Seminario 2

Eclipse y Junit

PROGRAMACION 3

David Rizo, Pedro J. Ponce de León
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Contenidos

Instalación

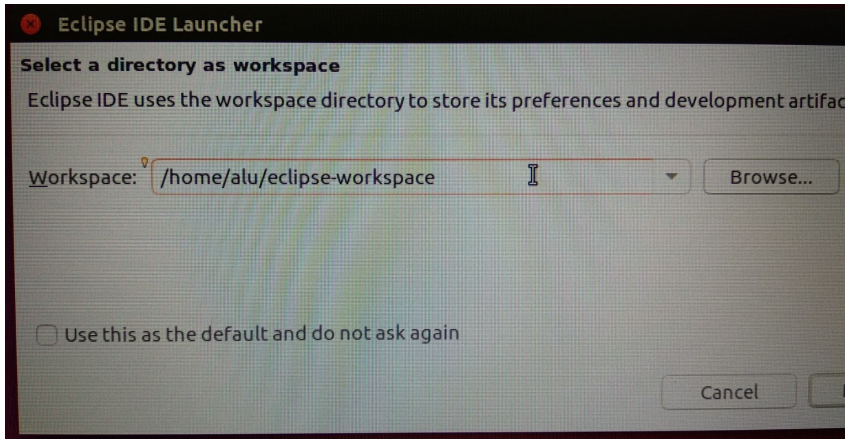
En *Programación 3* usaremos la versión 2022-06 (aunque las versiones posteriores deben funcionar bien)

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

David Rizo, Pedro J. Ponce de León

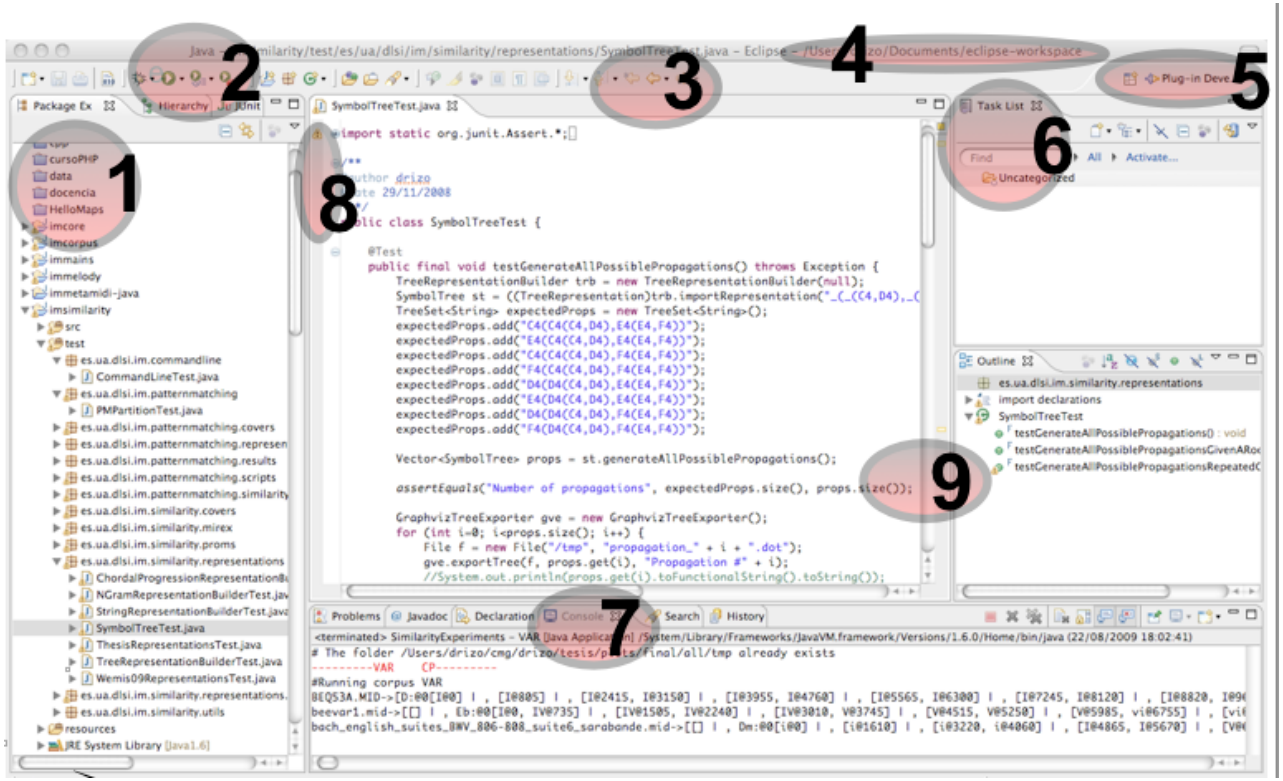
Workspace

- Eclipse guarda toda su configuración en un directorio que denomina *workspace*
- Cuando iniciamos el entorno debemos decir dónde guardar el *workspace*



- Indica un directorio en tu carpeta de usuario (en el laboratorio). Eclipse lo creará si no existe.
- Podemos cambiar de *workspace* cuando queramos pulsando en `File>Switch workspace`

Interfaz



Herramientas y ayudas visuales

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

Creación de proyectos

- `File > New > Java project`
  - Nombre del proyecto
- Esto crea un directorio que contiene por defecto subdirectorios:
  - directorio `src/` : contiene el código fuente.
  - directorio `bin/` : contiene el código compilado.
  - Los ficheros ocultos `.project` y `.classpath`
    - Estos ficheros contienen los metadatos del proyecto, entre ellos, la versión del JDK y el *classpath*, que apunta al directorio *bin/*.

David Rizo, Pedro J. Ponce de León

Importación de proyectos

La importación se realiza pulsando `File > Import > General > Existing Projects into Workspace` y seleccionando `Select root directory:` o `Select archive file:` dependiendo de si el proyecto a importar está en una carpeta o en un archivo comprimido.

Actividad

Descarga el proyecto de Eclipse preconfigurado que encontrarás en la web de la aisgnatura e impórtalo a Eclipse. Esto debe crear un proyecto Eclipse sin código llamado **prog3-base** con dos carpetas de código fuente: **src**, para tu código y **test** para el código de pruebas.
El proyecto está configurado para

- Utilizar el JDK 1.8
- Utilizar la codificación de caracteres UTF-8 al crear archivos de código fuente.
- Utilizar saltos de línea estilo Unix (carácter `'\n'` para finales de línea) en los ficheros de código fuente.

Importación de Clases

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

Actividad

- Añade el paquete `es.ua.dlsi.prog3.p1` a la carpeta `src`:
  - Botón derecho sobre la carpeta, después `New... -> Package.`
- Añade (si lo tienes) el fichero de código fuente `Coordinate.java` de la práctica 1 al paquete que acabas de crear. Si no lo tienes, pasa a la siguiente página.

David Rizo, Pedro J. Ponce de León

Clases

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

Actividad

- Si aún no la tienes, crea la clase *Coordinate* en el paquete `es.ua.dlsi.prog3.p1` y añade el atributo privado `double[] components`. Escribiendo encima de éste `/**` y pulsando *enter* nos ayudará a crear la documentación *javadoc*.
- Implementa uno de los constructores de la clase según el enunciado de la práctica 1. Añade de la misma forma la documentación.
- Si tenemos algún error usaremos las ayudas que aparecen en la barra izquierda del editor de código.

Ejecución desde Eclipse

- 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

- Añade un método `main` a tu clase `Coordinate`. Déjalo vacío: `public static void main(String[] args) { }`
- Ejecútalo como se indica arriba.

Ejecución desde terminal

Actividad

En línea de comandos lo anterior sería equivalente a:

- Abrir un terminal
- Situarse en el directorio del proyecto.
- Ejecutar `java -cp bin es.ua.dlsi.prog3.p1.Coordinate` (*Eclipse* automáticamente compila las clases y las deja en el directorio `bin` del proyecto).

David Rizo, Pedro J. Ponce de León

Depuración

- Pulsando en el menú `Run > Debug` (o en el icono del insecto 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* o *punto de parada*, haciendo doble-click en la barra situada a la izquierda de la línea de código donde queremos que la ejecución se detenga.
- Al arrancar la depuración se cambia la *perspectiva* de *Eclipse* a *Debug*.

Ayuda

Step into (F5)	Ejecutar paso a paso entrando en cada método.
Step over (F6)	Ejecutar la siguiente línea completa en un solo paso.
Step return (F7)	Ejecutar el resto del método actual y retornar al punto de llamada.
Resume (F8)	Continuar la ejecución hasta el siguiente breakpoint (o fin de la aplicación).
Run to line (^R)	Continuar la ejecución hasta la línea donde está situado el cursor.

Depuración

Actividad

1. Añade este código a tu `main`:

```
double[] d1 = new double[] { 2.5, 3.4 };
double[] d2 = new double[] { 2.5, 3.4, -3.2 };
Coordinate c1 = new Coordinate(d1);
Coordinate c2 = new Coordinate(d2);
System.out.println(c1.getDimensions());
System.out.println(c2.getDimensions());
```

2. Pon un *breakpoint* en la primera línea de código de la función `main` y
3. ejecútalo línea a línea.

Generación de código

- La implementación de algunas operaciones como `equals`, `hashcode` 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()` o `Source > Generate hashCode and equals()`. Esto generará un código base que luego será fácil modificar.

Actividad

Prueba a generar los métodos `hashCode` y `equals` para `Coordinate`.

¡Anteción!

Los métodos generados de esta manera no siempre hacen lo que queremos. Por ejemplo, `toString()` podría crear una cadena con un formato distinto, o `equals()` podría estar comparando los objetos de una forma distinta a como deseamos hacerlo.

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.

David Rizo, Pedro J. Ponce de León

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`
- Usaremos la versión *JUnit 4*. Ya está incluída en el proyecto base que descargaste.

David Rizo, Pedro J. Ponce de León

JUnit

Separamos los ficheros de los tests unitarios del resto de código fuente.

Actividad

- Descomprime el archivo que contiene las pruebas de la práctica 1 `tests_p1.tgz`. Copia y pega la carpeta `es` dentro de la carpeta `test` del proyecto (los archivos de código que contienen las pruebas pertenecen también al paquete `es.ua.dlsi.prog3.p1`).
- Actualiza el proyecto en Eclipse (F5)

La ejecución de las pruebas se realiza pulsando sobre el paquete o la clase que las contiene con el botón derecho y seleccionando `Run as > JUnit test`.

David Rizo, Pedro J. Ponce de León

JUnit

Actividad

- Abre el test unitario *CoordinateTest.java*
- 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 números reales) .
- `assertTrue, assertFalse` comprueban que su argumento devuelve `true` o `false`, respectivamente.
- `fail` produce un fallo en el test cuando es ejecutado.

David Rizo, Pedro J. Ponce de León

JUnit

Actividad

- Ejecuta los test: `Run / Debug as... -> JUnit Test` sobre el archivo de los test (los que contienen instrucciones `fail` fallarán). Se abre la pestaña `JUnit` donde podemos ver el resultado de la ejecución.
- Selecciona un test que falle. En el panel `Failure trace` haz doble click sobre la primera línea que indique `at es.ua.dlsi.prog3.p1.CoordinateTest ...`. Te llevará a la línea que produjo el error.
- Modifica algún valor esperado en un test que funciona. Ahora fallará y seleccionando el test en el panel `Failure trace` podrás ver la causa en la primera línea.

David Rizo, Pedro J. Ponce de León

Nuevo test unitario

Para generar un nuevo test unitario sobre una clase, debemos pulsar con el botón derecho sobre ésta en la vista de paquetes, y

- seleccionar `New > JUnit test case`.
- Seleccionar JUnit 4.
- En `Source folder`, selecciona `test` en lugar de `src`.

Actividad

- Implementa un método en `Coordinate` que devuelva la suma de las componentes.
- Crea un test unitario (o varios) para comprobar que el método funciona correctamente.
- Para ejecutar todos los tests podemos pulsar con botón derecho sobre el proyecto y seleccionar `Run as > JUnit test`.
- También puedes ejecutar alguno en concreto (o sólo los que han fallado)
- Elimina el método y sus tests cuando hayas terminado con esta actividad.

David Rizo, Pedro J. Ponce de León