

Seminario 2

Eclipse y Junit

PROGRAMACION 3

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



[Eclipse y Junit](#)

David Rizo, Pedro J.
Ponce de León



[Contenidos](#)

[Instalación](#)

[Entorno](#)

Workspace

Interfaz

[Proyectos](#)

Creación

[Clases](#)

Importación clases

Creación de clases

[Ejecución](#)

Depuración

[Pruebas unitarias con JUnit](#)

[Generación de código](#)

Contenidos

1 Instalación

2 Entorno

Workspace
Interfaz

3 Proyectos

Creación

4 Clases

Importación clases
Creación de clases

5 Ejecución

Depuración

6 Pruebas unitarias con JUnit

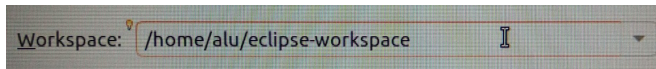
7 Generación de código

- Localizado en www.eclipse.org
- Descargar *Eclipse IDE for Java Developers 2019-06*
- Descomprimir y arrancar el ejecutable `eclipse`

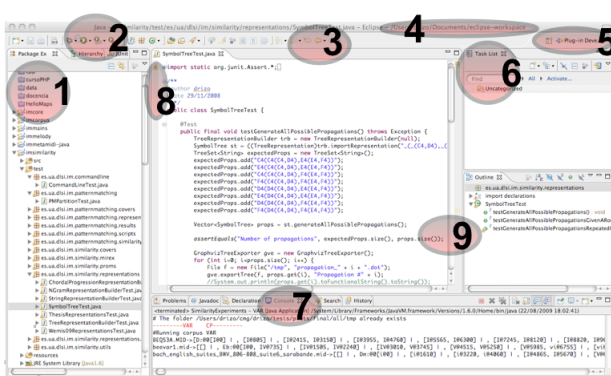




- Eclipse guarda toda su configuración en un directorio que denomina *workspace*
- Cuando iniciamos el entorno debemos decir dónde guardar el *workspace*. Selecciona el que Eclipse sugiere por defecto:



- Podemos cambiar de *workspace* cuando queramos pulsando en `File>Switch workspace`



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



- `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

Importación de proyectos

La importación se puede realizar 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 el enunciado de la primera práctica e impórtalo a Eclipse. Esto debe crear un proyecto Eclipse llamado **prog3-blockworld** que ya contiene el fichero `Main1.java` en `src/mains`.

El proyecto está configurado para

- Utilizar el JDK preconfigurado por Eclipse (JDK 1.8 en los laboratorios)
- 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.



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 importar (recuerda que `mains/Main1.java` ya está en el proyecto).



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

Actividad

- Crear una clase denominada *Player* en el paquete `model` y añade los atributos `String name` y `double health`. Escribiendo sobre ellos `/**` y pulsando *enter* nos ayudará a crear la documentación *javadoc*.
- Crear el constructor `public Player(String name)` para que asigne ese nombre al jugador y un nivel de salud inicial igual a 20. Añade de la misma forma la documentación.
- Si tenemos algún error usaremos las ayudas que aparecen en la barra izquierda del editor de código.



- 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
- Situarse en el directorio del proyecto.
- Ejecutar `java -cp bin mains.Main1` (*Eclipse* automáticamente compila las clases y las deja en `bin`).

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

- 1 Pon un *breakpoint* en la primera línea de código de la función `main` en `Main1.java` y
- 2 ejecútalo línea a línea.

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.





- 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.



- 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*.



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

Actividad

- Crea el directorio de código fuente `test` en el proyecto pulsando sobre éste en la vista de paquetes y pulsando `New > Source folder`
- Descomprime el archivo que contiene las pruebas `prog3-blockworld-pl-pretest.zip`. Copia y pega la carpeta `model` dentro de `test` (los archivos de código que contienen las pruebas pertenecen también al paquete `model`).
- 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`.



Abre el archivo de tests unitario *LocationPreTest.java*

- Fíjate en los atributos declarados. Servirán como referencias a los objetos sobre los que haremos las pruebas.
- Los métodos con anotaciones `@Before` configuran el test. Se ejecutan antes de cada método `@Test`.
- Los métodos `@Test` contienen pruebas unitarias (métodos *assert* o aserciones)
- `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
- `fail` produce un fallo del test al ejecutarse.



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 model.LocationPreTest` 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.



- En *LocationPreTest* comprobarás que hay algunos test sin implementar.
- En los comentarios de cada uno se especifica qué debe hacer ese test.
- Impleméntalos sirviéndote de los test ya implementados como ejemplo.
- ¡Hazlo bien! Esos test (y algunos más) los usaremos para la corrección de la práctica. Piensa si falta algo por probar y añade tus propios tests o amplía los que hay.

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 los métodos `getName()` y `getHealth()` de `Player`. Simplemente deben devolver el valor del atributo correspondiente.
- Crear un test unitario nuevo para `Player` que compruebe el constructor. Usa los getters para obtener el valor de los atributos y comprobar que son los esperados.
- Para ejecutar todos los tests podemos pulsar con botón derecho sobre el proyecto completo y seleccionar `Run as > JUnit test`.





- 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

Prueba a generar estos métodos para `Player` (los usaremos en la próxima práctica).