DEBUGGING JAVA APPLICATION USING ECLIPSE

1.1 Objective

- What is debugging and why it is used?
- Using the debugger
- Starting the debugger
- Setting breakpoints
- Stepping through the code
- Inspecting variables and expressions
- Hot code replace
- What if it is java web application?

1.2 Content

1.2.1 What is debugging and why it is used?

What Is Debugging?
Programming is error-prone. For whimsical reasons, programming errors are called bugs and the process of tracking them down is called debugging.

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming. In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

Debugging is also like an experimental science. Once you have an idea about what is going wrong, you modify your program and try again. If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one.

For some people, programming and debugging are the same thing. That is, programming is the process of gradually debugging a program until it does what you want. The idea is that you should start with a program that does something and make small modifications, debugging them as you go, so that you always have a working program.

Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors. It is useful to distinguish between them in order to track them down more quickly.

1.2.1.1 Syntax Error

Java can only execute a program if the syntax is correct; otherwise, the compiler displays an error message. Syntax refers to the structure of a program and the rules about that structure.For example, parentheses have to come in matching pairs, so (1 + 2) is legal, but 8) is a syntax error.
In English readers can tolerate most syntax errors, which is why we can read the poetry of e. e. cummings without spewing error messages. Java is not so forgiving. If there is a single syntax error anywhere in your program, Java will display an error message and you will not be able to run your program.

1.2.1.2 Runtime Error

The second type of error is a runtime error, so called because the error does not appear until after the program has started running. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

1.2.1.3 Semantic Error

The third type of error is the semantic error. If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else. Specifically, it will do what you told it to do.

The problem is that the program you wrote is not the program you wanted to write. The meaning of the program (its semantics) is wrong. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

1.2.3 Eclipse Debugger

The Eclipse Java tools include a world-class debugger.
To debug your code, you run it in debug mode.You don't have to change your code or recompile it. in any way.But wait what if eclipse debugger is not there!! How do you debug your code!!
How do you check values of varibales and expressions in your programme?

Probably you are thinking that you will each time write "System.out.println(nameOfVariable);"
Don't you think it is very boring!! Now as eclipse debugger is there then you first keep a break point at place where you want to check a particular variable or expression.
But question here is what is break point!!!

1.2.3.1 Break Point

Depending on the structure of your code, you may want to stop the debugger in the main() method.Generally the debugger is stopped at the position where the developeer might suspect the origin of the bug or an error.

You can also create a debug configuration.Which accepts many parameters to configure your debug settings.But generally we dont use it,we use default configuration .

To read more please visit www.tutorialspoint.com/eclipse/eclipse_debug_configuration.htm

F5 - Step by Step debugging

F6 - Skips loops and Subroutines

F7 - Skips the loop or subroutine and returns to the last cursor point.

F8 - Execute and come out of debugging

1.2.3.2 Types of BreakPoints:

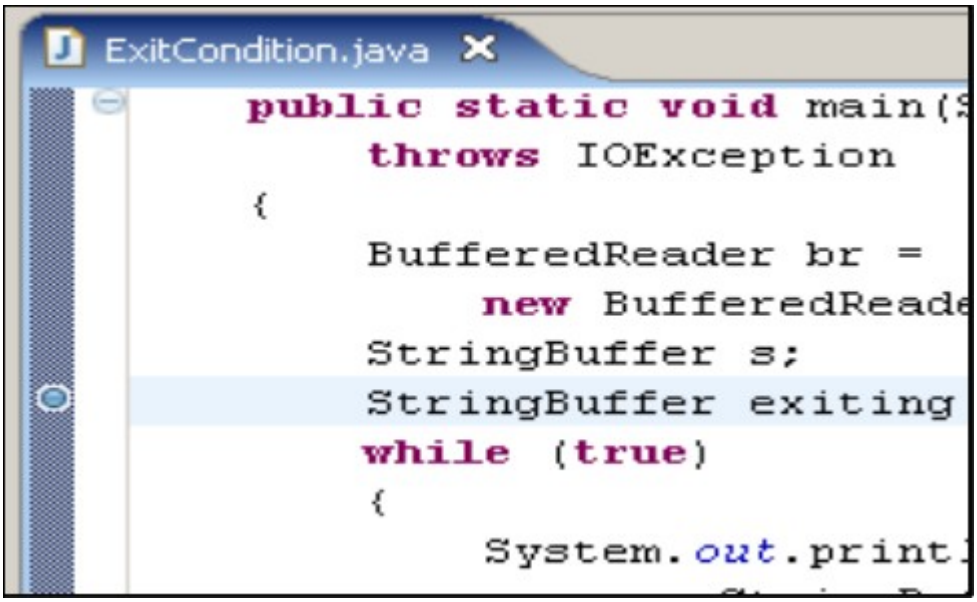
The simplest kind of breakpoint is a line breakpoint.

To create one, double-click in the margin next to a line of code.

Double-click the icon to remove it.

A method breakpoint stops when the debugger enters or exits a particular method.

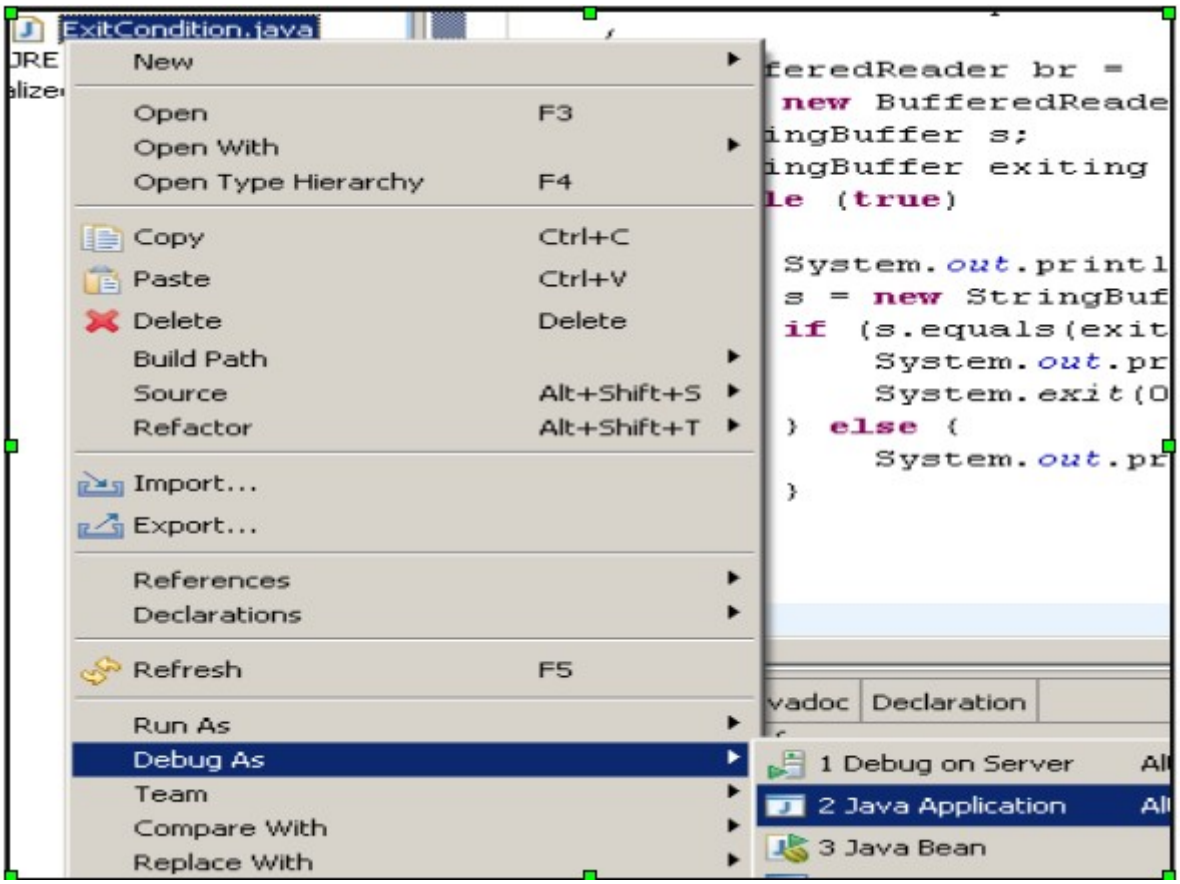
You can set an exception breakpoint on a particular Java exception (caught or uncaught).



Now after understanding this that what is break point you follow the below mentioned step to run a debugger in eclipse!!

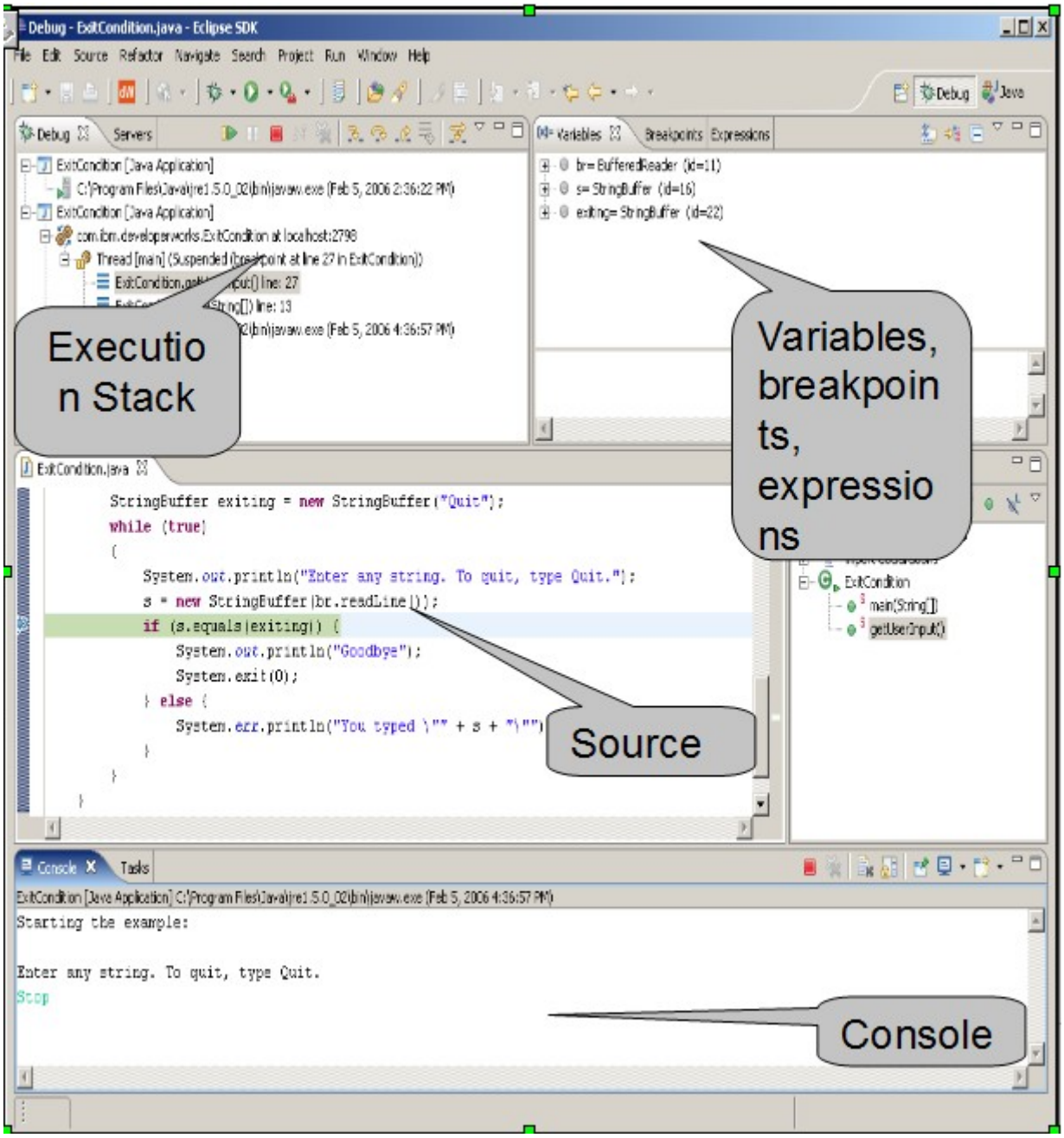
1.2.3.3 How to run debugger in eclipse:

Right-click on the Java file, then select Debug As✳Java Application (instead of Run As✳Java Application).



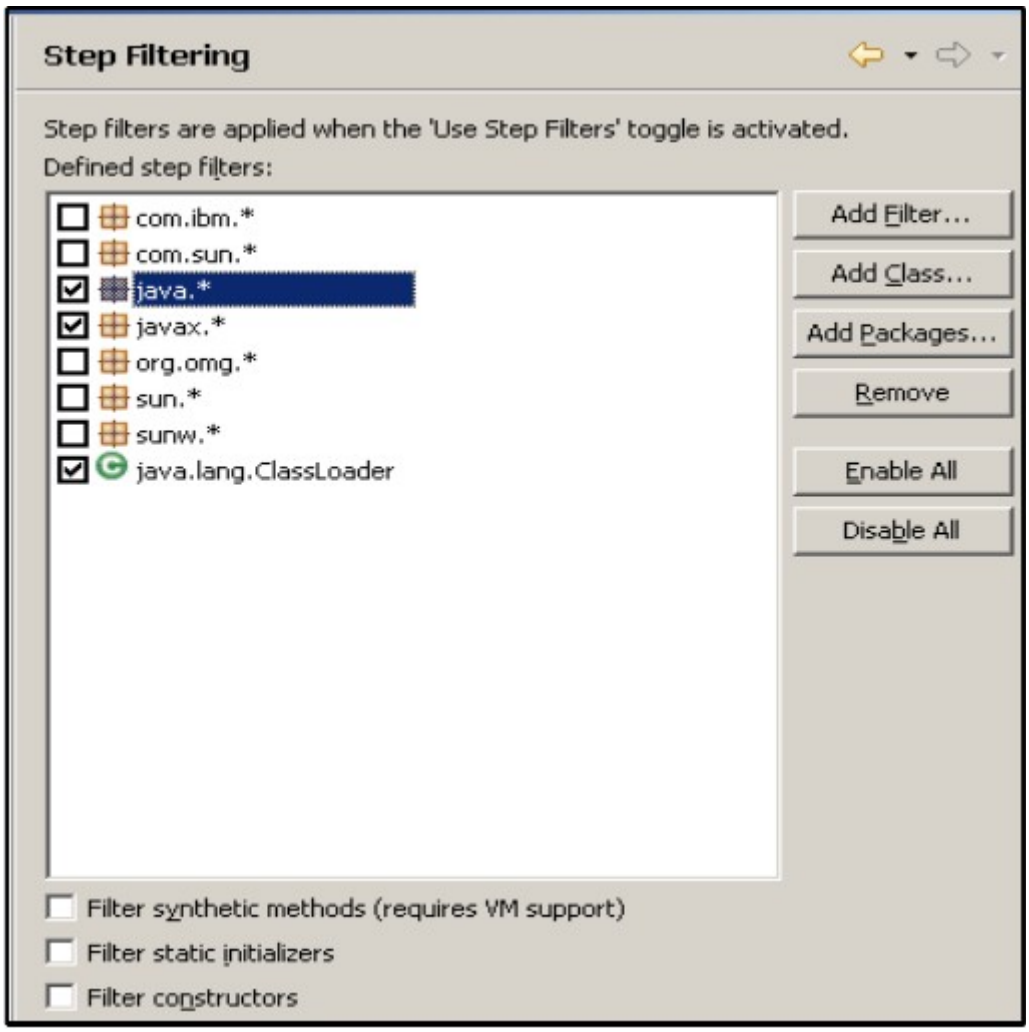
Once you will do this and if you have included debug point and your programme is going to that debug point then eclipse will ask you that "Would you like to open debug prospective?" say "yes" and you will able to see below mentioned window!

1.2.3.4 The debug perspective



1.2.4 Step filtering:

You can set filters in your debugging session. This tells the debugger not to stop on certain lines of code.If you combine filters with the Step with Filters button, each step with the debugger (step into, step over or step return) will skip the filtered lines of code.

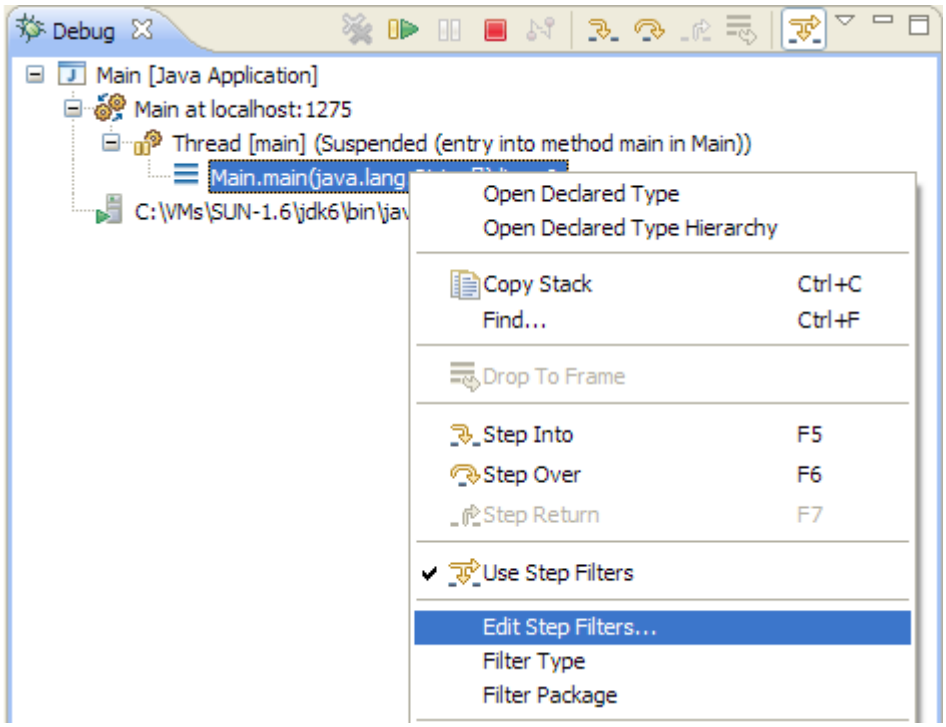


1.2.4.1 How to open step filtering

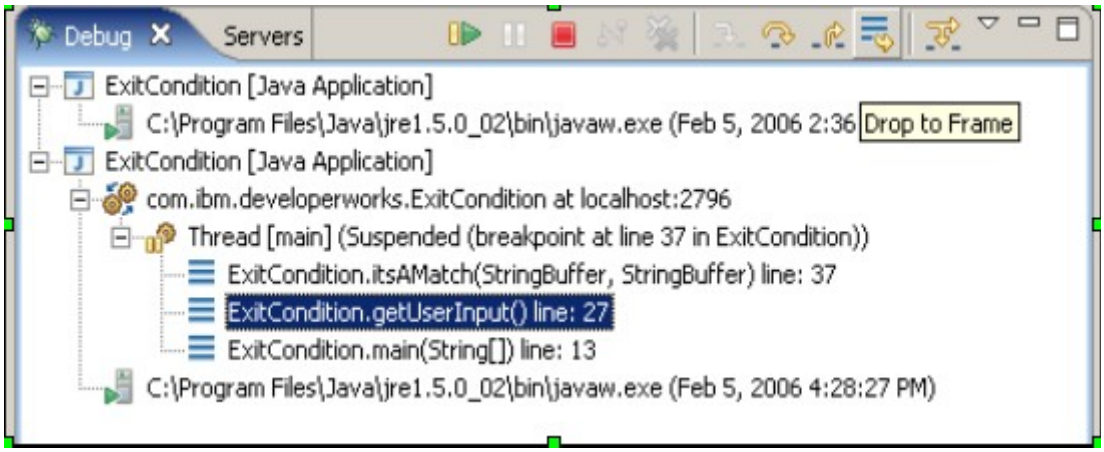
1.Right Click on any stack Frame.

2. Choose Edit Step Filters

3. You will be taken to the same window shown in the previous page.



1.2.5 Stack Frames:



Whenever a thread invokes a method, that invocation is added to the stack.

main() called getUserInput(), which called itsAMatch().

The Drop to Frame button lets you go backwards to the point at which a thread invoked a particular method.

Here we can clearly see that the stack are arranged in order of calling.

We can clearly identify the order of called methods and open a particular stack to view the details.

1.2.6 Inspecting variables and expressions:

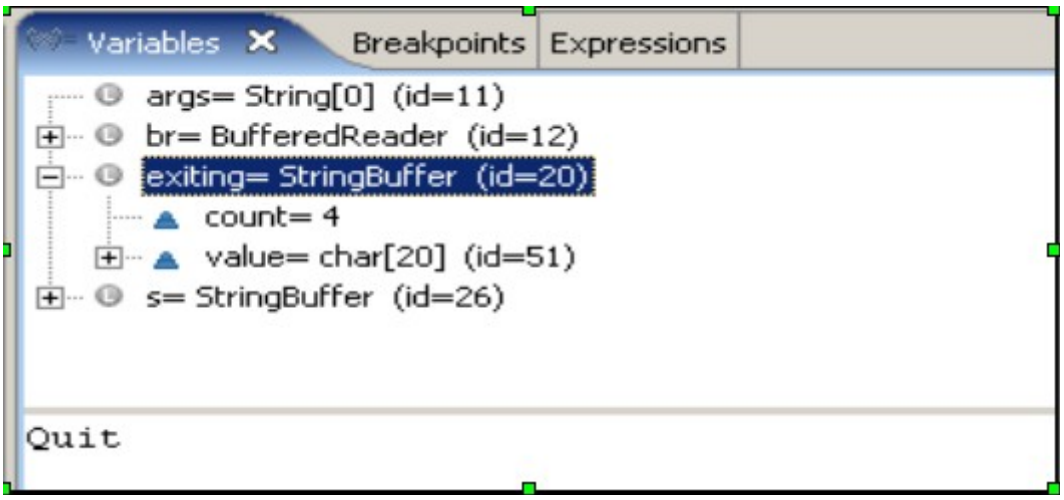
The debugger has a variables view that shows all the variables currently in scope.

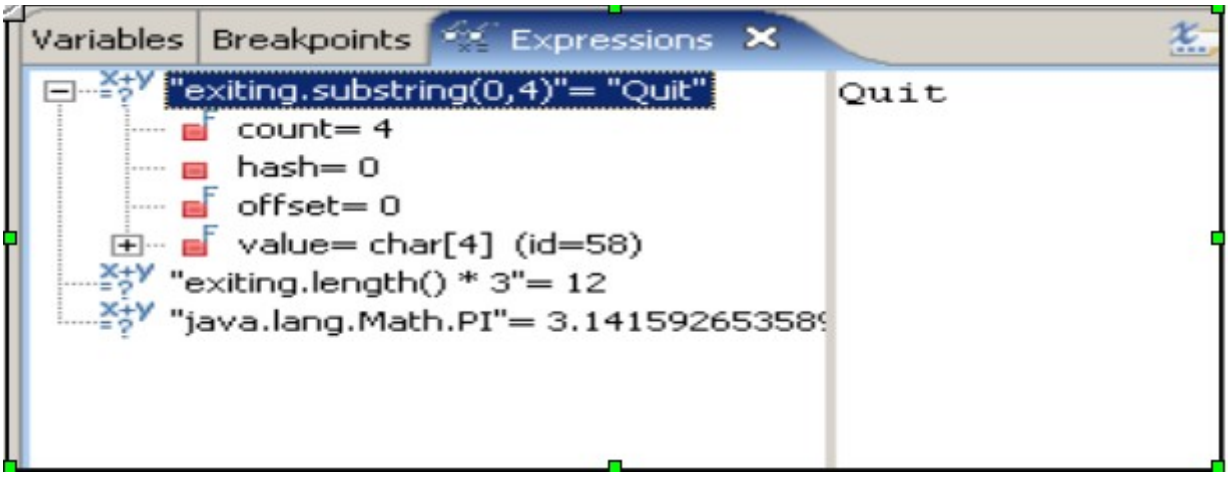
You can change the values of those variables if you want.

There's also an expressions view that lets you evaluate expressions.

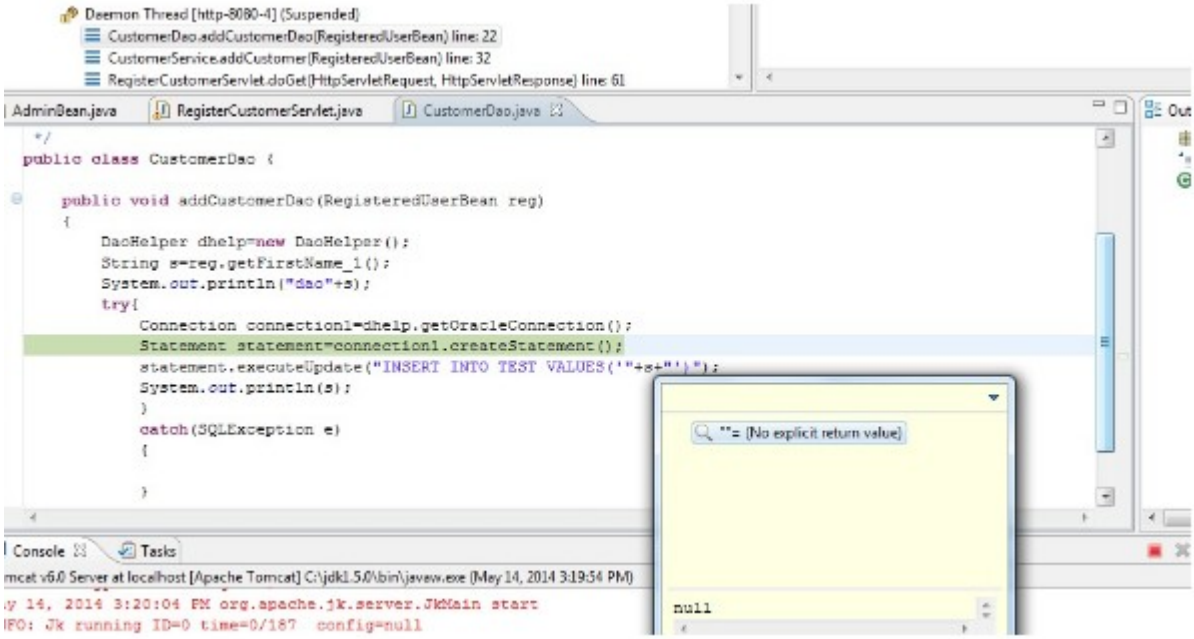
These typically involve some variable in scope, but can be any Java language expression.

You can use static methods of a class not used in your code, for example.

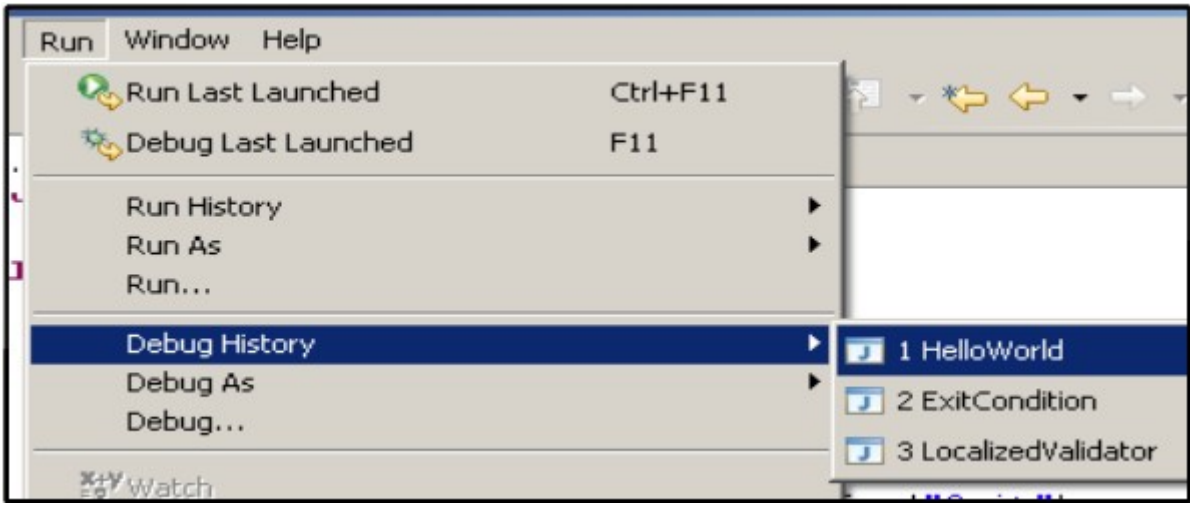




Press ctrl+shift+i to view dynamic values at runtime



1.2.7 Re-running the debugger:



Once you've run your code with the debugger, a reference to it appears in the Run menu. You can click your program's name in the Debug History menu to debug it again.
Debug Last Launched (F11) does the same thing.
Debug history simply shows what you have debugged in the past and provides shortcuts to Debug them again.

1.3 Hot code replace

One of the coolest features of the Java debugger is the ability to replace code currently running in the debugger.You don't have to restart the debugger or recreate the state of your program when you changed the code.
How it works:

The JVM you're using must support hot code replace (most JVMs 1.4.x and later do)
You can't do anything that changes the "signature" of a class (add or remove methods or instance variables, move it up and down in the class hierarchy, and so forth)
You typically can't change the main() method
You typically go back up the stack to whatever called the code you changed, but that's probably what you wanted to do anyway

Here's the console:

Starting the example:
Enter any string. To quit, type Quit.You typed "Quit"
Enter any string. To quit, type Quit.You typed "Quit"
Enter any string. To quit, type Quit.

The program should exit when you type quit.But it does not.
Hence it has an error.
Our code never exits, regardless of what we type. We'll run this through the debugger …

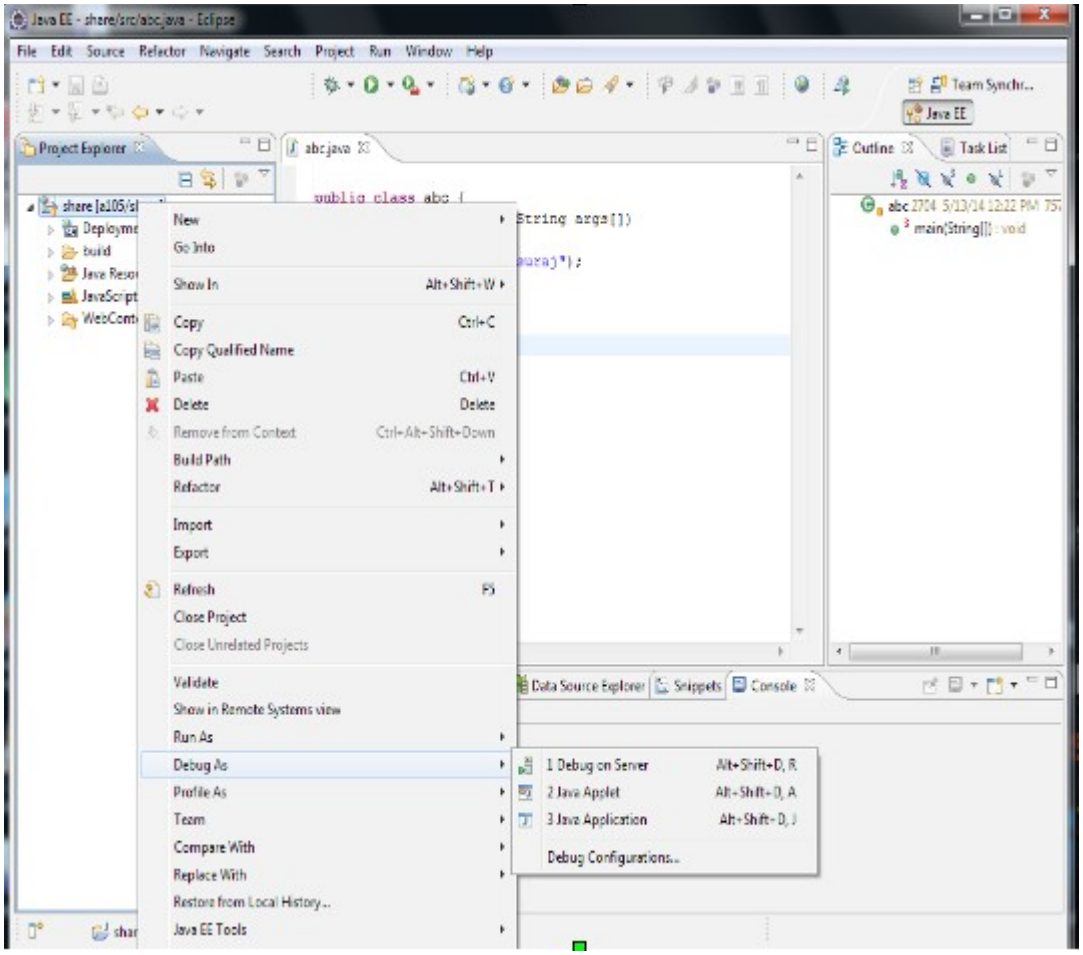
We're stepping through the code when we realize the problem. This line: s.equals(exiting)should be: s.toString().equals(exiting.toString())

With hot code replace, we can fix the code without restarting the debugger.And adding the .toString() method while debugging .

As the debugging session continues, we can verify that our code change fixed the problem.
Hence in hotcode fixes are done while debugging.

1.4 What if it is java web application?

The Eclipse Java tools include a world-class debugger.
To debug your code, you run it in debug mode.You don't have to change your code or recompile it in any way.Right-click on the Dynamic web project or any file , Debug On Server (instead of Run As✳ Run on server).



This is the only change which is there in java web application for debugging!!
Rest all step will remain as it is as you have just seen!!