Using code templates

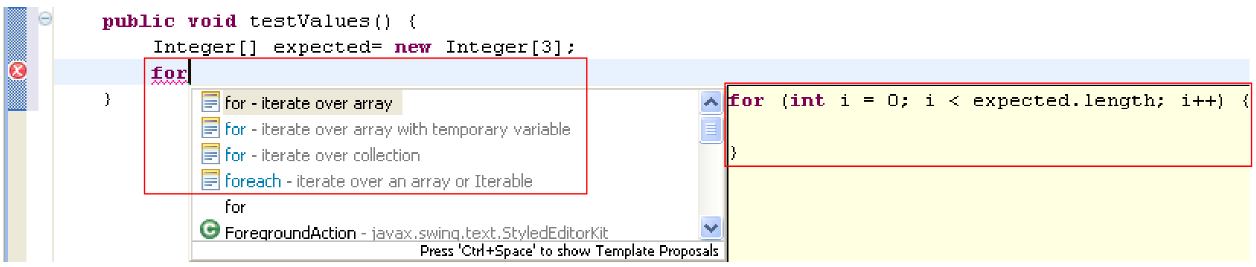
In this section you will use content assist to fill in a template for a common loop structure. Open *junit.samples/VectorTest.java* file in the Java editor if you do not already have it open.

1. Start adding a new method by typing the following:

   ```
   public void testValues() {
       Integer[] expected= new Integer[3];
       for
   ```

Alt + /

2. With the cursor at the end of `for`, press `Ctrl+Space` to enable content assist. You will see a list of common templates for "for" loops. When you single-click a template, or select it with the `Up` or `Down` arrow keys, you'll see the code for the template in its help message. Note that the local array name is guessed automatically.

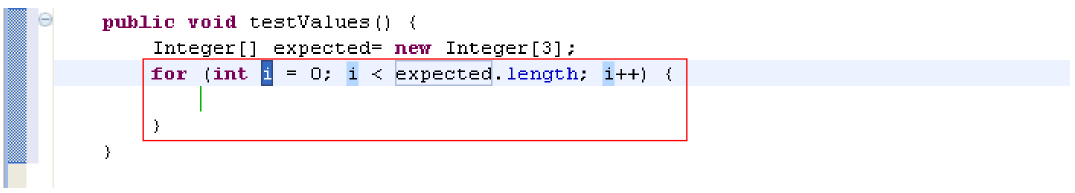


```
public void testValues() {
    Integer[] expected= new Integer[3];
    for
}
```

```
for - iterate over array
for - iterate over array with temporary variable
for - iterate over collection
foreach - iterate over an array or Iterable
for
ForegroundAction - javax.swing.text.StyledEditorKit
Press 'Ctrl+Space' to show Template Proposals
```

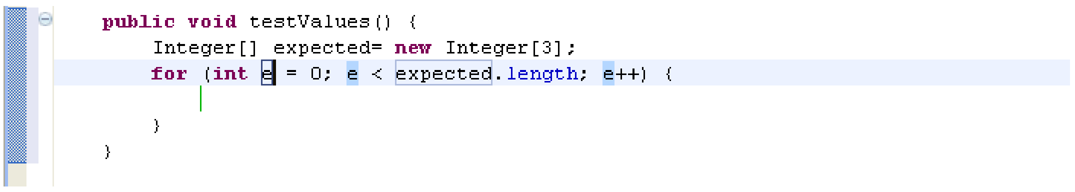
```
for (int i = 0; i < expected.length; i++) {

}
```

3. Choose the `for - iterate over array` entry and press `Enter` to confirm the template. The template will be inserted in your source code.



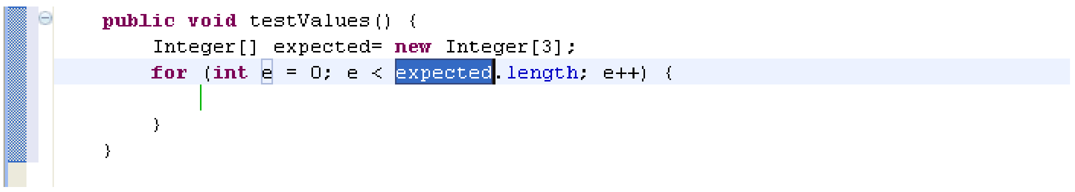
```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int i = 0; i < expected.length; i++) {

    }
}
```

4. Next we change the name of the index variable from *i* to *e*. To do so simply press `e`, as the index variable is automatically selected. Observe that the name of the index variable changes at all places. When inserting a template all references to the same variable are connected to each other. So changing one changes all the other values as well.



```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int e = 0; e < expected.length; e++) {

    }
}
```

5. Pressing the `tab` key moves the cursor to the next variable of the code template. This is the array *expected*.



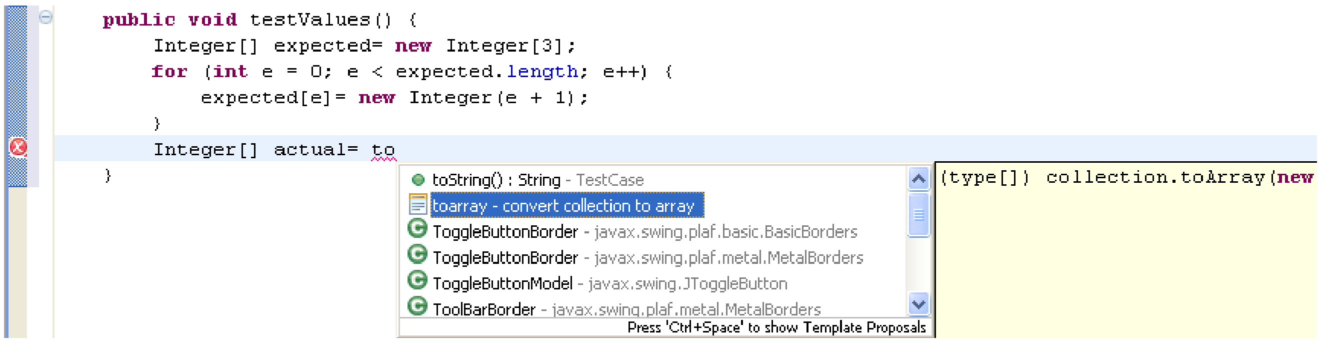
```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int e = 0; e < expected.length; e++) {

    }
}
```

   Since we don't want to change the name (it was guessed right by the template) we press tab again, which leaves the template since there aren't any variables left to edit.

6. Complete the `for` loop as follows:

   ```
   for (int e= 0; e < expected.length; e++) {
       expected[e]= new Integer(e + 1);
   }
   Integer[] actual= to
   ```

7. With the cursor at the end of `to`, press `Ctrl+Space` to enable content assist. Pick `toarray - convert collection to array` and press `Enter` to confirm the selection (or double-click the selection).

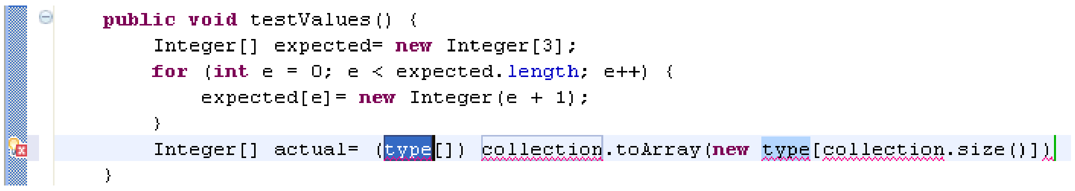


```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int e = 0; e < expected.length; e++) {
        expected[e]= new Integer(e + 1);
    }
    Integer[] actual= to
}
```

toString() : String - TestCase
toarray - convert collection to array
ToggleButtonBorder - javax.swing.plaf.basic.BasicBorders
ToggleButtonBorder - javax.swing.plaf.metal.MetalBorders
ToggleButtonModel - javax.swing.JToggleButton
ToolBarBorder - javax.swing.plaf.metal.MetalBorders
Press 'Ctrl+Space' to show Template Proposals

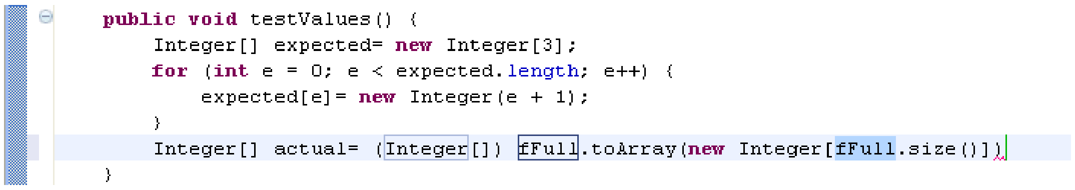
```
(type[]) collection.toArray(new
```

The template is inserted in the editor and `type` is highlighted and selected.



```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int e = 0; e < expected.length; e++) {
        expected[e]= new Integer(e + 1);
    }
    Integer[] actual= (type[]) collection.toArray(new type[collection.size()])
}
```

8. Overwrite the selection by typing `Integer`. The type of array constructor changes when you change the selection.
9. Press `Tab` to move the selection to `collection` and overwrite it by typing `fFull`.



```
public void testValues() {
    Integer[] expected= new Integer[3];
    for (int e = 0; e < expected.length; e++) {
        expected[e]= new Integer(e + 1);
    }
    Integer[] actual= (Integer[]) fFull.toArray(new Integer[fFull.size()])
}
```

10. Add a semicolon and the following lines of code to complete the method:

```
assertEquals(expected.length, actual.length);
for (int i= 0; i < actual.length; i++)
    assertEquals(expected[i], actual[i]);
```

11. Save the file.

Editor Templates

Templates are a structured description of coding patterns that reoccur in source code. The Java editor supports the use of templates to fill in commonly used source patterns. Templates are inserted using content assist (**Ctrl+Space**).

Alt + /

For example, a common coding pattern is to iterate over the elements of an array using a for loop that indexes into the array. By using a template for this pattern, you can avoid typing in the complete code for the loop. Invoking content assist after typing the word `for` will present you with a list of possible templates for a for loop. You can choose the appropriate template by name (`iterate over array`). Selecting this template will insert the code into the editor and position your cursor so that you can edit the details.

Templates can contain template variables. Variables mark the editable locations. They can be resolves to a concrete value when the template is evaluated in its context. They can also provide a list of alternative proposals valid at the given location.

Many common templates are already defined. These can be viewed with the **Java > Editor > Templates** preference page. You can also create your own templates or edit the existing ones.

Java Editor Template Variables

- General Template Variables
- Java Specific Template Variables

Template variables may be used in the template pattern. Variables are resolved to their concrete value when the template is evaluated in its context. Variables may be specified using *simple* or *full* syntax:

- **Simple** variables take the following form:

  `${array}`

  This defines a variable with name 'array' that will resolve to an array. It can be referenced multiple times as is.
- **Full** variables take the following form:

  `${it:var(java.util.Iterator)}`

  This defines a variable with name 'it' that will resolve to a local variable of type `java.util.Iterator`. It can be referenced multiple times by simply giving its name without the type: `${it}`.

If there are several possible matches for a variable, they may be presented as proposals to the user.

General Template Variables

Both Java and Javadoc context define the following variables:

Variable	Description
${cursor}	Specifies the cursor position when the template edit mode is left. This is useful when the cursor should jump to another place than to the end of the template on leaving template edit mode.
${date}	Evaluates to the current date.
${dollar}	Evaluates to the dollar symbol '$'. Alternatively, two dollars can be used: '$$'.
${enclosing_method}	Evaluates to the name of the enclosing name.
${enclosing_method_arguments}	Evaluates to a comma separated list of argument names of the enclosing method. This variable can be useful when generating log statements for many methods.
${enclosing_package}	Evaluates to the name of the enclosing package.
${enclosing_project}	Evaluates to the name of the enclosing project.
${enclosing_type}	Evaluates to the name of the enclosing type.
${file}	Evaluates to the name of the file.
${line_selection}	Evaluates to content of all currently selected lines.
${primary_type_name}	Evaluates to the name primary type of the current compilation unit.
${return_type}	Evaluates to the return type of the enclosing method.
${time}	Evaluates to the current time.
${user}	Evaluates to the user name.
${word_selection}	Evaluates to the content of the current text selection.
${year}	Evaluates to the current year.

Java Specific Template Variables

The Java context additionally defines the following variables. **Note:** In the table below, *id* is a user-chosen name of a new variable.

Variable	Description
${*id*:field(*type[,type])}**	Evaluates to a field in the current scope that is a subtype of any of the given types. If no type is specified, any non-primitive field matches. Example: `${count:field(int)}`
${*id*:var(*type[,type])}**	Evaluates to a field, local variable or parameter visible in the current scope that is a subtype of any of the given types. If no type is specified, any non-primitive variable matches. Example: `${array:var(java.lang.Object[])}`

Variable	Description
${*id*:localVar(*type[,type])}**	Evaluates to a local variable or parameter visible in the current scope that is a subtype of any of the given type. If no type is specified, any non-primitive local variable matches. **${array}** is a shortcut for **${array:localVar(java.lang.Object[])}**, but also matches arrays of primitive types. **${collection}** is a shortcut for **${collection:localVar(java.util.Collection)}**. **${iterable}** is a shortcut for **${iterable:localVar(java.lang.Iterable)}**, but also matches arrays.
${*id*:argType(*variable, n*)}	Evaluates to the *nth* type argument of the referenced template variable. The reference should be the name of another template variable. Resolves to `java.lang.Object` if the referenced variable cannot be found or is not a parameterized type. Example: `${type:argType(vector, 0)} ${first:name(type)} = ${vector:var(java.util.Vector)}.get(0)`
${*id*:elemType(*variable*)}	Evaluates to the element type of the referenced template variable. The reference should be the name of another template variable that resolves to an array or an instance of `java.lang.Iterable`. The **elemType** variable type is similar to **`${id:argType(reference,0)}`**, the difference being that it also resolves the element type of an array. **${array_type}** is a shortcut for **${array_type:elemType(array)}**. **${iterable_type}** is a shortcut for **${iterable_type:elemType(iterable)}**.
${*id*:newName(*reference*)}	Evaluates to an non-conflicting name for a new local variable of the type specified by the reference. The reference may either be a Java type name or the name of another template variable. The generated name respects the code style settings. **${index}** is a shortcut for **${index:newName(int)}**. **${iterator}** is a shortcut for **${iterator:newName(java.util.Iterator)}**. **${array_element}** is a shortcut for **${array_element:newName(array)}**. **${iterable_element}** is a shortcut for **${iterable_element:newName(iterable)}**.
${*id*:newType(*qualifiedTypeName*)}	Evaluates to a type name given the fully qualified Java type name. Evaluates to a simple type name and an import if no conflicting type exists. Evaluates to a fully qualified type name otherwise. Example: `${type:newType(java.util.Iterator)}`
${:import(*type[,type])}**	Adds an import statement for each type. Does nothing if the import already exists. Does nothing if a conflicting import exists. Evaluates to nothing. Example: `${:import(java.util.List, java.util.Collection)}`
${:importStatic(*[qualifiedName[,qualifiedName]*]*)}	Adds a static import statement for each qualified name that is not already imported. The `qualifiedName` is the fully qualified name of a static field or method or it is the qualified name of a type plus a '.*' suffix. Does nothing if a conflicting import exists. Evaluates to nothing. Example: `${is:importStatic(java.util.Collections.EMPTY_SET, 'java.lang.System.*')}`
${*id*:link(*proposal[,proposal])}**	Evaluates to *id* if the list of proposals is empty, evaluates to the first proposal otherwise. The evaluated value is put into linked mode. A proposal window shows all the given proposals. Example: `java.util.Collections.${kind:link(EMPTY_SET, EMPTY_LIST, EMPTY_MAP)}`
${array}	Evaluates to a proposal for an array visible in the current scope.
${array_element}	Evaluates to a name for a new local variable for an element of the **${array}** variable match.

Variable	Description
${array_type}	Evaluates to the element type of the **${array}** variable match.
${collection}	Evaluates to a proposal for a collection visible in the current scope.
${exception_variable_name}	Exception variable name in catch blocks.
${index}	Evaluates to a proposal for an undeclared array index.
${iterator}	Evaluates to an unused name for a new local variable of type `java.util.Iterator`.
${iterable}	Evaluates to a proposal for an iterable or array visible in the current scope.
${iterable_element}	Evaluates to a name for a new local variable for an element of the **${iterable}** variable match.
${iterable_type}	Evaluates to the element type of the **${iterable}** variable match.
${todo}	Evaluates to a proposal for the currently specified default task tag.

Related concepts

Templates

Related reference

Editing templates
Templates preference page
Java content assist
Task tag preferences
Code templates preferences
Code style preferences

Java Editor Templates Preferences

The **Java > Editor > Templates** preference page allows to create new and edit existing templates. A template is a convenience for the programmer to quickly insert often reoccurring source code patterns.

The following buttons allow manipulation and configuration of templates:

Action	Description
New...	Opens the Template dialog to create a new template.
Edit...	Opens the Template dialog to edit the currently selected template.
Remove	Removes all selected templates.
Restore Removed	Restores any preconfigured templates that have been removed.
Revert to Default	Restores any preconfigured templates to their default. This does not modify user-created templates.
Import...	Imports templates from the file system.
Export...	Exports all selected templates to the file system.
Use code formatter	If enabled, the template is formatted according to the code formatting rules specified in the Code Formatter preferences, prior to insertion. Otherwise, the template is inserted as is, but correctly indented.

Java Editor Templates Preferences

The following fields and buttons appear in the dialog:

Option	Description
Name	The name of the template.
Context	The context determines where the template can be used and the set of available pre-defined template variables. The Java editor defines a context for Java and Javadoc areas. • **Javadoc**: Proposal is only offered in Javadoc comments • **Java**: Proposal is only offered in Java code • **Java type members**: Proposal is only offered when completing a Java type member • **Java statement**: Proposal is only offered when completing a Java statement • **SWT**: Proposal is only offered when completing in Java code with SWT on the build path • **SWT type members**: Proposal is only offered when completing a Java type member with SWT on the build path • **SWT statement**: Proposal is only offered when completing a Java statement with SWT on the build path
Automatically insert	If selected, code assist will automatically insert the template if it is the only proposal available at the caret position.
Description	A description of the template, which is displayed to the user when choosing the template.
Pattern	The template pattern.
Insert Variables...	Displays a list of pre-defined context specific template variables.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<templates>
  <template autoinsert="true" context="javadoc" deleted="false" description="author name" enabled="true" id="org.eclipse.jdt.ui.templates.author" name="@author">
    @author ${user}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<b></b>" enabled="true" id="org.eclipse.jdt.ui.templates.b_tag" name="<b>">
    <b>${word_selection}${}</b>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<code></code>" enabled="true" id="org.eclipse.jdt.ui.templates.code_tag" name="<code>">
    <code>${word_selection}${}</code>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<font> color red </font>" enabled="true" name="<font>">
    <font color="red"><b>${word_selection}${}</b></font>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<i></i>" enabled="true" id="org.eclipse.jdt.ui.templates.i_tag" name="<i>">
    <i>${word_selection}${}</i>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<pre></pre>" enabled="true" id="org.eclipse.jdt.ui.templates.pre_tag" name="<pre>">
    <pre>${word_selection}${}</pre>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<tt> </tt>" enabled="true" name="<tt>">
    <tt>${word_selection}${}</tt>${cursor}
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="active task" enabled="true" id="org.eclipse.mylyn.ide.ui.template.activeTask" name="active_task">
    ${activeTaskPrefix}${activeTaskKey}
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="add an element to an array" enabled="true" id="org.eclipse.jdt.ui.templates.arrayadd" name="arrayadd">
    ${array_type}[] ${result:newName(array)} = new ${array_type}[${array}.length + 1];
    System.arraycopy(${array}, 0, ${result}, 0, ${array}.length);
    ${result}[${array}.length]= ${var};
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="merge two arrays into one" enabled="true" id="org.eclipse.jdt.ui.templates.arraymerge" name="arraymerge">
    ${array_type}[] ${result:newName(array1)} = new ${array_type}[${array1:array}.length + ${array}.length];
    System.arraycopy(${array1}, 0, ${result}, 0, ${array1}.length);
    System.arraycopy(${array}, 0, ${result}, ${array1}.length, ${array}.length);
  </template>
  <template autoinsert="true" context="java-members" deleted="false" description="" enabled="true" name="BeforeClass">
    ${testngImport:import('org.testng.annotations.*')}
    @BeforeClass
    public void oneTimeSetUp() {
      ${cursor}
    }
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="dynamic cast" enabled="true" id="org.eclipse.jdt.ui.templates.cast" name="cast">
    ${type} ${new_name} = (${type}) ${name};
```

```xml
</template>
<template autoinsert="false" context="java" deleted="false" description="catch block" enabled="true" id="org.eclipse.jdt.ui.templates.catch" name="catch">
  catch (${Exception} ${exception_variable_name}) {
    ${cursor}// ${todo}: handle exception
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="do while statement" enabled="true" id="org.eclipse.jdt.ui.templates.do" name="do">
  do {
    ${line_selection}${cursor}
  } while (${condition:var(boolean)});
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="else block" enabled="true" id="org.eclipse.jdt.ui.templates.else" name="else">
  else {
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="else if block" enabled="true" id="org.eclipse.jdt.ui.templates.elseif" name="elseif">
  else if (${condition:var(boolean)}) {
    ${cursor}
  }
</template>
<template autoinsert="true" context="java-members" deleted="false" description="Override Object.equals()" enabled="true" name="equals">
  @${:newType(java.lang.Override)}
  public boolean equals(Object obj) {
    if (!(obj instanceof ${enclosing_type}))
      return false;

    ${enclosing_type} another${enclosing_type} = (${enclosing_type}) obj;
    return another${enclosing_type}.${field1} == ${field1}
        && another${enclosing_type}.${field2}.equals(${field2});
  }
</template>
<template autoinsert="false" context="java" deleted="false" description="$FALL-THROUGH$ marker" enabled="true" id="org.eclipse.jdt.ui.templates.fall-through" name="fall-through">
  //$$FALL-THROUGH$$
</template>
<template autoinsert="true" context="javadoc" deleted="false" description="<code>false</code>" enabled="true" id="org.eclipse.jdt.ui.templates.code_tag_false" name="false">
  <code>false</code>
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="iterate over array" enabled="true" id="org.eclipse.jdt.ui.templates.for_array" name="for">
  for (int ${index} = 0; ${index} < ${array}.length; ${index}++) {
    ${line_selection}${cursor}
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="iterate over array with temporary variable" enabled="true" id="org.eclipse.jdt.ui.templates.for_temp" name="for">
  for (int ${index} = 0; ${index} < ${array}.length; ${index}++) {
    ${array_type} ${array_element} = ${array}[${index}];
    ${cursor}
  }
</template>
```

```
<template autoinsert="false" context="java-statements" deleted="false" description="iterate over collection" enabled="true" id="org.eclipse.jdt.ui.templates.for_collection" name="for">
  for (${iteratorType:newType(java.util.Iterator)} ${iterator} = ${collection}.iterator(); ${iterator}.hasNext(); ) {
    ${type:elemType(collection)} ${name:newName(type)} = (${type}) ${iterator}.next();
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="iterate over an array or Iterable" enabled="true" id="org.eclipse.jdt.ui.templates.for_iterable" name="foreach">
  for (${iterable_type} ${iterable_element} : ${iterable}) {
    ${cursor}
  }
</template>
<template autoinsert="true" context="java-members" deleted="false" description="groups test for TestNG" enabled="true" name="Group">
  @Test(groups = "${newName}")
  public void ${newName}() {
    //
  }
</template>
<template autoinsert="true" context="java-members" deleted="false" description="Override Object.hashCode()" enabled="true" name="hashCode">
  @${:newType(java.lang.Override)}
  public int hashCode() {
    int hash = 17;
    hash = 31 * hash + ((int) (${longField} ^ (${longField} >>> 32)));
    hash = 31 * hash + ${intField};
    return hash;
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="if statement" enabled="true" id="org.eclipse.jdt.ui.templates.if" name="if">
  if (${condition:var(boolean)}) {
    ${line_selection}${cursor}
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="if else statement" enabled="true" id="org.eclipse.jdt.ui.templates.ifelse" name="ifelse">
  if (${condition:var(boolean)}) {
    ${cursor}
  } else {

  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="dynamic type test and cast" enabled="true" id="org.eclipse.jdt.ui.templates.instanceof" name="instanceof">
  if (${name:var} instanceof ${type}) {
    ${type} ${new_name} = (${type})${name};
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="lazy creation" enabled="true" id="org.eclipse.jdt.ui.templates.lazy" name="lazy">
  if (${name:var} == null) {
    ${name} = new ${type}(${arguments});
    ${cursor}
  }
```

```xml

  return ${name};
</template>
<template autoinsert="false" context="java-members" deleted="false" description="main method" enabled="true" id="org.eclipse.jdt.ui.templates.main" name="main">
  public static void main(String[] args) {
    ${cursor}
  }
</template>
<template autoinsert="true" context="java-members" deleted="false" description="Mockito" enabled="true" name="mock">
  ${varName} = mock(${Type}.class);${staticImport:importStatic('org.mockito.Mockito.*')}
</template>
<template autoinsert="true" context="java-members" deleted="false" description="Mock Annotation for Mockito" enabled="true" name="Mock Annotation">
  @Mock${mockitoImport:import('org.mockito.*')}
</template>
<template autoinsert="false" context="java-statements" deleted="false" description="create new object" enabled="true" id="org.eclipse.jdt.ui.templates.new" name="new">
  ${type} ${name} = new ${type}(${arguments});
</template>
<template autoinsert="false" context="java" deleted="false" description="non-externalized string marker" enabled="true" id="org.eclipse.jdt.ui.templates.non-nls" name="nls">
  //$$NON-NLS-${N}$$
</template>
<template autoinsert="true" context="javadoc" deleted="false" description="<code>null</code>" enabled="true" id="org.eclipse.jdt.ui.templates.code_tag_null" name="null">
  <code>null</code>
</template>
<template autoinsert="false" context="java-members" deleted="false" description="private method" enabled="true" id="org.eclipse.jdt.ui.templates.private_method" name="private_method">
  private ${return_type} ${name}(${}) {
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-members" deleted="false" description="private static method" enabled="true" id="org.eclipse.jdt.ui.templates.private_static_method" name="private_static_method">
  private static ${return_type} ${name}(${}) {
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-members" deleted="false" description="protected method" enabled="true" id="org.eclipse.jdt.ui.templates.protected_method" name="protected_method">
  protected ${return_type} ${name}(${}) {
    ${cursor}
  }
</template>
<template autoinsert="false" context="java-members" deleted="false" description="public method" enabled="true" id="org.eclipse.jdt.ui.templates.public_method" name="public_method">
  public ${return_type} ${name}(${}) {
    ${cursor}
  }
</template>
<template autoinsert="false" context="java" deleted="false" description="runnable" enabled="true" id="org.eclipse.jdt.ui.templates.runnable" name="runnable">
```

```xml
    new Runnable() {
      public void run() {
        ${line_selection}
      }
    }
  </template>
  <template autoinsert="false" context="java-members" deleted="false" description="static final field" enabled="true" id="org.eclipse.jdt.ui.templates.static_final" name="static_final">
    ${visibility:link(public,protected,private)} static final ${type:link(String,int)} ${NAME};
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="switch case statement" enabled="true" id="org.eclipse.jdt.ui.templates.switch" name="switch">
    switch (${key}) {
      case ${value}:
        ${cursor}
        break;

      default:
        break;
    }
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="synchronized block" enabled="true" id="org.eclipse.jdt.ui.templates.synchronized" name="synchronized">
    synchronized (${mutex:var}) {
      ${line_selection}
    }
  </template>
  <template autoinsert="true" context="java-statements" deleted="false" description="print to standard error" enabled="true" id="org.eclipse.jdt.ui.templates.syserr" name="syserr">
    System.err.println(${word_selection}${});${cursor}
  </template>
  <template autoinsert="true" context="java-statements" deleted="false" description="print to standard out" enabled="true" id="org.eclipse.jdt.ui.templates.sysout" name="sysout">
    System.out.println(${word_selection}${});${cursor}
  </template>
  <template autoinsert="true" context="java-statements" deleted="false" description="print current method to standard out" enabled="true" id="org.eclipse.jdt.ui.templates.systrace" name="systrace">
    System.out.println("${enclosing_type}.${enclosing_method}()");
  </template>
  <template autoinsert="false" context="java-members" deleted="false" description="test method (JUnit 4)" enabled="true" id="org.eclipse.jdt.ui.templates.test_junit4" name="Test">
    ${junit4Import:import('org.junit.*')}
    @Test
    public void ${testName}() {
      ${staticImport:importStatic('org.junit.Assert.*')}${cursor}
    }
  </template>
  <template autoinsert="true" context="java-members" deleted="false" description="" enabled="true" name="TestNG">
    ${testngImport:import('org.testng.annotations.*')}
    @Test
    public void ${testName}() {
      ${staticImport:importStatic('org.testng.Assert.*')}${cursor}
    }
  </template>
  <template autoinsert="false" context="java" deleted="false" description="convert
```

```xml
collection to array" enabled="true" id="org.eclipse.jdt.ui.templates.toarray" name="toarray">
    (${type:elemType(collection)}[]) ${collection}.toArray(new ${type}[${collection}.size()])
  </template>
  <template autoinsert="true" context="java-members" deleted="false" description="Override Object.toString()" enabled="true" name="toString">
    @${:newType(java.lang.Override)}
    public String toString() {
      StringBuilder sb = new StringBuilder();
      sb.append("${enclosing_type}{")
        .append(${field}).append(',')
        .append(${field});
        .append("}");
      return sb.toString();
    }
  </template>
  <template autoinsert="true" context="javadoc" deleted="false" description="<code>true</code>" enabled="true" id="org.eclipse.jdt.ui.templates.code_tag_true" name="true">
    <code>true</code>
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="try catch block" enabled="true" id="org.eclipse.jdt.ui.templates.try" name="try">
    try {
      ${line_selection}${cursor}
    } catch (${Exception} ${exception_variable_name}) {
      // ${todo}: handle exception
    }
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="iterate with enumeration" enabled="true" id="org.eclipse.jdt.ui.templates.while_enumeration" name="while">
    while (${en:var(java.util.Enumeration)}.hasMoreElements()) {
      ${type:argType(en)} ${elem:newName(type)} = (${type}) ${en}.nextElement();
      ${cursor}
    }
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="iterate with iterator" enabled="true" id="org.eclipse.jdt.ui.templates.while_iterator" name="while">
    while (${it:var(java.util.Iterator)}.hasNext()) {
      ${type:argType(it)} ${elem:newName(type)} = (${type}) ${it}.next();
      ${cursor}
    }
  </template>
  <template autoinsert="false" context="java-statements" deleted="false" description="while loop with condition" enabled="true" id="org.eclipse.jdt.ui.templates.while_condition" name="while">
    while (${condition:var(boolean)}) {
      ${line_selection}${cursor}
    }
  </template>
</templates>
```