

Unit 8. Debug applications

What this unit is about

This unit introduces the Eclipse-based debugging capabilities. You learn to step through code, add and configure breakpoints, and view and modify variable values during debugging.

What you should be able to do

After completing this unit, you should be able to:

- Use the Debug perspective to debug Java applications
- Step through code using the Debug view toolbar
- Add and configure breakpoints in a Java application
- View and change variables during debugging
- Execute and inspect expressions during debugging

How you will check your progress

- Checkpoint questions
- Lab exercise

Unit objectives

After completing this unit, you should be able to:

- Use the Debug perspective to debug Java applications
- Step through code using the Debug view toolbar
- Add and configure breakpoints in a Java application
- View and change variables during debugging
- Execute and inspect expressions during debugging


© Copyright IBM Corporation 2011

Figure 8-1. Unit objectives

WD154 / VD1542.0

Notes:

Debug tools

- Eclipse and Rational Application Developer provide a visual debugging tool for finding problems in Java code
 - Launch the debug tool using the **Debug** button  on the toolbar
 - The workbench prompts you to open the **Debug** perspective and provides debugging capabilities
- Debug tool capabilities:
 - Set breakpoints in the Java editor
 - Modify data values while stepping through code
 - Modify and apply changes to the application while debugging

© Copyright IBM Corporation 2011

Figure 8-2. Debug tools

WD154 / VD1542.0

Notes:

Both Eclipse and Rational Application Developer provide a powerful, visual debug environment.

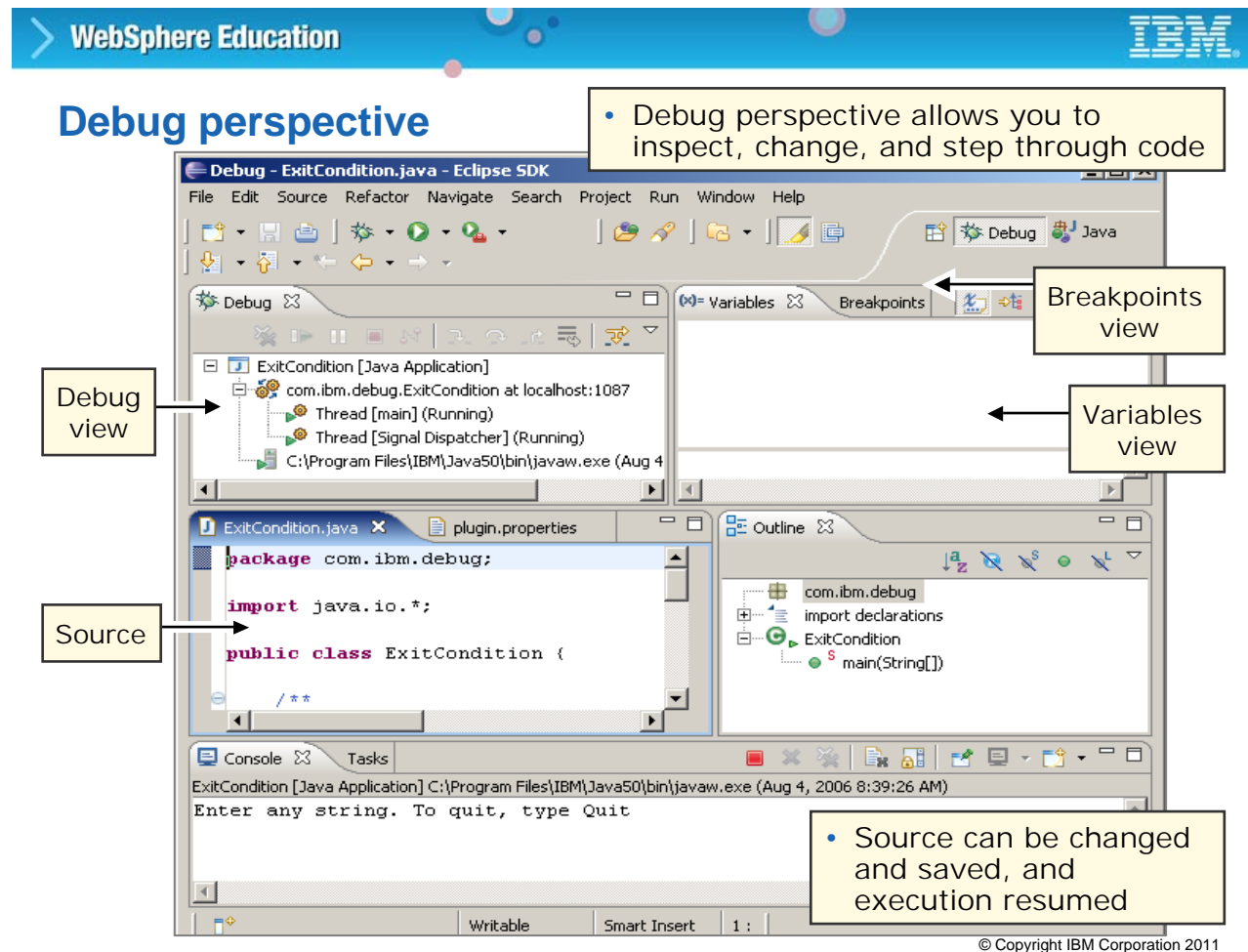


Figure 8-3. Debug perspective

WD154 / VD1542.0

Notes:

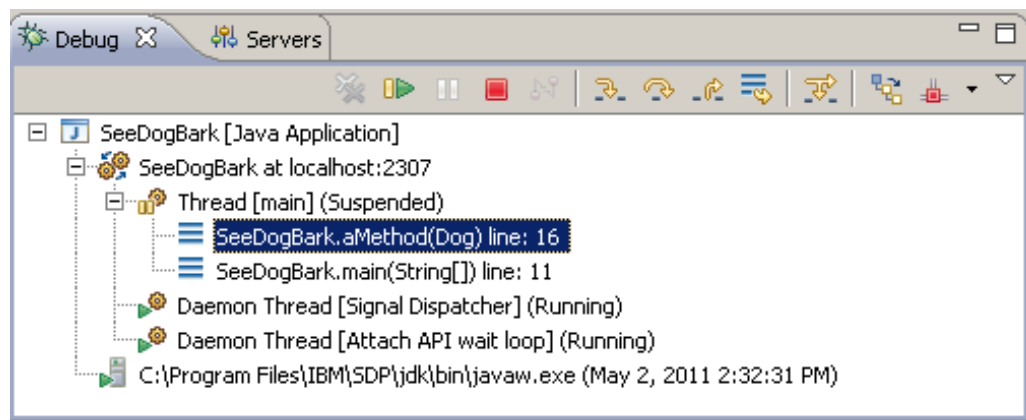
The Debug perspective is designed for debugging your Java program. It includes an editor area and the following views:

- Debug
- Breakpoints
- Expressions
- Variables
- Display
- Outline
- Console

Debug view

Debug view displays:

- The process for each target being run
- A list of stack frames for each target being debugged
 - A stack frame corresponds to a method called before the exception or breakpoint occurred
 - Stack frames are in reverse chronological order (last one was executed first)
- Terminated launches



© Copyright IBM Corporation 2011

Figure 8-4. Debug view

WD154 / VD1542.0

Notes:

The Debug view shows a stack frame of the processes and threads for the program that is being debugged.

Stepping through code commands (1 of 2)

1. **Remove All Terminated Launches:** clears all terminated debug targets from the view
2. **Resume:** continues execution of a suspended thread until a breakpoint is encountered or the thread ends
3. **Suspend:** suspends a running thread so that you can step through code, inspect, and so on
4. **Terminate:** ends the execution of the selected debug target



© Copyright IBM Corporation 2011

Figure 8-5. Stepping through code commands (1 of 2)

WD154 / VD1542.0

Notes:

When execution has been suspended at a breakpoint, the method source for the current stack frame can be executed line by line using the navigation buttons provided on the Debug menu.

Stepping through code commands (2 of 2)

5. **Step Into:** Steps into a method call and executes the first line of code in that method; steps into the highlighted statement.
6. **Step Over:** Executes the next line of code in the current method; steps over the highlighted statement.
7. **Step Return:** Continues execution until the end of the current method (until a return).
8. **Drop to Frame:** Allows you to drop back and reenter a specified stack frame.
9. **Step with Filters:** Continues execution until the next line of code which is not filtered out.



© Copyright IBM Corporation 2011

Figure 8-6. Stepping through code commands (2 of 2)

WD154 / VD1542.0

Notes:

The **Step Over** feature steps over the highlighted statement. Execution continues at the next line either in the same method; or, if you are at the end of a method, it continues in the method from which the current method was called. The cursor jumps to the declaration of the method and selects this line.

The **Drop to Frame** feature is similar to running backwards and restarting your program part of the way through.

Step filtering is a preference, controlled by navigating to **Window > Preferences > Java > Debug > Step Filtering**.

Step filters allow you to avoid debugging internal WebSphere Application Server methods if you are developing an Enterprise Java application, so that you only debug your application. You can also filter out any other external packages that you do not want to debug.

Breakpoints

- Breakpoints are temporary markers you place in your program to tell the debugger to suspend execution of your program at that point
- Reaching a breakpoint in a method causes the **Debug** perspective to open
 - The method at the breakpoint appears on the stack frame
 - Source can then be stepped through and objects inspected
- Breakpoints are set until they are explicitly removed from a method
 - They can be temporarily disabled
- The breakpoint where execution has paused is marked with an arrow



© Copyright IBM Corporation 2011

Figure 8-7. Breakpoints

WD154 / VD1542.0

Notes:

You can choose, based on a preference setting, which perspective opens when the debugger encounters a breakpoint. The most convenient perspective for this purpose is the Debug perspective.

When a breakpoint is enabled, it causes a thread to suspend whenever it is hit.

1. Enabled breakpoints are indicated with a blue circle.
2. When a breakpoint is disabled, it does not cause threads to suspend. Disabled breakpoints are indicated with a white circle.
3. Enabled breakpoints that are successfully installed in a class in a VM at run time are indicated with a check mark overlay.
4. Conditional breakpoints have a question mark overlay.
5. The next line of code about to be run is indicated with an arrow.

Breakpoints view

- Lists all the breakpoints you have set in the workbench as well as Java Exception breakpoints
 - Double-click a breakpoint to display its location
 - Add, delete, enable, or disable breakpoints

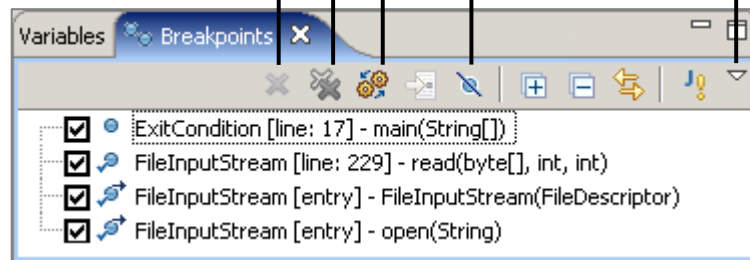
Add Java Exception breakpoint

Skip all breakpoints

Show breakpoints supported
by selected target

Remove all breakpoints

Remove selected breakpoints



© Copyright IBM Corporation 2011

Figure 8-8. Breakpoints view

WD154 / VD1542.0

Notes:

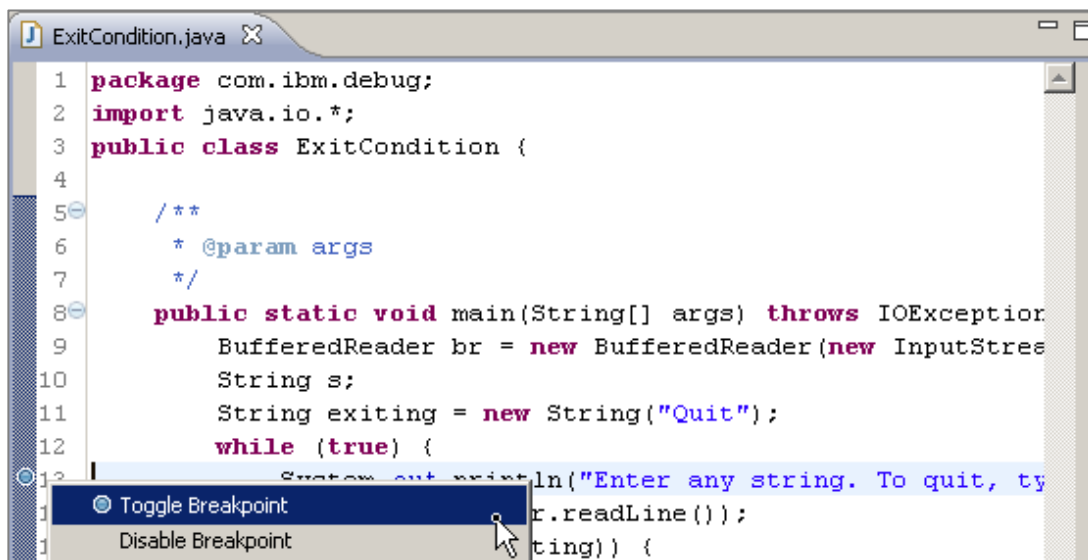
Java exception breakpoints suspend execution at the point where the exception is thrown. Execution can be suspended when exception is uncaught, caught or both.

Most of the toolbar icons on the Breakpoints view are self-explanatory. The **Show Breakpoints Supported by Selected Target** button deserves an explanation:

- Normally a list of all breakpoints (for all debug sessions) appears in the Breakpoints view, unless you use the filter by debug target action.
- To filter out breakpoints that are not related to the current debug session, click the Breakpoints view **Show Breakpoints Supported by Selected Target** push button.
- Alternatively, right-click in the view and enable the **Show Supported Breakpoints** pop-up menu item. (This menu item is enabled when there is a check mark to the left of it.)

Adding a breakpoint into a Java application

1. In the editor area, open the file where you want to add the breakpoint or directly to the left of the line where you want to add the breakpoint
2. Right-click the marker bar (vertical ruler)
3. Select **Toggle Breakpoint** from the pop-up menu



© Copyright IBM Corporation 2011

Figure 8-9. Adding a breakpoint into a Java application

WD154 / VD1542.0

Notes:

You can also double-click in the marker bar (toggle action).

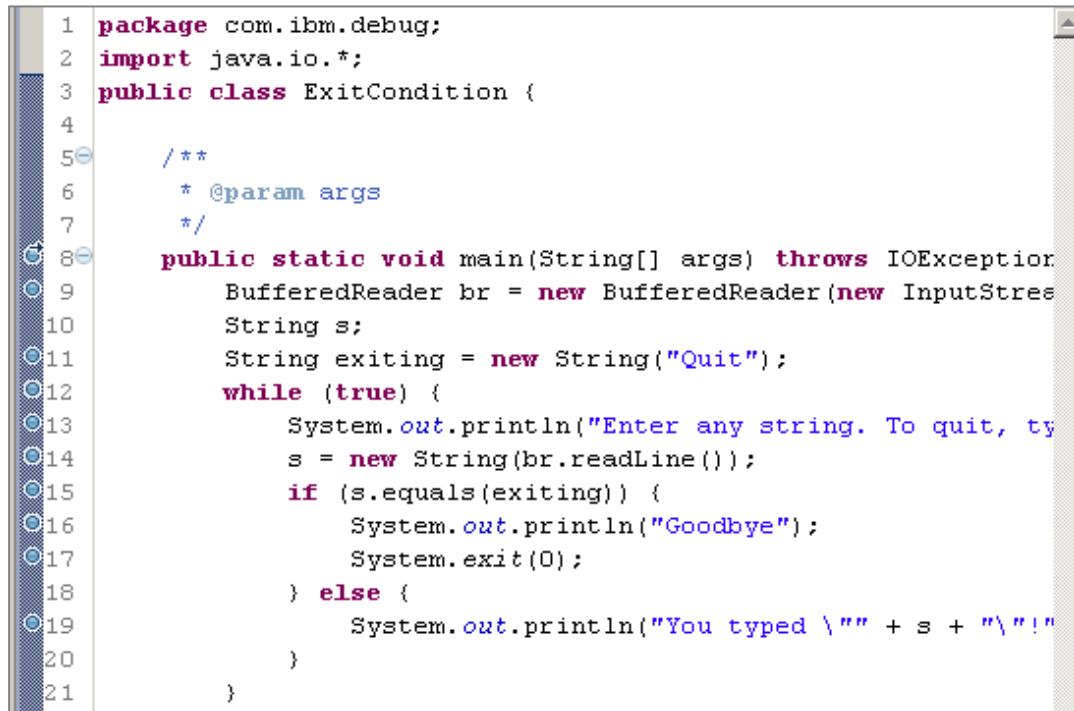
The debugger stops immediately before executing the line with the breakpoint.

Breakpoints can be disabled from the Breakpoints view or from the pop-up menu.

Java exception breakpoints can only be set from the Breakpoints view.

Where can breakpoints be set?

- Breakpoints are set on an executable line of a program



© Copyright IBM Corporation 2011

Figure 8-10. Where can breakpoints be set?

WD154 / VD1542.0

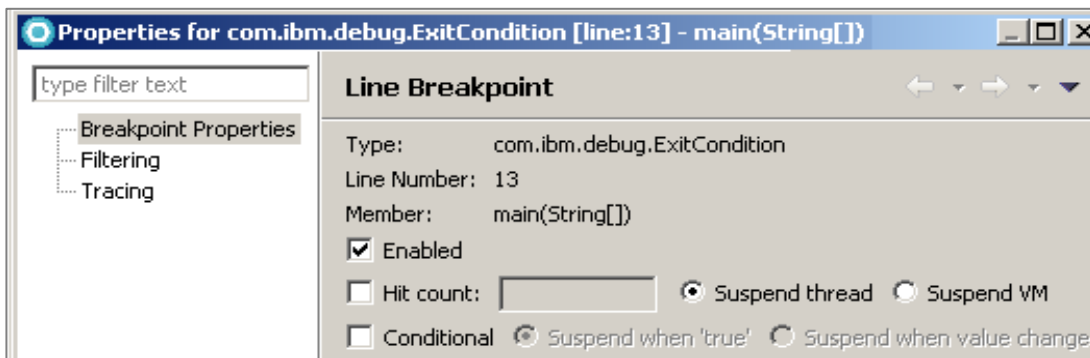
Notes:

Not every line of source in the editor can have a breakpoint. If you try to set a breakpoint on the else condition, it adds the breakpoint on the next line in the Java code. If you try to set one after all executable code (on one of the closing braces, for example) the attempt is ignored.

The slide shows all the executable lines of the program with breakpoints set.

Configuring breakpoint properties

- Breakpoint properties can be set in their **Properties** dialog
- Breakpoints can be enabled or disabled
 - Newly added breakpoints are enabled by default
- Hit count breakpoints:
 - The thread execution suspends when the breakpoint is hit for the n^{th} time
 - The breakpoint is disabled until either it is reenabled or its hit count is changed
- Conditional breakpoints:
 - Must evaluate to a boolean
 - Only invoked if condition evaluates to `true`



© Copyright IBM Corporation 2011

Figure 8-11. Configuring breakpoint properties

WD154 / VD1542.0

Notes:

To access the Properties dialog for a breakpoint, right-click a breakpoint in the Breakpoints view or the vertical ruler. If you are in the Breakpoints view, select **Properties** from the pop-up menu. If you right-clicked a breakpoint in the vertical ruler of a Java editor, select **Breakpoint Properties** from the pop-up menu.

Breakpoints can be enabled or disabled. Newly added breakpoints are enabled by default.

Hit count breakpoints:

- The thread execution suspends when the breakpoint is hit for the n^{th} time
- The breakpoint is disabled until either it is reenabled or its hit count is changed.
- The hit count must be a single positive integer.

Conditional breakpoints:

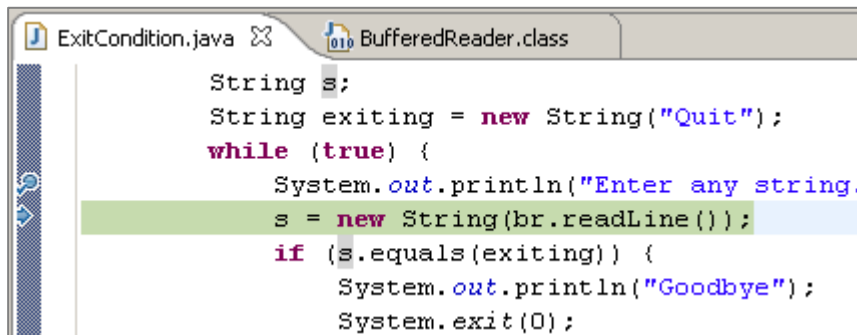
- The condition can be complex, and might suspend execution several times. Any expression is valid as long as it evaluates to a boolean:
 - `Name.equals(employee.getName())`

- `Double.compare(Math.random(), 0.5) == 0`
- `thisTemperature != previousTemperature`
- Execution suspends either when the condition is true or when the value changes, depending on which option button has been selected.

The breakpoint suspend property determines whether the thread or the entire VM suspends when the breakpoint is encountered.

Source view

- The current line is highlighted in the source view



© Copyright IBM Corporation 2011

Figure 8-12. Source view

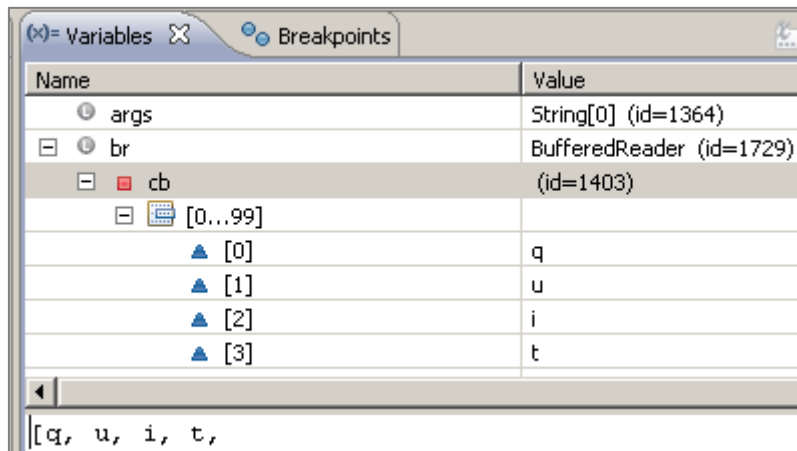
WD154 / VD1542.0

Notes:

The Source view of the Debug perspective indicates the current line of the program that is being debugged.

Variables view

- Examine the contents of variables when a thread suspends
 - Top stack frame of the thread is automatically selected
 - Visible variables of the selected stack frame are displayed



© Copyright IBM Corporation 2011

Figure 8-13. Variables view

WD154 / VD1542.0

Notes:

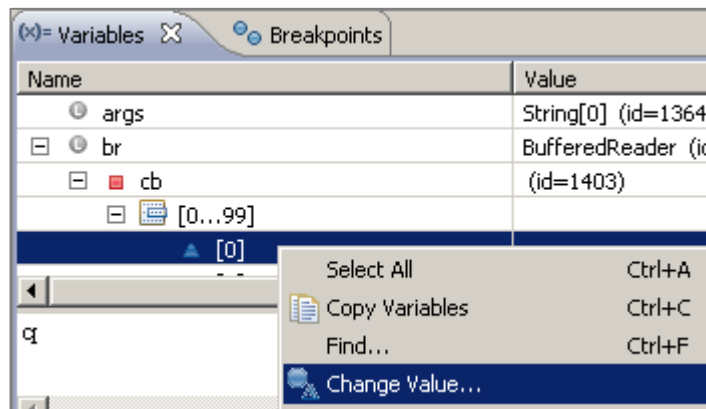
When a thread is suspended, the top stack frame of the thread is automatically selected, and all visible variables are shown in the Variables view.

A visible variable is one that has been initialized.

Variables such as arrays can be expanded to reveal the values of all of their elements.

Changing variable values

- Use the **Variables** view to change the value of a variable
- Right-click a variable and select **Change Value** from the pop-up menu



© Copyright IBM Corporation 2011

Figure 8-14. Changing variable values

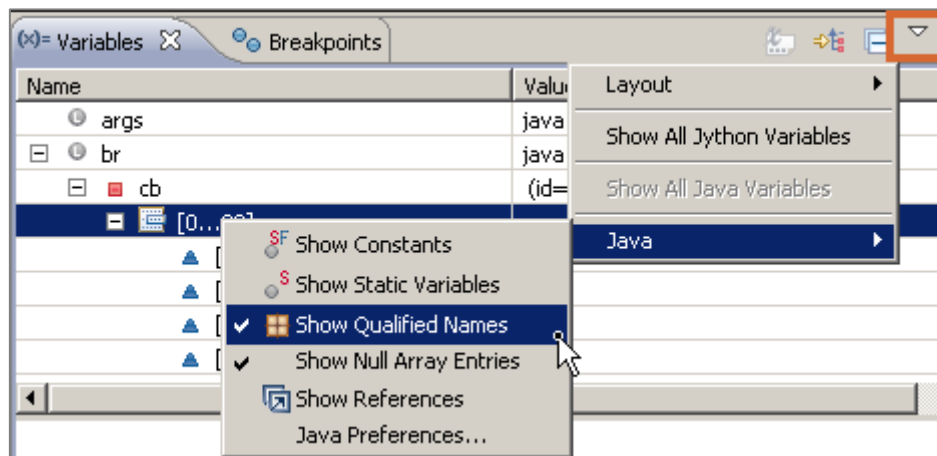
WD154 / VD1542.0

Notes:

You can use the Variables view to change the values of program variables.

Using the View menu with variables

- Use the view drop-down menu
- **Show Qualified Names** adds the fully qualified type for the variable selected



© Copyright IBM Corporation 2011

Figure 8-15. Using the View menu with variables

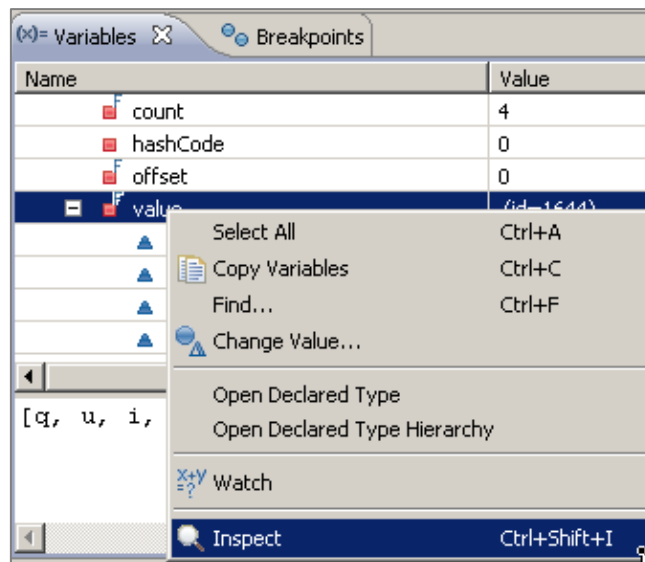
WD154 / VD1542.0

Notes:

If **Show Qualified Names** is checked, then String is shown as `java.lang.String` and `BufferedReader` is shown as `java.io.BufferedReader`.

Data in the Expressions view (1 of 2)

- Data from a scrapbook page or a stack frame of a suspended thread can be inspected in the **Expressions** view
- The view is not initially open in the Debug perspective
 - It opens automatically when an item is added to the view



© Copyright IBM Corporation 2011

Figure 8-16. Data in the Expressions view (1 of 2)

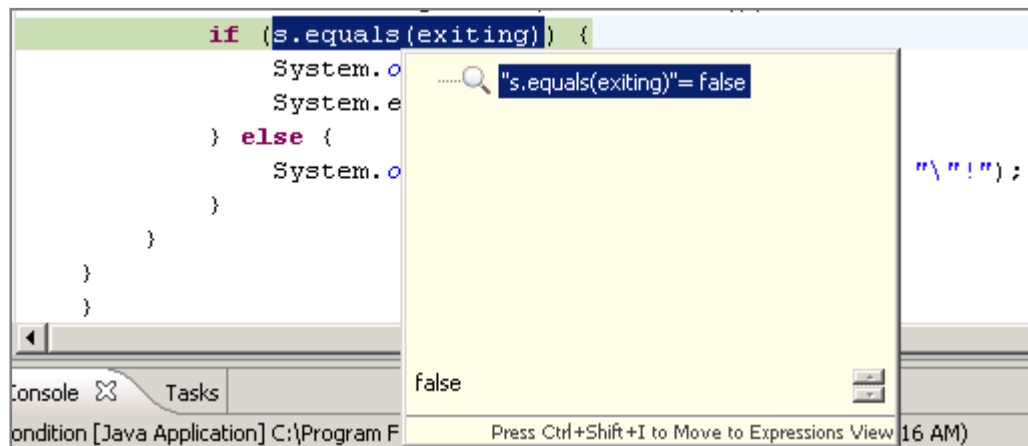
WD154 / VD1542.0

Notes:

The Expressions view is opened if you right-click an item and choose **Inspect** from the pop-up menu.

Data in the Expressions view (2 of 2)

- To inspect data in this view, select a variable in the source code and press Ctrl+Shift+I twice
 - The first time opens a window with the expression
 - The second time moves the contents of the window to the Expressions view



© Copyright IBM Corporation 2011

Figure 8-17. Data in the Expressions view (2 of 2)

WD154 / VD1542.0

Notes:

After selecting a statement in the Source view, you can move it to the Expressions view by pressing Ctrl + Shift + I.

The result of evaluating the expression is displayed in the Expressions view.

Unit summary

Having completed this unit, you should be able to:

- Use the Debug perspective to debug Java applications
- Step through code using the Debug view toolbar
- Add and configure breakpoints in a Java application
- View and change variables during debugging
- Execute and inspect expressions during debugging

© Copyright IBM Corporation 2011

Figure 8-18. Unit summary

WD154 / VD1542.0

Notes:

Checkpoint questions

1. In the middle of debugging a method, you decide you want to skip ahead to the point where control is handed back to the caller. What can you do?
2. What information is shown in the Debug view?
3. How do you know that a breakpoint has some condition set?
4. On what lines of code are you able to enter a breakpoint?

© Copyright IBM Corporation 2011

Figure 8-19. Checkpoint questions

WD154 / VD1542.0

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.

Checkpoint answers

1. Click the **Step Return** button, or press F7.
2. Stack frames, process threads, terminated launches.
3. There is a question mark overlay on the icon.
4. Executable lines.

© Copyright IBM Corporation 2011

Figure 8-20. Checkpoint answers

WD154 / VD1542.0

Notes:

Exercise

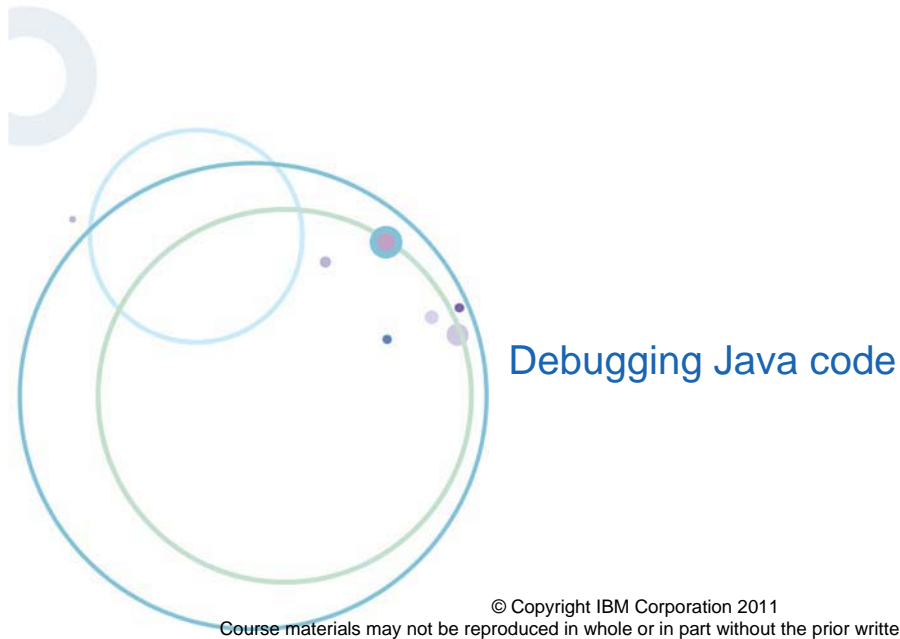


Figure 8-21. Exercise

WD154 / VD1542.0

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Run an application in debug mode
- Set breakpoints to view and modify variables and expressions
- Step through the code using the debugger to quickly track down problems and bugs

© Copyright IBM Corporation 2011

Figure 8-22. Exercise objectives

WD154 / VD1542.0

Notes: