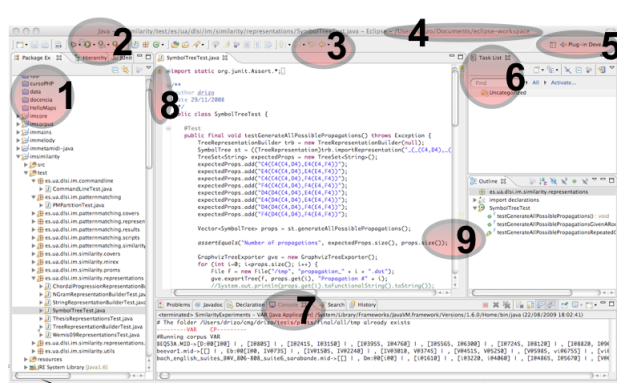
2. Entorno

2.1. Workspace

Workspace

- Eclipse guarda toda su configuración en un directorio que denomina *workspace*
- Cuando iniciamos el entorno debemos decir dónde guardar el *workspace*
- Podemos cambiar de *workspace* cuando queramos pulsando en `File>Switch workspace`

Seminario 2.4



2.2. Interfaz

Interfaz

Herramientas y ayudas visuales

- | | | |
|----------------------------|----------------------|---|
| 1. Proyectos y paquetes | 5. Perspectiva | 8. Breakpoints, enlace para solución de errores |
| 2. Ejecución y depuración | 6. Una vista: tareas | |
| 3. Navegación por ficheros | 7. Consola | 9. Errores, warnings, TO-DO |
| 4. Workspace activo | | |

Seminario 2.5

3. Proyectos

3.1. Creación

Creación de proyectos

- File > New > Java project
 - Nombre del proyecto
 - contents*: seleccionar directorio nuestro o dejar el del workspace
- Esto crea un directorio que contiene por defecto:
 - bin, src
 - Los ficheros ocultos *.project* y *.classpath*
 - Estos ficheros contienen los metadatos del proyecto
 - Cuando queramos llevarnos a otro ordenador un proyecto *eclipse* los usará para identificar un directorio como contenedor de un proyecto
 - La importación se puede realizar pulsando File > Import > General > Existing Projects into Workspace y seleccionando el directorio del proyecto

Seminario 2.6

4. Clases

4.1. Importación clases

Importación de Clases

Podemos importar ficheros *.java* de clases escritas fuera de *eclipse* simplemente copiando los ficheros en el navegador de ficheros del sistema operativo y pegándolos en la vista de paquetes.

Actividad

Añadir los ficheros de la práctica 1 al directorio *src* del proyecto que acabamos de crear.

Seminario 2.7

4.2. Creación de clases

Clases

- Creación con File > New > Class
- Especificamos nombre, paquete, y opcionalmente si queremos que nos añada un *main*

Actividad

- Crear una clase denominada *Gameboard* en el paquete `model` y añade los atributos `int height` e `int width`. Escribiendo sobre ellos `/**` y pulsando *enter* nos ayudará a crear la documentación *javadoc*.
- Crear el constructor `public Gameboard(Coordinate c)` y añade de la misma forma la documentación
- Si tenemos algún error usaremos las ayudas que aparecen en la barra izquierda del editor de código

Seminario 2.8

5. Ejecución

Ejecución

- Dado que un proyecto puede tener varios ficheros con un método `main` lo más sencillo para ejecutar es pulsar con el botón derecho sobre la clase que contiene el `main` a ejecutar y pulsar en `Run as > Java application`.
- Esto crea una configuración de ejecución (menú `Run > Run configurations`), donde podemos añadir parámetros adicionales a la ejecución

Actividad

En línea de comandos esto sería equivalente a:

- Abrir un terminal
- Situar en el directorio del proyecto.
- Ejecutar `java -cp bin mains.Main1C1415` (*Eclipse* automáticamente compila las clases y las deja en `bin`).

Seminario 2.9

5.1. Depuración

Depuración

- Pulsando en el menú `Run > Debug` (también en la barra de herramientas) se arranca la depuración de nuestra aplicación.
- Si queremos evaluar un elemento concreto en un punto determinado debemos fijar un *breakpoint*
- Al arrancar la depuración se cambia la *perspectiva* de *Eclipse* a *Debug*.

Actividad

Ejecutar línea a línea `mains.Main1C1415`

Seminario 2.10

6. Pruebas unitarias

Pruebas unitarias

- Una **prueba unitaria** es un fragmento de código que verifican un caso concreto de uso de un componente software según las especificaciones.
- Cada prueba se configura para probar un caso determinado de uso de la interfaz de una clase.
- Las pruebas se organizan en conjuntos o **suites** de pruebas. Cada 'suite' se asocia a una clase.
- Se prueban, por ejemplo, condiciones o valores límite en argumentos de métodos, o condiciones bajo las que un método genera excepciones.

Seminario 2.11

JUnit

- La herramienta más usada en Java para pruebas unitarias es **JUnit**.
- En *Eclipse* se configura en `Project > Properties > Java Build Path > Libraries > Add Library`

Actividad

Configura tu proyecto para que use *JUnit 4*.

Seminario 2.12

JUnit

- Separamos los ficheros de los tests unitarios del resto de código fuente
- Creamos directorio `test` en el proyecto pulsando sobre éste en la vista de paquetes y pulsando `New > Source folder`
- Creamos el directorio `model` dentro de `test` (los archivos de código que contienen las pruebas pertenecen también al paquete `model`). Pega ahí el fichero de pruebas `CoordinateTestEclipse.java`.
- Actualiza el proyecto en *Eclipse* (F5)
- La ejecución de las pruebas se realiza pulsando sobre la clase que las contiene con el botón derecho y seleccionando `Run as > JUnit test`.

Seminario 2.13

JUnit

Actividad

- Abrir el test unitario *CoordinateTestEclipse*
 - Los métodos con anotaciones `@Before` configuran el test. Se ejecutan antes de cada método `@Test`.
 - Los métodos `@Test` contienen una o más pruebas unitarias.
 - `assertEquals` comprueba que el valor esperado coincide con el real. Los parámetros son por este orden: título (opcional), valor esperado, valor real, diferencia en valor absoluto permitida (opcional, útil para los reales) .
 - `assertTrue`, `assertFalse` comprueban que su argumento devuelve `true` o `false`, respectivamente
- Probar una ejecución sin errores y otra en la que no se cumpla alguna aserción (modificando algún valor esperado del test para provocar el error). Veremos cómo detectar el problema pulsando en el panel `Failure trace`.

Seminario 2.14

Nuevo test unitario

Para generar un nuevo test unitario sobre una clase, pulsar con el botón derecho sobre ésta en la vista de paquetes, y seleccionar `New > JUnit test case`.

- Seleccionar `JUnit 4`
- En el directorio, seleccionar `test` en lugar de `src`

Actividad

- Implementa los métodos `getHeight()` y `getWidth()` de `Gameboard`
- Crear un test unitario nuevo para `Gameboard` que compruebe el constructor
- Para ejecutar todos los tests podemos pulsar con botón derecho sobre el proyecto y seleccionar `Run as > JUnit test`

Seminario 2.15

7. Generación código

Generación de código

- La implementación de algunas operaciones como `equals` o `toString` suele ser rutinaria
- *Eclipse* nos ayuda a realizarlo pulsando con el botón derecho en el código de la clase y seleccionando `Source > Generate toString()` y `Source > Generate hashCode and equals()`. Esto generará un código base que luego será fácil modificar.

Actividad

Generar estos métodos para `Gameboard` (`equals()` no lo necesitaremos en la práctica 2)

Seminario 2.16