Local Debug

Summary

Local debugger provides the following features in order for programs running can be debugged on the spot.

Setting breakpoints

Debug View lets you set breakpoints for analyzing runtime program states.

Step debugging

Programs can be stepped through to see the code flow and variable states.

Step filtering

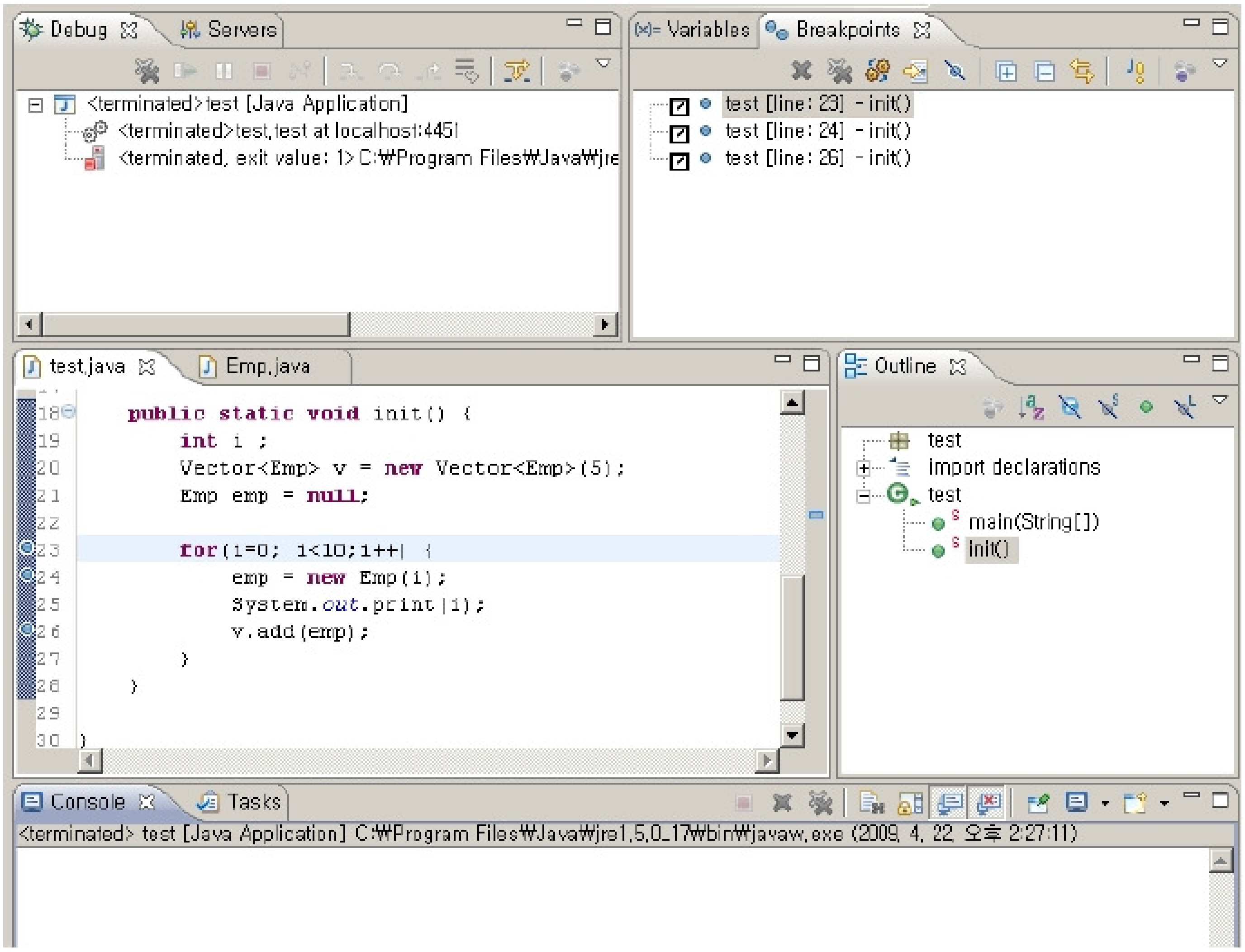
Certain methods can be excluded and prevented from being stepped-in.

Evaluating Expressions

Running programs can be status checked without being halted.

Variables View

Variable values can be checked while the programs are running.



```
<terminated>test [Java Application]
    <terminated>test.test at localhost:4451
    <terminated, exit value: 1>C:₩Program Files₩Java₩jre
```

```
test [line: 23] - init()
test [line: 24] - init()
test [line: 26] - init()
```

```
    public static void init() {
        int i ;
        Vector<Emp> v = new Vector<Emp>(5);
        Emp emp = null;

        for(i=0; i<10;i++) {
            emp = new Emp(i);
            System.out.print(i);
            v.add(emp);
        }
    }
}
```

```
test
import declarations
test
    main(String[])
    init()
```

```
<terminated> test [Java Application] C:₩Program Files₩Java₩jre1.5.0_17₩bin₩javaw.exe (2009. 4. 22 오후 2:27:11)
```

Manual

Setting breakpoints

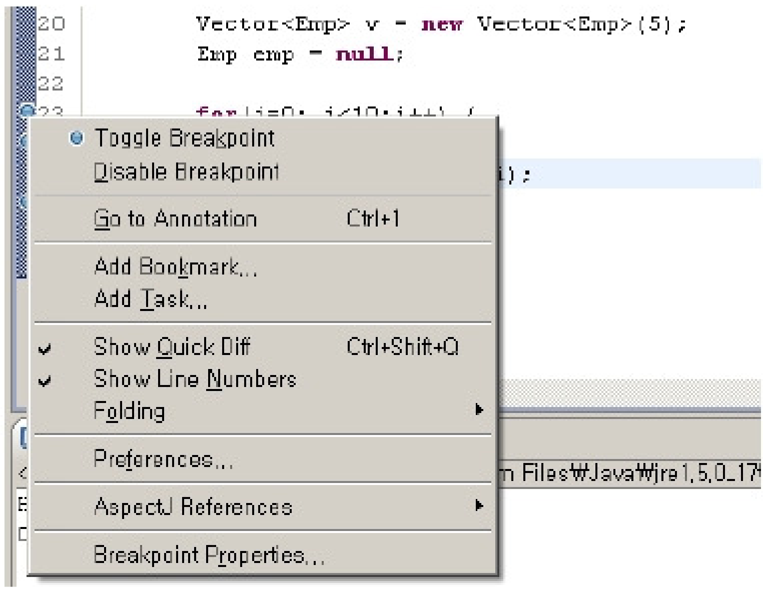
You can set breakpoints as debugging points to analyze any lines of code in question.

Add/remove breakpoints

Double-click the marker bar on the line you want to toggle breakpoints, or use the context menu.

Disable breakpoints

Marker bar's context menu lets you disable breakpoints so that you do not need to remove them in order to temporarily disable them, and thus let the code execution run through.



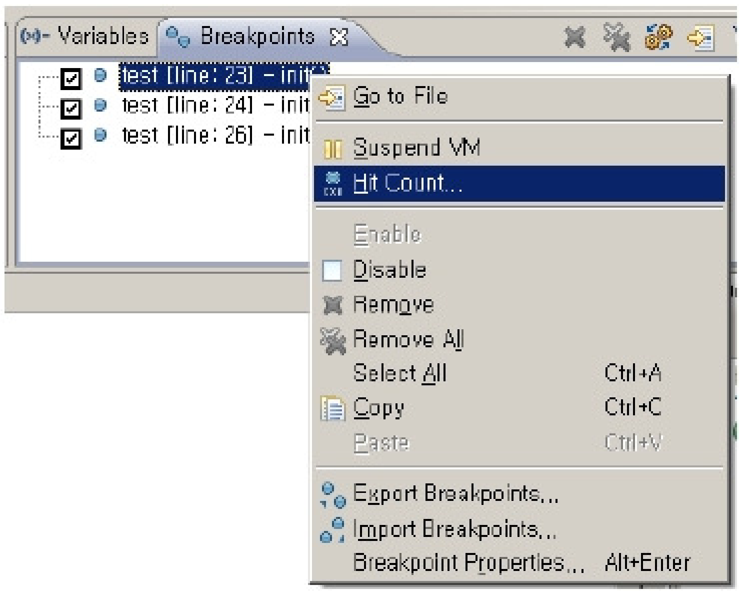
```
        Vector<Emp> v = new Vector<Emp>(5);
        Emp emp = null;
```

Breakpoints View

The entire set of breakpoints can be conveniently managed in the Breakpoints View.

Hit Count

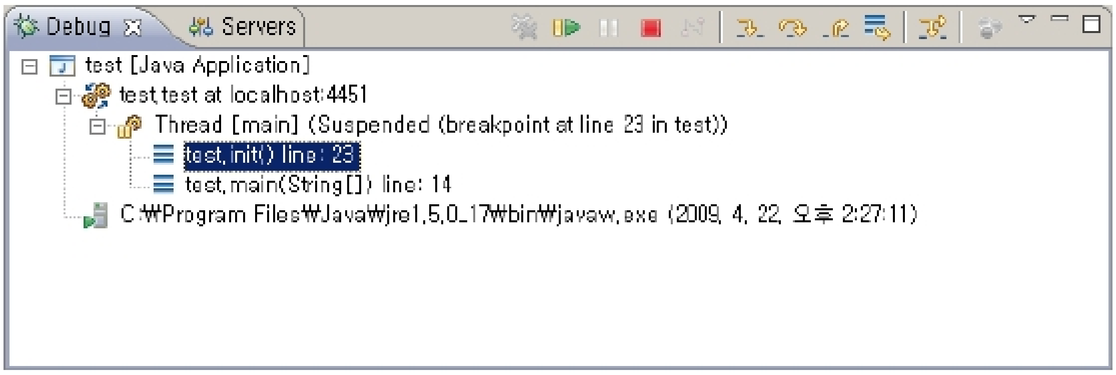
Monitors how many hits a given line has received in an execution run, and halts the program when it reaches a preset value.



```
test [line: 23] - init
test [line: 24] - init
test [line: 26] - init
```

Step-wise debugging

Programs can be stepped over or in from breakpoints for debugging purposes.



```
test [Java Application]
  test.test at localhost:4451
    Thread [main] (Suspended (breakpoint at line 23 in test))
      test.init() line: 23
      test.main(String[]) line: 14
  C:₩Program Files₩Java₩jre1.5.0_17₩bin₩javaw.exe (2009. 4. 22. 오후 2:27:11)
```

Step Into(F5)

Step a line, and if it is a method call, step into the method code block. Used for checking the method's implementation and functionality.

Step Over(F6)

Step a line, and does not step in even if it is a method call.

Step Return(F7)

Exits the current method, and stops on the line of the method's call.
.

Resume(F5)

Runs the program until end of program or another breakpoint.

Suspend

Pauses the currently running program's thread. This displays the current call stack, as well as local and other viewable variables in the Variables View.

Drop to Frame

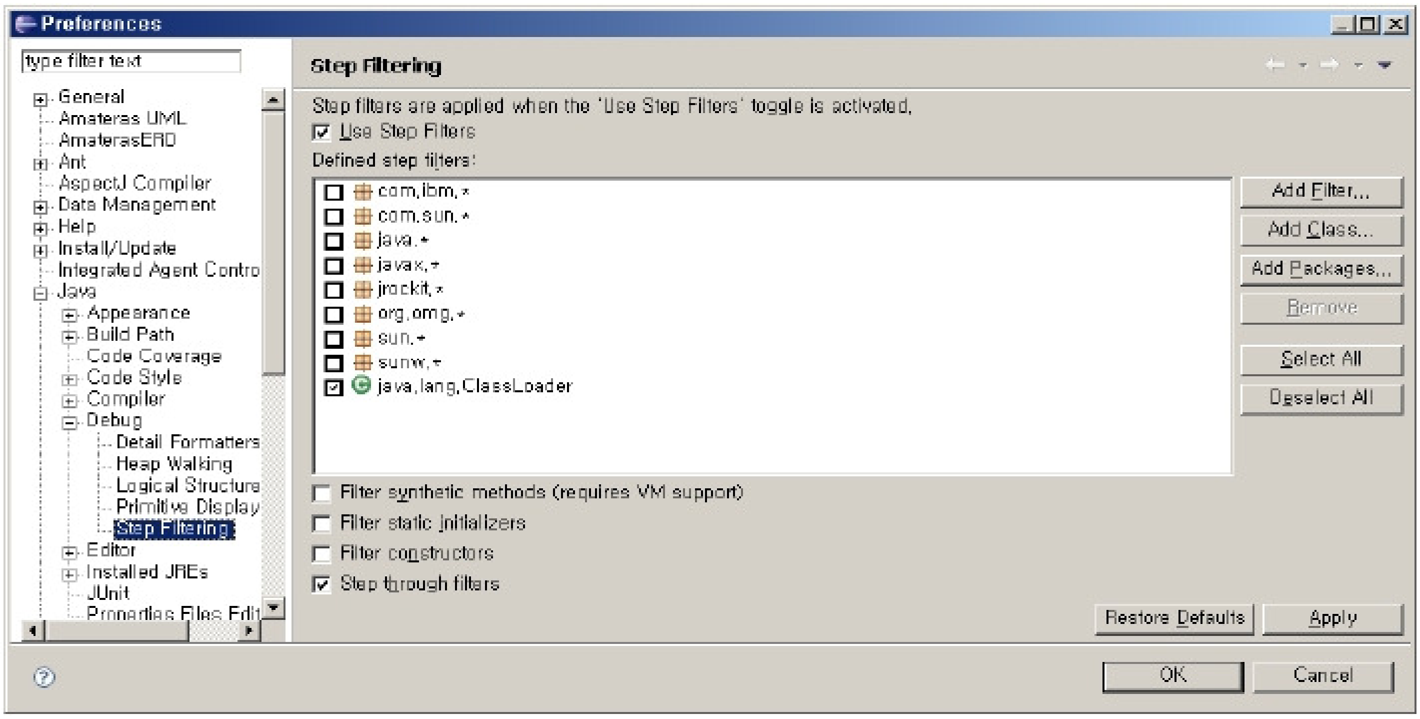
Jumps back to the first line of the current stack frame. Used for re-starting the execution of the current method.

Terminate

Ends and exits the program.

Step filtering

Step filtering is a feature such that if a program is a filtered (targeted) program, it works identically as step-over functionality, and if it is not such a program, it works identically as step-into functionality. Debug View's Use Step Filters button activates this feature.


Step Filtering

Step filters are applied when the 'Use Step Filters' toggle is activated.

[✓] Use Step Filters

Defined step filters:

- [ ] com.ibm.*
- [ ] com.sun.*
- [ ] java.*
- [ ] javax.*
- [ ] jrockit.*
- [ ] org.omg.*
- [ ] sun.*
- [ ] sunw.*
- [✓] java.lang.ClassLoader

[ ] Filter synthetic methods (requires VM support)
[ ] Filter static initializers
[ ] Filter constructors
[✓] Step through filters

Filter synthetic methods

Filters out all synthetic methods. Synthetic methods are methods which the compiler artificially creates and includes in the byte code to satisfy certain programming language requirements when compiling a particular class. These methods usually are not to be debugged by application developers.

Filter static initializers

Filters out all static initializers and static member initializations.

Filter constructors

Filters out all constructors.

Step through filters

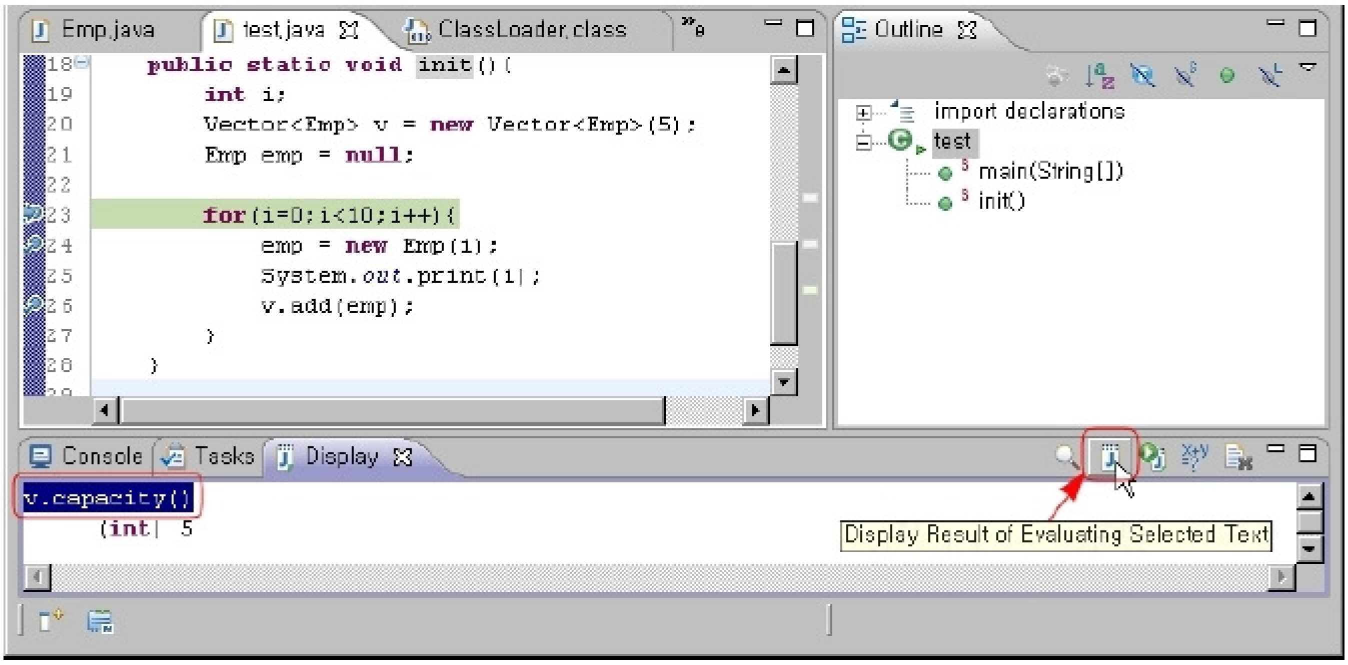
Default option for Step filtering

Evaluating_Expressions

Display

- Display View allows you to check instances' values or their alignments without halting programs. It allows printing instance values or running particular methods to check status information.

1. Breakpoint or step to your desired part of code.
2. From the menu, choose Window > Show View > Display, to display the Display View.
3. Enter a name of an object or method to evaluate, and highlight the selected text.
4. Click the "Display Result Evaluating Selected Text" on the top right of the Display View.



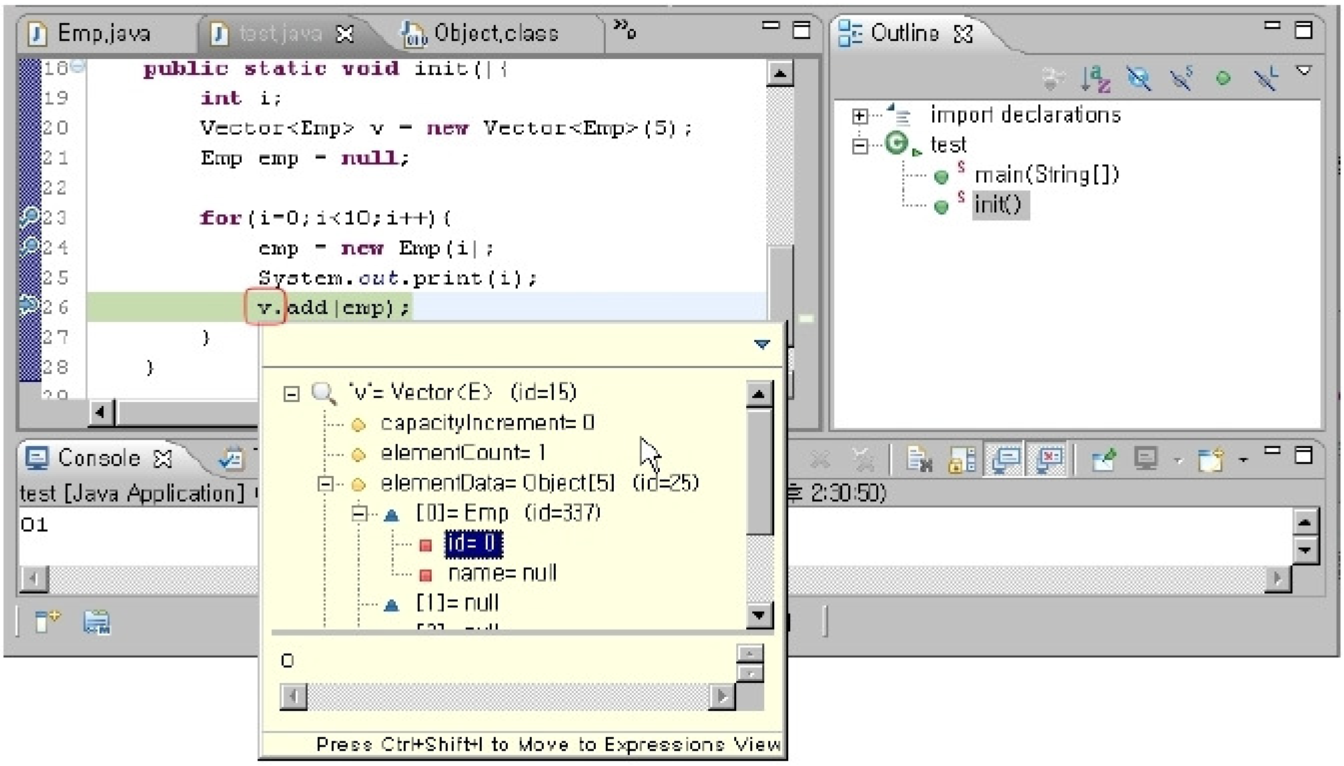
```
public static void init(){
    int i;
    Vector<Emp> v = new Vector<Emp>(5);
    Emp emp = null;

    for(i=0;i<10;i++){
        emp = new Emp(i);
        System.out.print(i);
        v.add(emp);
    }
}
```

```
v.capacity()
    (int) 5
```

Inspect

- Display View's downside is that it always displays in Strings. You can use Inspect feature instead to further inspect objects.

1. Choose an object from the editor or from the Display View.
2. Choose Inspect from the context menu, or use Ctrl+Shift+I.



```
public static void init(){
    int i;
    Vector<Emp> v = new Vector<Emp>(5);
    Emp emp = null;

    for(i=0;i<10;i++){
        emp = new Emp(i);
        System.out.print(i);
        v.add(emp);
    }
}
```

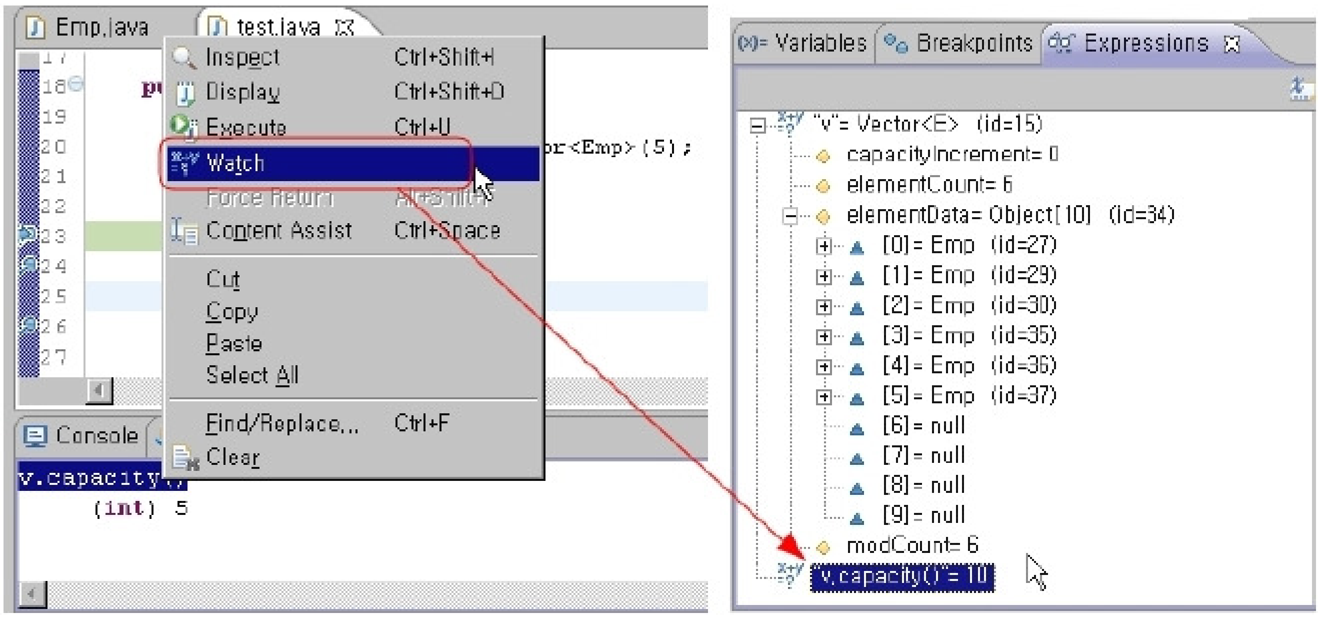
```
'v'= Vector<E>  (id=15)
    capacityIncrement= 0
    elementCount= 1
    elementData= Object[5]  (id=25)
        [0]= Emp  (id=337)
            id= 0
            name= null
        [1]= null
0
```

```
test [Java Application] 2:30:50)
01
```

Watch

- Variables can be continuously monitored once they are added to the watch list. The Watch feature can also handle expressions and lets you monitor its evaluated result just the same as with any variable.

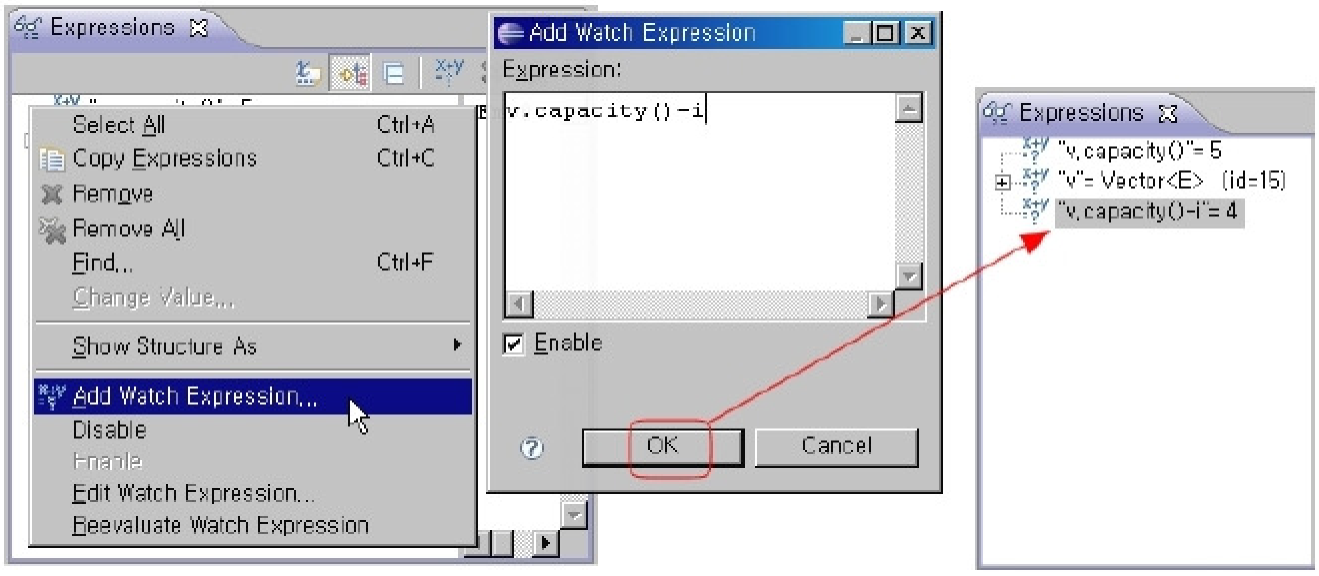
1. Choose an expression to track from the editor or the Display View.
2. From the context menu, choose Watch.
3. Watch pane gets added on the Expression View.
4. To add expressions to watch, choose Add Watch Expression... from the Expressions View's context menu.



```
v.capacity()
     (int) 5
```

```
"v"= Vector<E>  (id=15)
    capacityIncrement= 0
    elementCount= 6
    elementData= Object[10]  (id=34)
        [0]= Emp  (id=27)
        [1]= Emp  (id=29)
        [2]= Emp  (id=30)
        [3]= Emp  (id=35)
        [4]= Emp  (id=36)
        [5]= Emp  (id=37)
        [6]= null
        [7]= null
        [8]= null
        [9]= null
    modCount= 6
"v.capacity()"= 10
```

Configure an object for Watch.



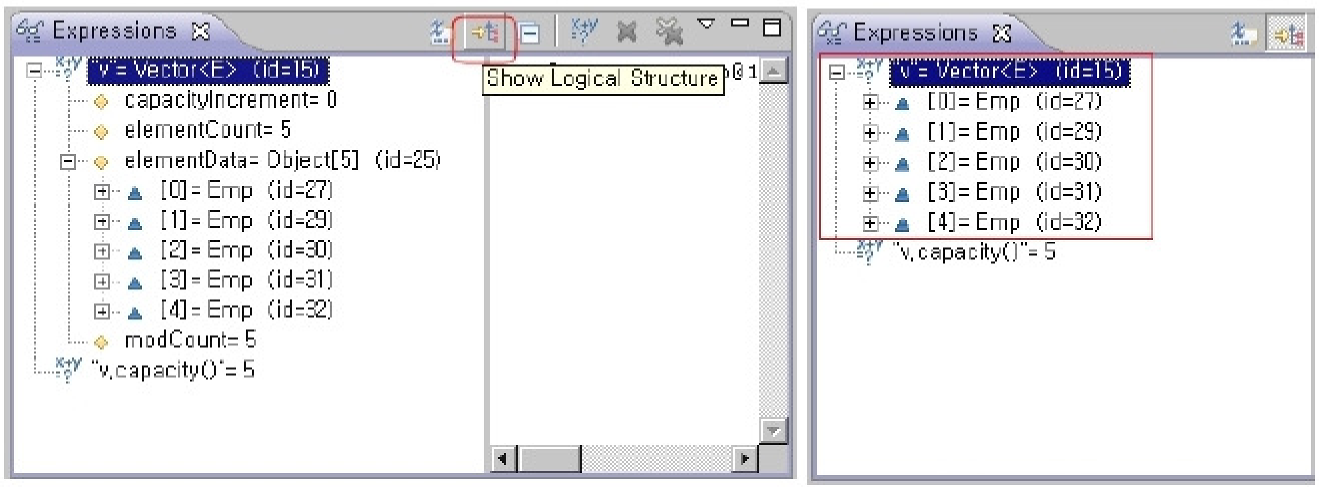
```
v.capacity()-i
```

```
"v.capacity()"= 5
"v"= Vector<E>  (id=15)
"v.capacity()-i"= 4
```

Adding an optional formula in the Expressions view.

Viewing logical structures

- You can inspect object and instances closely in a compact manner by using Show Logical Structure option. Click the Show Logical Structure button in the View toolbar.

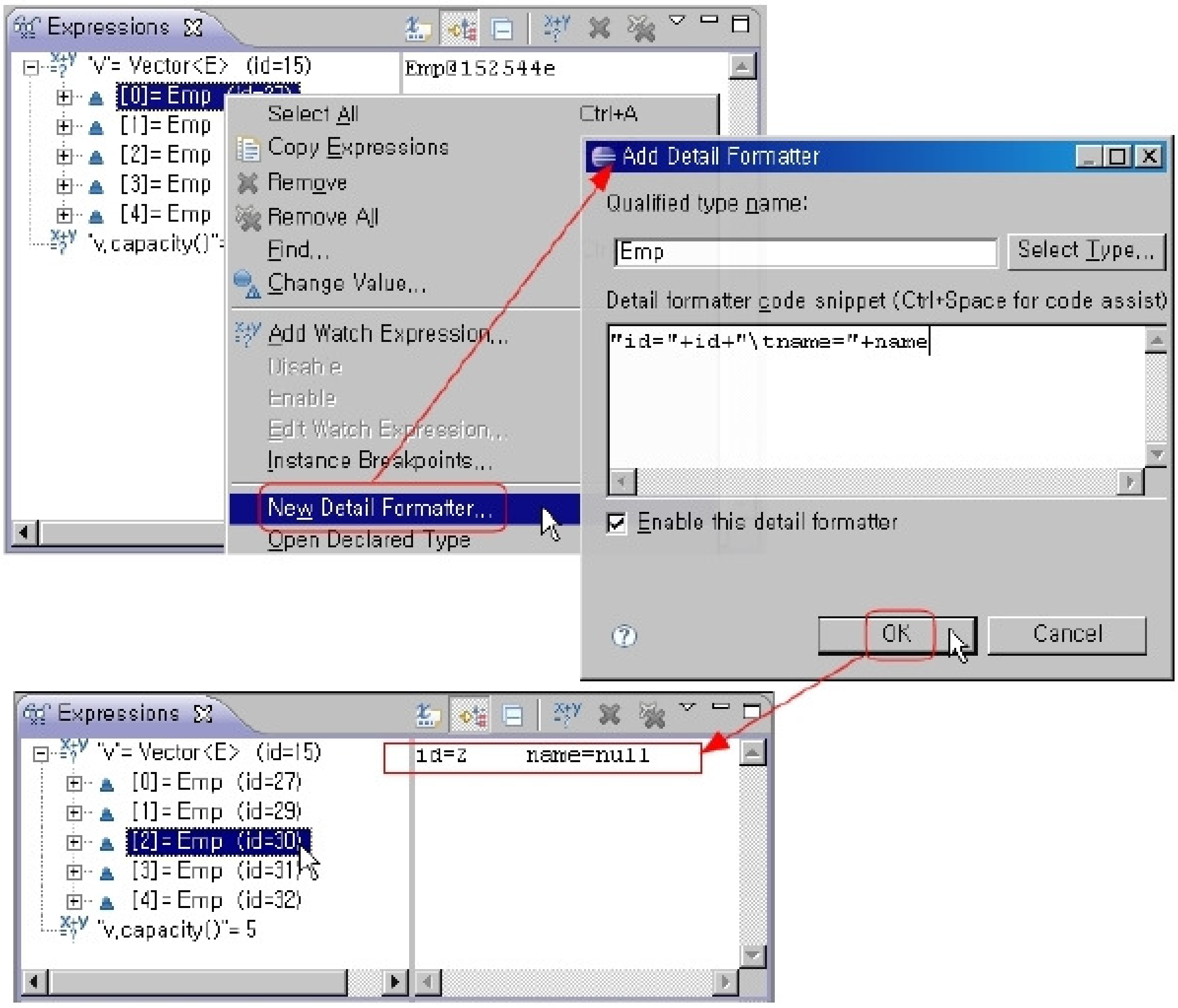


```
"v"= Vector<E>  (id=15)
    capacityIncrement= 0
    elementCount= 5
    elementData= Object[5]  (id=25)
        [0]= Emp  (id=27)
        [1]= Emp  (id=29)
        [2]= Emp  (id=30)
        [3]= Emp  (id=31)
        [4]= Emp  (id=32)
    modCount= 5
"v.capacity()"= 5
```

```
"v"= Vector<E>  (id=15)
    [0]= Emp  (id=27)
    [1]= Emp  (id=29)
    [2]= Emp  (id=30)
    [3]= Emp  (id=31)
    [4]= Emp  (id=32)
"v.capacity()"= 5
```

Detail formatter

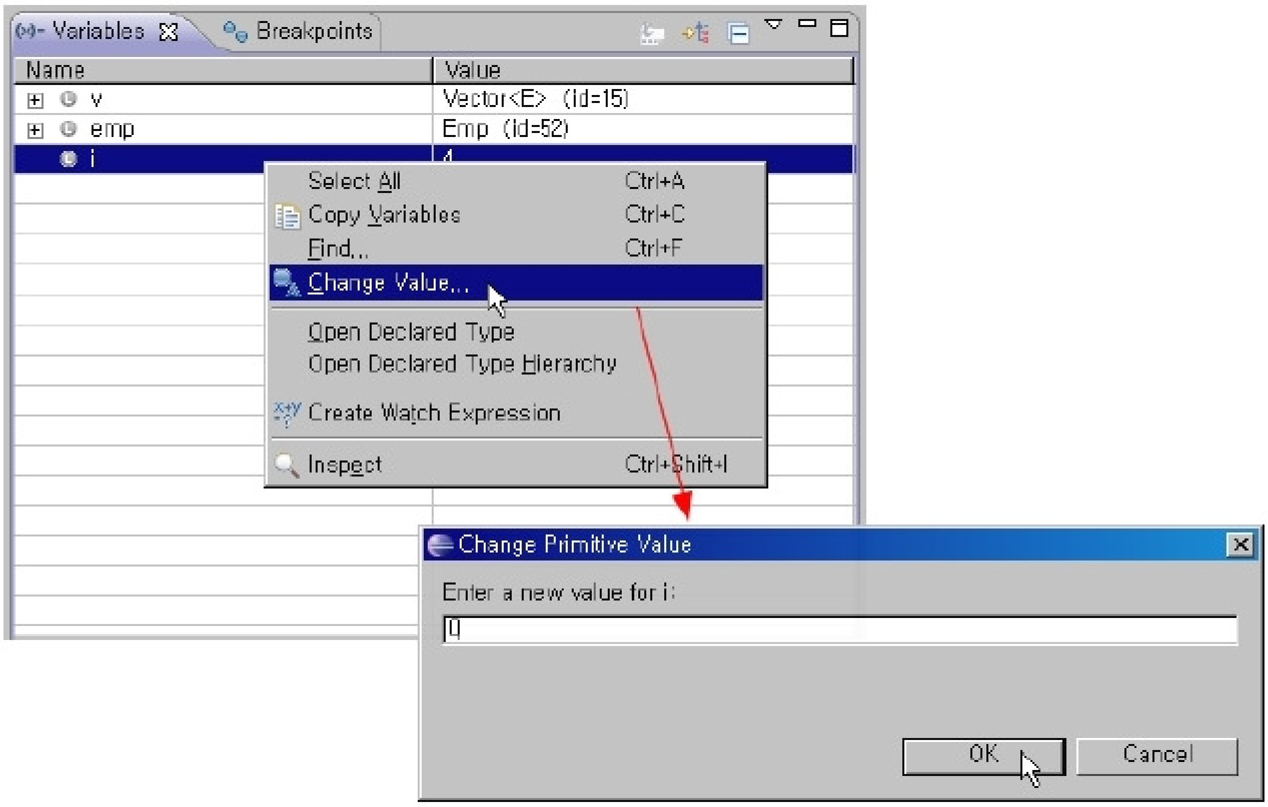
- Detail formatter lets you choose display option for viewing instance types in the Details panel, which leverages objects' detail formatter, or just to String Methods. In order to set detail formatter, choose an expression from the Expressions View, then choose New Detail Formatter from the context menu. Then in the dialog, formulate the expression you want to monitor.



Modifying variable values

- You can change variable values on-the-fly while debugging to test your code.

1. Choose a variable from the Variable View.
2. Right-click to bring up the context menu, choose Change Value, then enter a value.



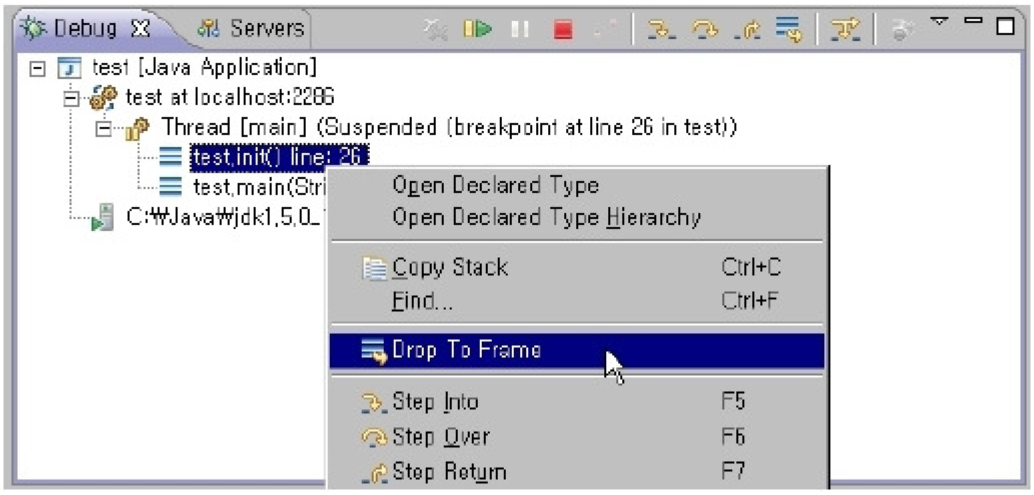
Hot Code Replace

- You can change the source code on-the-fly while in the debugging mode, and this feature is called Hot Code Replace. Change any source code in the editor, press Ctrl+S to save, and it will automatically be re-compiled into the currently running program. This works in all cases except when you add new member variables or methods.

Drop to Frame

- While debugging, you can use Drop to Frame feature to jump back to the very beginning of the current method. (Supported in VM versions 1.4 or higher)

1. From the Debug View, choose a stack frame for your target method.
2. Click the Drop to Frame button from the view toolbar, or choose Drop to Frame from the context menu after right-clicking.
3. Current line will jump to the beginning of the method.

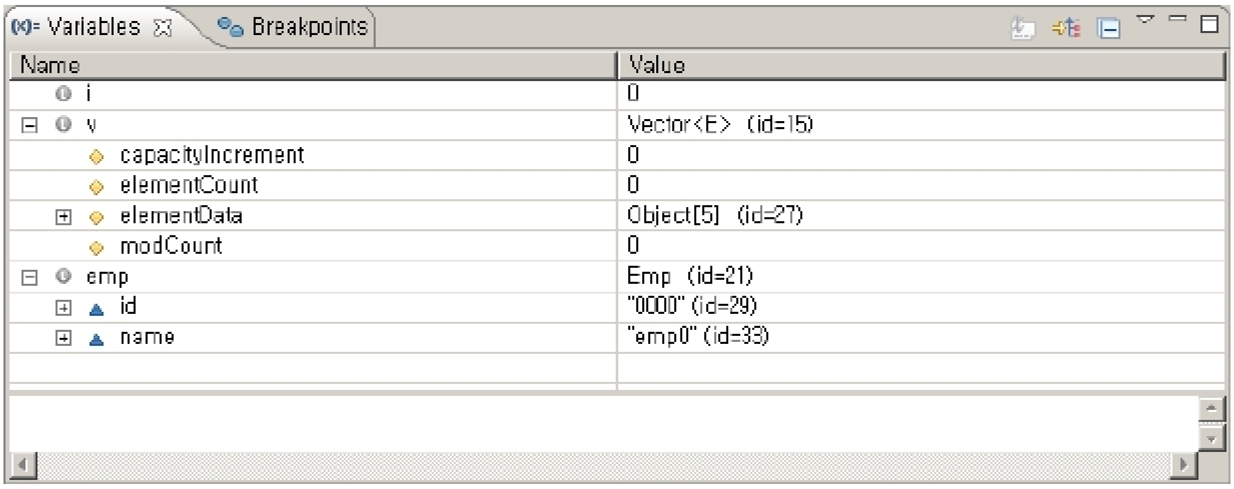


Variables View

You can check live variable values from the Variables View.

Displaying Variables View

From the Debug Perspective, you can access the Variables View as below.



Name	Value
i	0
v	Vector<E> (id=15)
capacityIncrement	0
elementCount	0
elementData	Object[5] (id=27)
modCount	0
emp	Emp (id=21)
id	"0000" (id=29)
name	"emp0" (id=33)

Sample

You can use the following Java source archive to test the debug features mentioned above. debug_sample.zip

References

- Eclipse Help - Java development user guide
- http://help.eclipse.org/help32/topic/org.eclipse.jdt.doc.user/concepts/clocdbug.htm
- [http://help.eclipse.org/help32/topic/org.eclipse.jdt.doc.user/concepts/clocdbug.htm]