Eclipse custom templates

Por Manuel Vega Ulloa

# 1. Introducción

Las plantilas en Eclipse son pequeñas piezas de código con marcadores predefinidos que nos permiten no repetir la escritura de código. La idea es no tener que escribir un código relativamente repetitivo (como puede ser la signatura de un bucle) o no tener que recordar la sintaxis de una pieza de código más o menos compleja.

No se trata de no aprender u olvidar cómo se escribe una sentencia foreach, se trata de que:

* me la se de memoria porque llevo años escribiéndola y quiero que, con una simple combinación de teclas, el IDE (hoy toca Eclipse) la escriba para mí,
* estoy aprendiendo y quiero tener algo que me facilite su escritura, a base de verla repetidas veces, terminaré memorizando su sintaxis.

El objetivo de las plantillas es ser más productivo y, con menos esfuerzo, escribir más código.

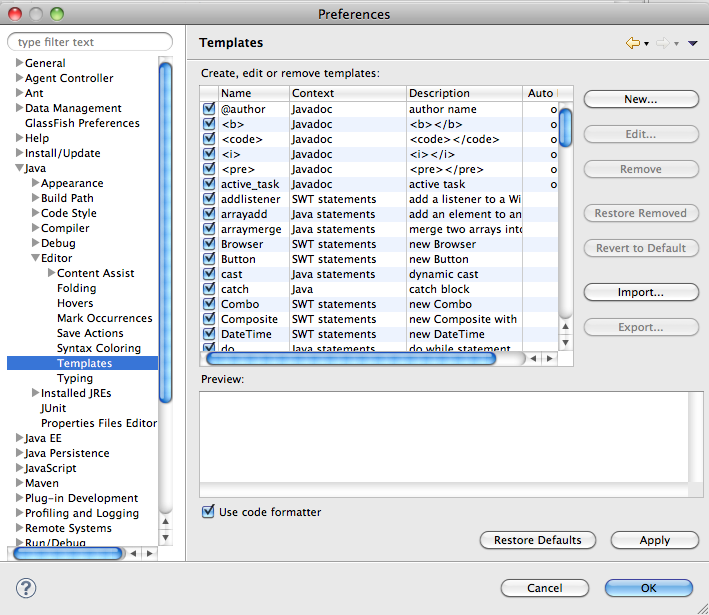
Cada plantilla tiene un nombre, basta con escribir ese nombre en el código y pulsar CTRL + espacio y el código de la plantilla se expandirá.

# 2. Plantilla predefinidas.

Eclipse viene provisto de una serie de plantillas predefinidas que podemos consultar y modificar accediendo, dentro de la opción preferencias, al lenguaje con el que trabajemos, dentro de su opción Editor > templates, así:

Preferences > XML > XML Files > Editor > Templates

Preferences > Java > Editor > Templates

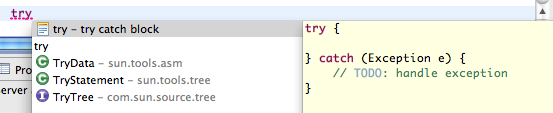


Una plantilla se compone de:

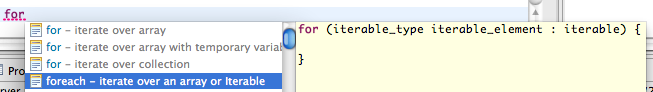
* un nombre,
* una descripción,
* un contexto en función del lenguaje (en java, si estamos en el código, en el javadoc, …) y
* un pattern, que es el  
  código de la plantilla. Dentro del código de la plantilla podemos usar texto fijo o una serie de variables predefinidas, por ejemplo:
  + {cursor}: posición en la que se establecerá el cursor de texto tras desplegar el código de la plantilla,
  + ${enclosing\_type}: tipo de la clase en la que nos encontramos, ${enclosing\_method}: nombre del método en el que nos encontramos,
  + ${todo}: TODO: task code,
  + ${year}: año en curso, ${time}: hora en curso,
  + …

Como hemos comentado, esas plantillas se mostrarán como sugerencias en el código tras comenzar a escribir su nombre y pulsar CTRL + espacio.

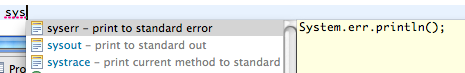
Ejemplo con try



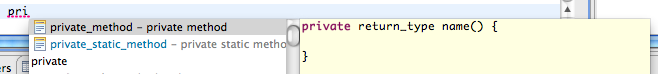
Ejemplo con for



Ejemplo con sys



Ejemplo con private

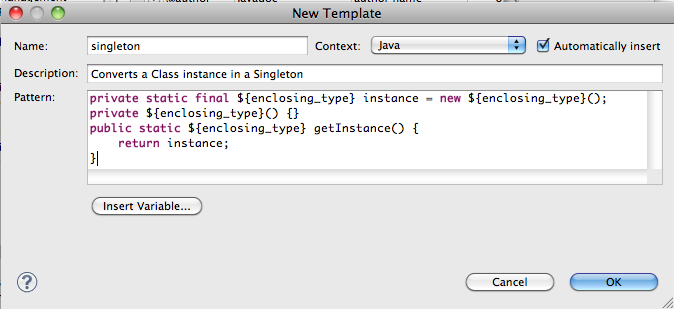


Basta con seleccionar la plantilla y pulsar INTRO para que el código de la plantilla se añada a nuestro código y si requiere de la introducción de algún tipo de texto se posicionará en aquella parte del código que lo requiera.

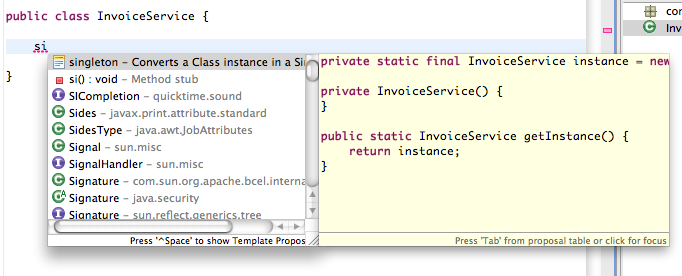
# 3. Creación de una plantilla personalizada.

Lo más interesante es que nosotros podemos crearnos nuestras propias plantillas, además de modificar las existentes. Para ello no tenemos más que añadir una nueva desde la opción de Templates, asignarle un nombre, descripción y elegir el código que queremos que se muestre al seleccionar la misma.

El siguiente es un ejemplo de un singleton:



Desde código no tenemos más que introducir las primeras letras del nombre de la plantilla y pulsar CTRL + espacio:



Las plantillas, al igual que el formateador del estilo de código se pueden importar de un fichero externo de modo que cada empresa puede tener las suyas propias, en el siguiente enlace podéis encontrar un ejemplo: http://sproutee.com/mytemplates.xml

Del mismo modo la configuración de las plantillas se almacena en el workspace, con lo que si cambias o creas un workspace nuevo se perderán. De ahí la necesidad de disponer de esos archivos xml para importarlos.

# 4. Un par de plantillas muy últiles.

A continuación, un par plantillas que pueden sernos de gran utilidad:

**singleton – Converts a Class instance in a singleton**

|  |
| --- |
| private static final ${enclosing\_type} instance = new ${enclosing\_type}();    private ${enclosing\_type}() {}    public static ${enclosing\_type} getInstance() {    return instance;  } |

La que hemos visto en el ejemplo de creación de una nueva plantilla.

**logger – create new Logger**

|  |
| --- |
| ${:import(org.apache.log4j.Logger)}  private static final Logger logger = Logger.getLogger(${enclosing\_type}.class.getName()); |

La creación de una instancia del logger que usemos es origen de un bug conocido, tiene incluso una regla de PMD, porque lo más común es no escribir el código del mismo sino copiar y pegar el código de otra clase **sin cambiar el parámetro del método getLogger()**, con lo que logamos para la clase de la que copiamos el código no la nuestra. Con esta plantilla evitamos ese error.

# 5. Conclusiones.

Cualquiera que desarrolle software necesita un medio probado para producir mejor, más rápido y más barato ([The Productive Programmer](http://oreilly.com/catalog/9780596519544" \t "_blank)). Conocer las características y facilidades de las herramientas que usamos día a día es nuestra responsabildiad.