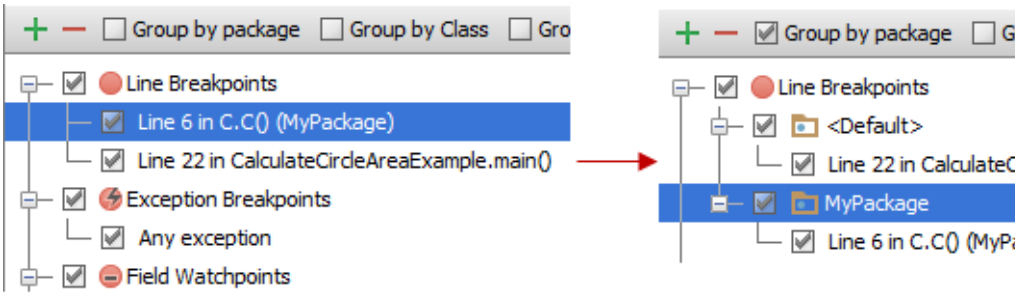
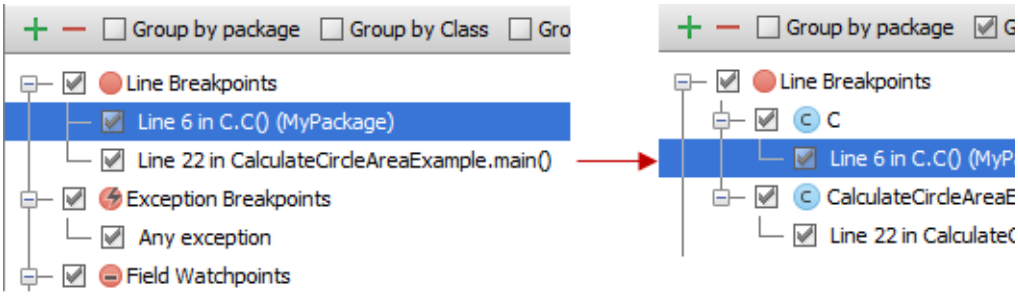
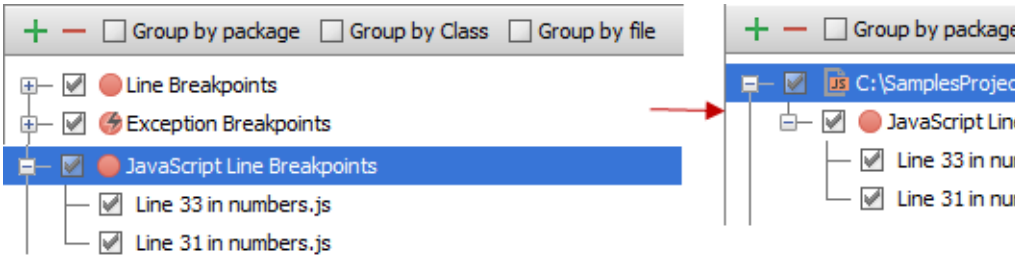
Breakpoints

Run | View Breakpoints

In this section:

Toolbar

Item	Tooltip and shortcut	Description
+	Add Breakpoint `Alt+Insert`	Click to show the list of available breakpoint types. Select the desired type to
−	Remove Breakpoint	Click this button to remove selected breakpoints.
Group by Package		Select this check box to display breakpoints under their respective packages, r 
Group by Class		Select this check box to display breakpoints under their respective classes: 
Group by File		Select this check box to display breakpoints under their respective files: 

Breakpoint options

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Suspend</td><td>
Select this check box to enable suspend policy for a breakpoint.

For Line/ Method/ Exception breakpoints and Field Watchpoints select one of the radio buttons to specify the way the running of the program is paused when a breakpoint is reached. If you work with Flex or JavaScript breakpoints you only need to specify whether you want to suspend program execution when the breakpoint is hit.
<table>
<tr><th>Suspend policy</th><th>Type of breakpoint</th><th>Description</th></tr>
<tr><td>All</td><td>Line/Exception/Field/Method</td><td>When a breakpoint is hit, all threads are suspended.</td></tr>
<tr><td>Thread</td><td>Line/Exception/Field/Method</td><td>When the breakpoint is hit, the thread where the breakpoint is hit is suspended.</td></tr>
<tr><td>Make default</td><td>Line/Exception/Field/Method</td><td>Click this button if you want the suspend policy specified for the breakpoint in question to be used as the default one for the subsequently created breakpoints.</td></tr>
</table>
If the check box is not selected, no threads are suspended.

This feature is useful, for instance, to create a master breakpoint, which, being hit, enables lots of dependent ones. Another way to use is to obtain logging information or calculate some expression at a certain point (to be shown in the console) not interrupting the program execution.
There are certain cases when IntelliJ IDEA will not stop at a breakpoint. Consider the following situation:

Two breakpoints are set at the different methods of a class, and there suspend policy is set to All.
When one of the breakpoints is hit, some step actions are performed.
If at the time of stepping another thread hits the second breakpoint, profuct will not stop there.

</td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Condition</td><td>Select this check box and specify a condition for hitting a breakpoint in the text field.

A condition is a Java Boolean expression (including a method returning <code>true</code> or <code>false</code>), for example, <code>str1.equals(str2)</code>.

This expression should be valid at the line where the breakpoint is set, and is evaluated every time the breakpoint is reached.If the evaluation result is <code>true</code>, user-selected actions are performed.

If evaluation result is <code>true</code>, user-selected actions are performed.

If the result is <code>false</code>, the breakpoint does not produce any effect. If the Debugger cannot evaluate the expression, it displays the Condition evaluation error message. You can select whether you would like to stop at this breakpoint or ignore it.

Conditions for field/method/exception breakpoints are calculated in the context for the given field/method/exception.</td></tr>
<tr><td>Actions</td><td>In this section, select the check boxes to define the action to be performed on hitting a breakpoint:
<table><tr><th>Action</th><th>Type of breakpoint</th><th>Description</th></tr><tr><td>Log message to console</td><td>All types</td><td>Select this check box if you want a log message to be displayed in the console output when the breakpoint is hit.</td></tr><tr><td>Log evaluated expression</td><td>Line breakpoints</td><td>Select this check box if you wish to evaluate a certain expression at this breakpoint and to export result to the console output.

If the expression to be evaluated is incorrect when a particular breakpoint is reached, the console output displays an error message: Unable to evaluate expression <your_expression>.</td></tr><tr><td>Temporary</td><td>All types</td><td>A temporary breakpoint stops the program just once, and is then removed.</td></tr></table></td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Watch</td><td>This options group is available for method breakpoints and field watchpoints only, and is required to specify under which circumstance the debugger should trigger a breakpoint.
<table>
<tr><th>Watch</th><th>Type of breakpoint</th><th>Description</th></tr>
<tr><td>Field access</td><td>Field watchpoints</td><td>Stands for triggering breakpoint every time the field is accessed.</td></tr>
<tr><td>Field modification</td><td>Field watchpoints</td><td>This check box is selected when simple read attempts shouldn't cause the breakpoint to trigger.</td></tr>
<tr><td>Method entry</td><td>Method breakpoints</td><td>Stands for triggering breakpoint every time the method is entered.</td></tr>
<tr><td>Method exit</td><td>Method breakpoints</td><td>Stands for triggering breakpoint every time the method is exited.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Filters</td><td>In this area, specify the filters to be used.
<table>
<tr><th>Filter</th><th>Type of breakpoint</th><th>Description</th></tr>
<tr><td>Instance filters</td><td>Line/ Method/ Field Watchpoint/ Exception</td><td>An instance filter is used to limit breakpoint hits only with particular object instances using instance IDs. The instance ID value can be introduced manually or using the Instance Filters dialog box called by clicking the ellipsis button. Existing instance filters are indicated by the instance ID delimited with spaces.</td></tr>
</table></td></tr>
</table>

Option	Filter	Type of breakpoint	Description

	Class filters	Line/ Method/ Field Watchpoint/ Exception	Select this check box to have the breakpoint behave differently in relation to particular classes.
			
			Define the class filter to appoint the classes where you want the breakpoint to be hit and the classes where the breakpoint should not be triggered.
			
			Classes in a filter can be identified by their names or by means of *class patterns*.
			
			A class pattern is a string that may start or end with an asterisk (*). The asterisk in a pattern means any number (including zero) of any characters. The patterns are matched against fully qualified class names.
			
			The breakpoint behavior is different in relation to classes specified by their names or using class patterns.
			
			A filter specified through a class name points at the class itself as well as at all its subclasses (i.e. the classes directly or indirectly extending this one).
			
			A filter specified through a class pattern points at the classes whose fully qualified names match the pattern. The subclasses of such classes are selected only if their fully qualified names also match the specified pattern.
			
			You can define a class filter in two ways:
			
			■ Click the **Browse** button ... and configure the filter in the Class Filters dialog box, that opens.
			■ Type the filter manually in the text box. Use the following syntax:
			  ■ Use spaces to separate class names and class patterns from each other.
			  ■ The classes to be excluded should have a minus in preposition.
			
			For example, the filter `package1.Class1 *s2 -package3.Class3` means that the corresponding breakpoint:
			
			  ■ Should be triggered in the class `package1.Class1` and all its subclasses, and also in the classes whose fully qualified class names end in `s2`.
			  ■ Should not be triggered in the class `package3.Class3` and all its subclasses.

Option	Filter	Type of breakpoint	Description
	Pass count	Line/ Method/ Field Watchpoint/ Exception	Specify the integer number, on which hit of the breakpoint it should be triggered. After the specified number of passes, the breakpoint is hit. This function is helpful for debugging loops or methods called several times. When the execution process comes to a breakpoint, where **Pass count** is set, the debugger reduces the count value by 1 and compares it to zero. If the comparison result is `true`, the breakpoint is hit. If it is `false`, no other actions are performed until the next breakpoint is reached. The **Pass count** condition can be satisfied only once. In other words, if you have a loop inside a method and the **Pass count** condition has been honored once, the breakpoint will not be hit the next time the said method is called. This option is only enabled, when **Condition**, **Instance filters** and **Class filters** options are disabled.

See Also

Concepts:

- Breakpoints

Procedures:

- Using Breakpoints

Reference:

- Class Filters Dialog
- New Filter Dialog

Web Resources:

- Developer Community