

## Table of Contents

- 1. Overview**
  - 1.1. What is debugging?
  - 1.2. Debugging support in Eclipse
- 2. Prerequisites**
- 3. Debugging in Eclipse**
  - 3.1. Setting Breakpoints
  - 3.2. Starting the Debugger
  - 3.3. Controlling the program execution
  - 3.4. Breakpoints view and deactivating breakpoints
  - 3.5. Evaluating variables in the debugger
  - 3.6. Changing variable assignments in the debugger
  - 3.7. Controlling the display of the variables with Detail Formatter
- 4. Advanced Debugging**
  - 4.1. Breakpoint properties
  - 4.2. Watchpoint
  - 4.3. Exception breakpoints
  - 4.4. Method breakpoint
  - 4.5. Breakpoints for loading classes
  - 4.6. Step Filter
  - 4.7. Hit Count
  - 4.8. Remote debugging
  - 4.9. Drop to frame
- 5. Exercise: Create Project for debugging**
  - 5.1. Create Project
  - 5.2. Debugging

### **1. Overview**

#### **1.1. What is debugging?**

Debugging allows you to run a program interactively while watching the source code and the variables during the execution.

By breakpoints in the source code you specify where the execution of the program should stop. To stop the execution only if a field is read or modified, you can specify watchpoints .

Breakpoints and watchpoints can be summarized as stop points.

Once the program is stopped you can investigate variables, change their content, etc.

#### **1.2. Debugging support in Eclipse**



## Table of Contents

### 1. Overview

- 1.1. What is debugging?
- 1.2. Debugging support in Eclipse

### 2. Prerequisites

### 3. Debugging in Eclipse

- 3.1. Setting Breakpoints
- 3.2. Starting the Debugger
- 3.3. Controlling the program execution
- 3.4. Breakpoints view and deactivating breakpoints
- 3.5. Evaluating variables in the debugger
- 3.6. Changing variable assignments in the debugger
- 3.7. Controlling the display of the variables with Detail Formatter

### 4. Advanced Debugging

- 4.1. Breakpoint properties
- 4.2. Watchpoint
- 4.3. Exception breakpoints
- 4.4. Method breakpoint
- 4.5. Breakpoints for loading classes
- 4.6. Step Filter
- 4.7. Hit Count
- 4.8. Remote debugging
- 4.9. Drop to frame

### 5. Exercise: Create Project for debugging

- 5.1. Create Project
- 5.2. Debugging

### 1. Overview

#### 1.1. What is debugging?

Debugging allows you to run a program interactively while watching the source code and the variables during the execution.

By breakpoints in the source code you specify where the execution of the program should stop. To stop the execution only if a field is read or modified, you can specify watchpoints .

Breakpoints and watchpoints can be summarized as stop points.

Once the program is stopped you can investigate variables, change their content, etc.

#### 1.2. Debugging support in Eclipse

Eclipse allows you to start a Java program in Debug mode.

Eclipse has a special Debug perspective which gives you a preconfigured set of views. In this perspective you control the execution process of your program and can investigate the state of the variables.

## 2. Prerequisites

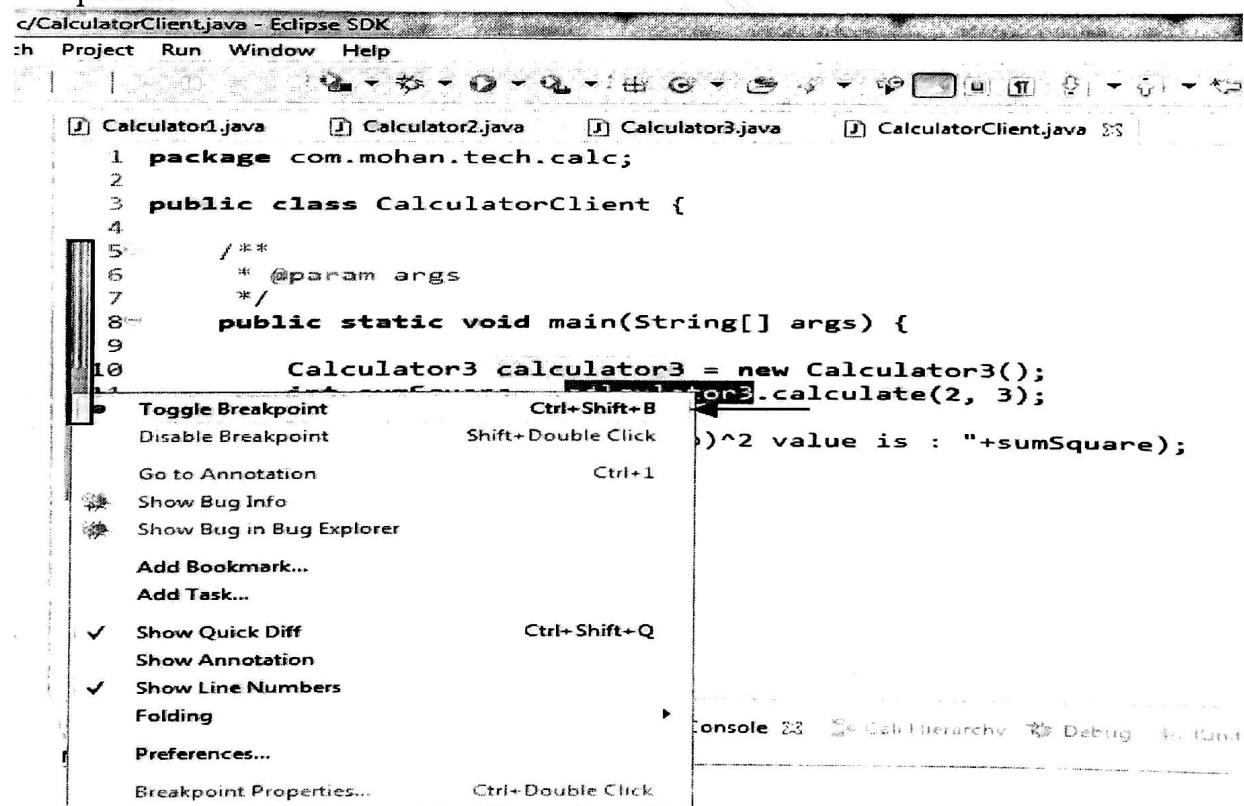
The following assumes you know how to develop simple standard Java programs. This article will focus on how to debug Java applications in Eclipse.

The installation and usage of Eclipse as Java IDE is described in Eclipse Java IDE Tutorial.

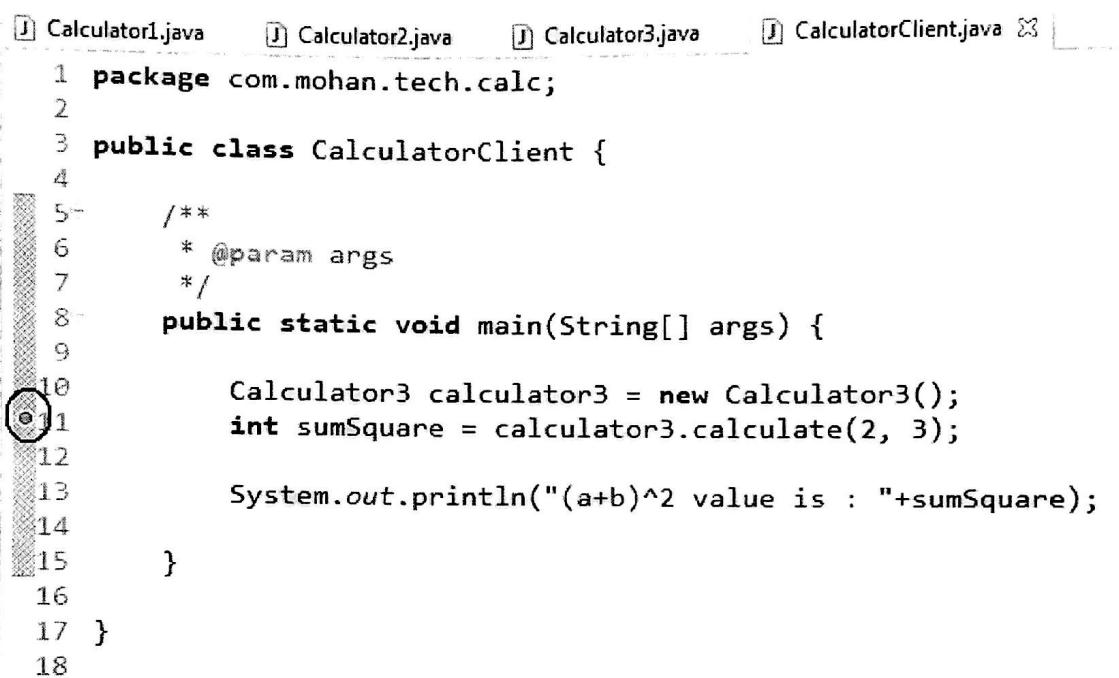
## 3. Debugging in Eclipse

### 3.1. Setting Breakpoints

To set breakpoints in your source code right-click in the small left margin in your source code editor and select Toggle Breakpoint. Alternatively you can double-click on this position.



For example in the following screen shot we set a breakpoint on the line  
int sumSquare = calculator3.calculate(2, 3);



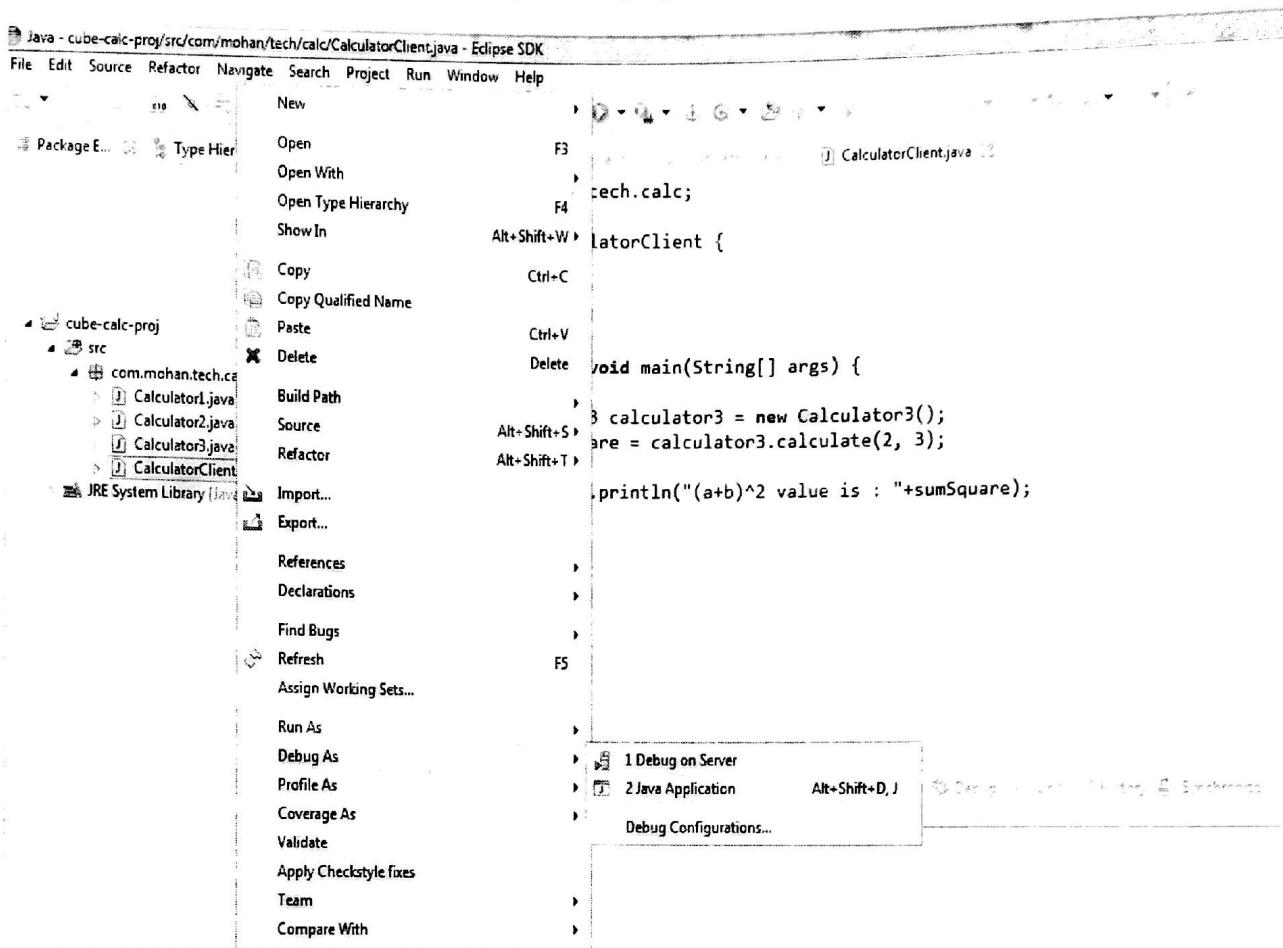
The screenshot shows a Java code editor with several tabs at the top: Calculator1.java, Calculator2.java, Calculator3.java, and CalculatorClient.java. The CalculatorClient.java tab is active. The code is as follows:

```
1 package com.mohan.tech.calc;
2
3 public class CalculatorClient {
4
5     /**
6      * @param args
7      */
8     public static void main(String[] args) {
9
10        Calculator3 calculator3 = new Calculator3();
11        int sumSquare = calculator3.calculate(2, 3);
12
13        System.out.println("(a+b)^2 value is : "+sumSquare);
14
15    }
16
17 }
18
```

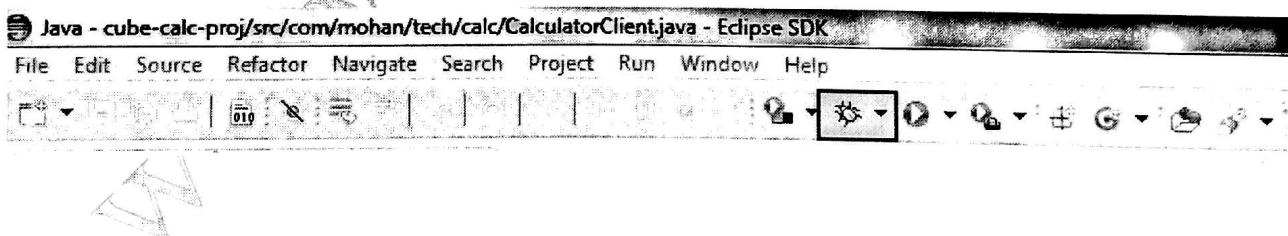
A circular selection mark highlights the line number 11, which contains the statement `int sumSquare = calculator3.calculate(2, 3);`. This indicates that a breakpoint has been set on this line.

### 3.2. Starting the Debugger

To debug your application, select a Java file which can be executed, right-click on it and select Debug As → Java Application.

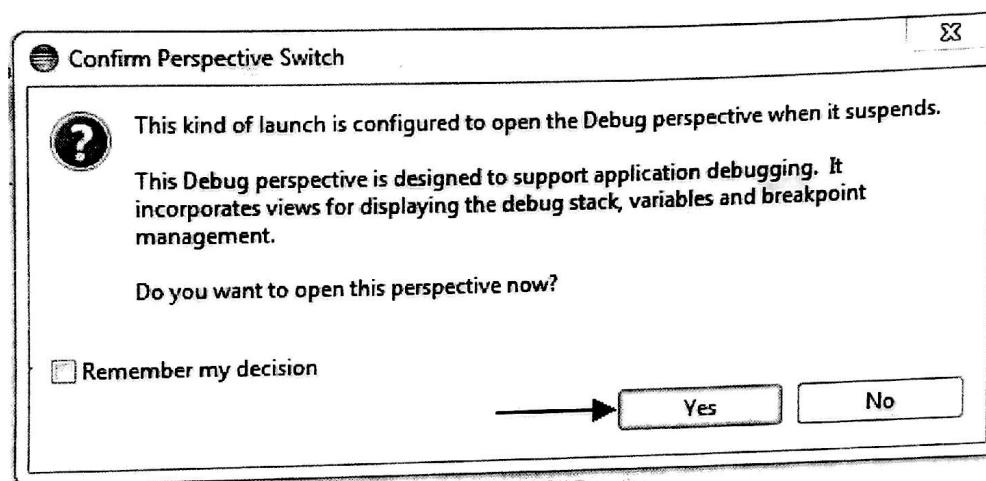


After you have started the application once via the context menu, you can use the created launch configuration again via the Debug button in the Eclipse toolbar.



If you have not defined any breakpoints, this will run your program as normal. To debug the program you need to define breakpoints.

If you start the debugger, Eclipse asks you if you want to switch to the Debug perspective once a stop point is reached. Answer Yes in the corresponding dialog.



Afterwards Eclipse opens this perspective, which looks similar to the following screenshot.

#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
[www.Sreenutech.com](http://www.Sreenutech.com)



```
CalculatorClient.java  com.mohan.tech.calc  CalculatorClient.java
1 package com.mohan.tech.calc;
2
3 public class CalculatorClient {
4
5     /**
6      * @param args
7      */
8     public static void main(String[] args) {
9
10        Calculator3 calculator3 = new Calculator3();
11        int sumSquare = calculator3.calculate(2, 3);
12
13        System.out.println("(a+b)^2 value is : "+sumSquare);
14    }
15
16 }
17 }
```

### 3.3. Controlling the program execution

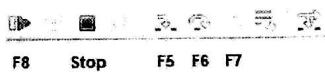
Eclipse provides buttons in the toolbar for controlling the execution of the program you are debugging. Typically it is easier to use the corresponding keys to control this execution.

You can use the F5, F6, F7 and F8 key to step through your coding. The meaning of these keys is explained in the following table.

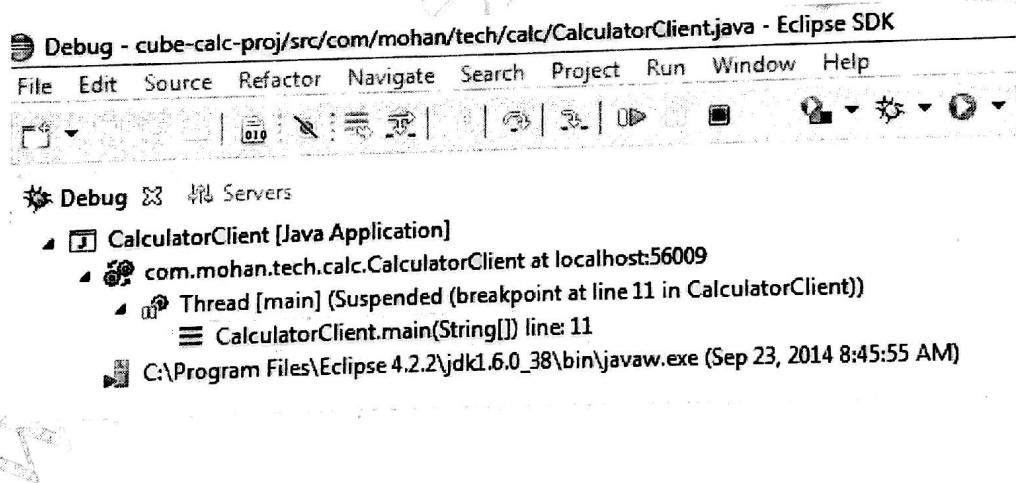
**Table 1. Debugging key bindings / shortcuts**

Key	Description
F5	Executes the currently selected line and goes to the next line in your program. If the selected line is a method call the debugger steps into the associated code.
F6	F6 steps over the call, i.e. it executes a method without stepping into it in the debugger.
F7	F7 steps out to the caller of the currently executed method. This finishes the execution of the current method and returns to the caller of this method.
F8	F8 tells the Eclipse debugger to resume the execution of the program code until it reaches the next breakpoint or watchpoint.

The following picture displays the buttons and their related keyboard shortcuts.



The call stack shows the parts of the program which are currently executed and how they relate to each other. The current stack is displayed in the Debug view.



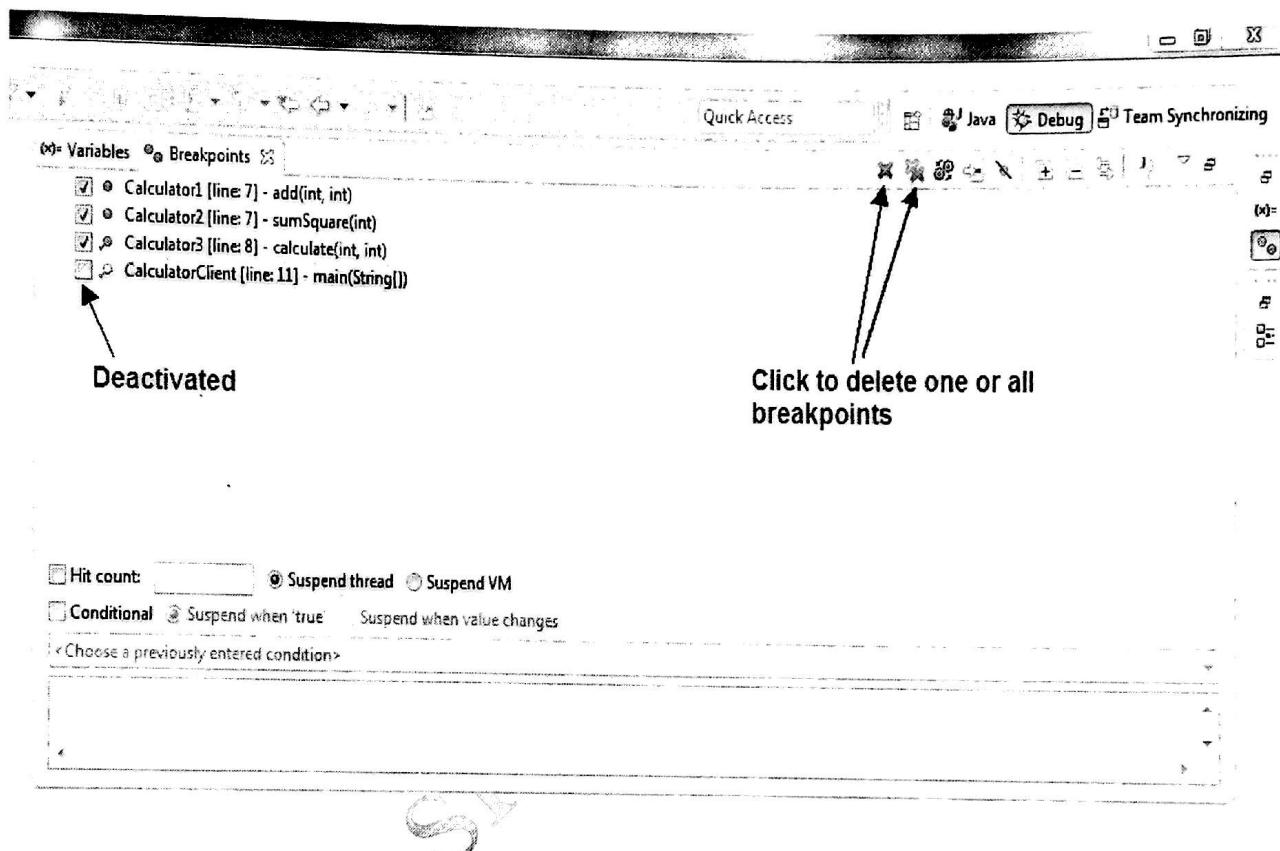
### 3.4. Breakpoints view and deactivating breakpoints

The Breakpoints view allows you to delete and deactivate stop points, i.e. breakpoints and

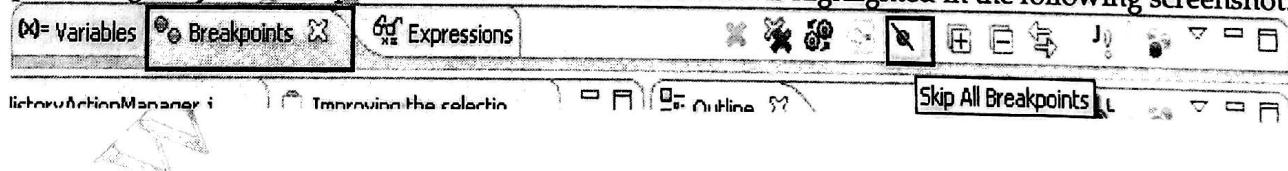
#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
 Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
[www.Sreenutech.com](http://www.Sreenutech.com)

watchpoints and to modify their properties.

To deactivate a breakpoint, remove the corresponding checkbox in the Breakpoints view . To delete it you can use the corresponding buttons in the view toolbar. These options are depicted in the following screenshot.



If you want to deactivate all your breakpoints you can press the Skip all breakpoints button. If you press it again, your breakpoints are reactivated. This button is highlighted in the following screenshot.



### 3.5. Evaluating variables in the debugger

The Variables view displays fields and local variables from the current executing stack. Please note you need to run the debugger to see the variables in this view .

# SREENU TECHNOLOGIES

(x) Variables Breakpoints

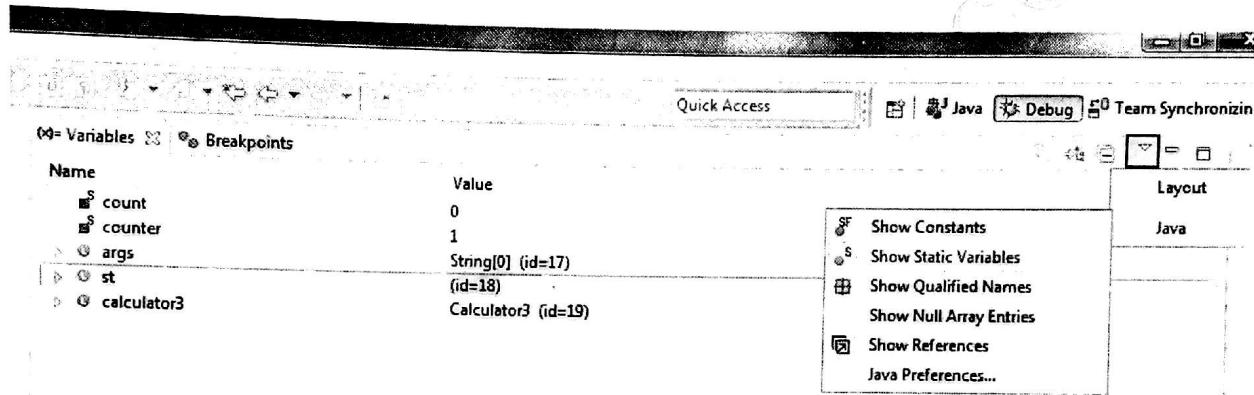
## Name

- this
- a
- b
- cal1
- cal2

## Value

- Calculator3 (id=17)
- 2
- 3
- Calculator1 (id=24)
- Calculator2 (id=25)

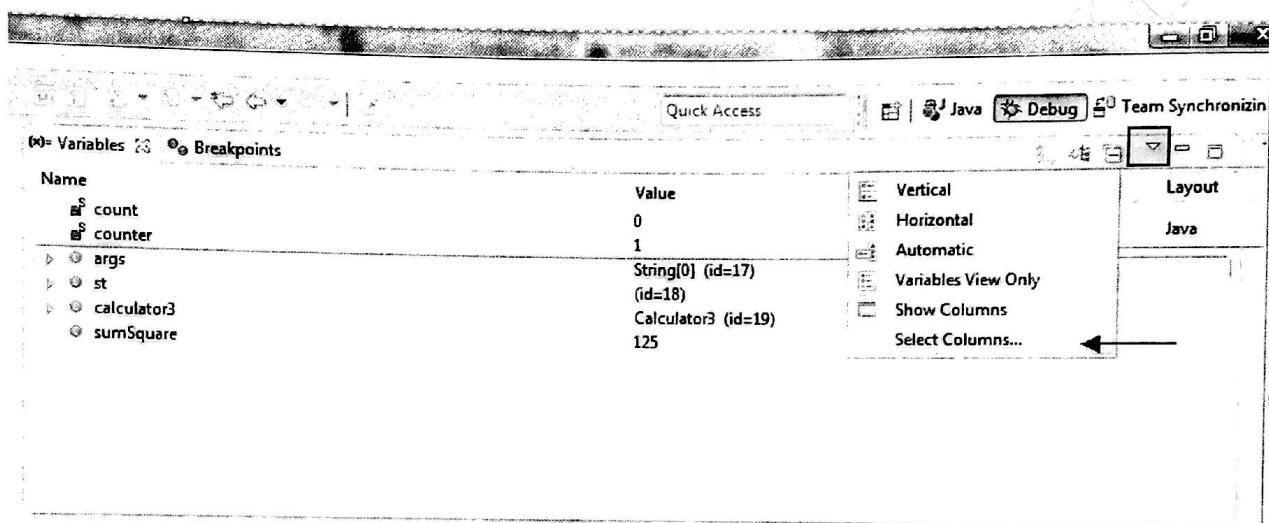
Use the drop-down menu to display static variables.



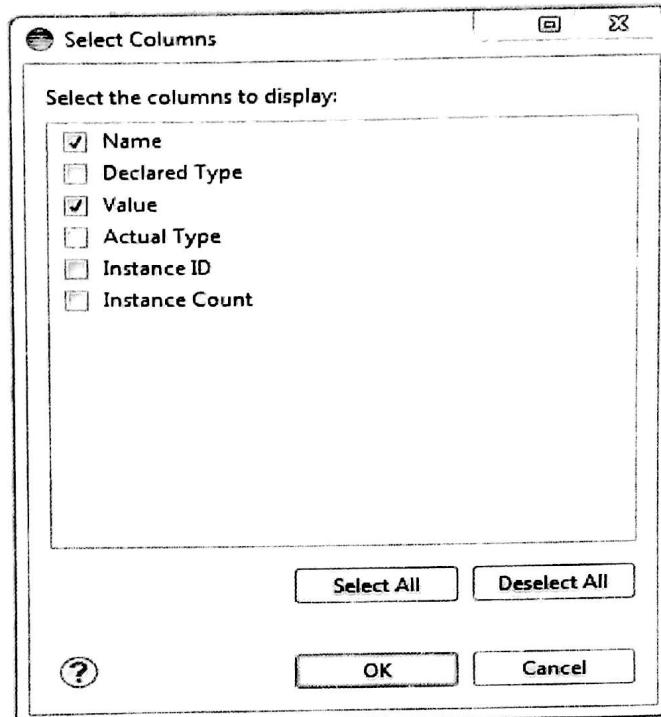
Via the drop-down menu of the Variables view you can customize the displayed columns. For example, you can show the actual type of each variable declaration. For this select Layout → Select Columns... → Type.

Via the drop-down menu of the Variables view you can customize the displayed columns. For example, you can show the actual type of each variable declaration. For this select Layout → Select Columns... → Type.

Via the drop-down menu of the Variables view you can customize the displayed columns. For example, you can show the actual type of each variable declaration. For this select Layout → Select Columns... → Type.

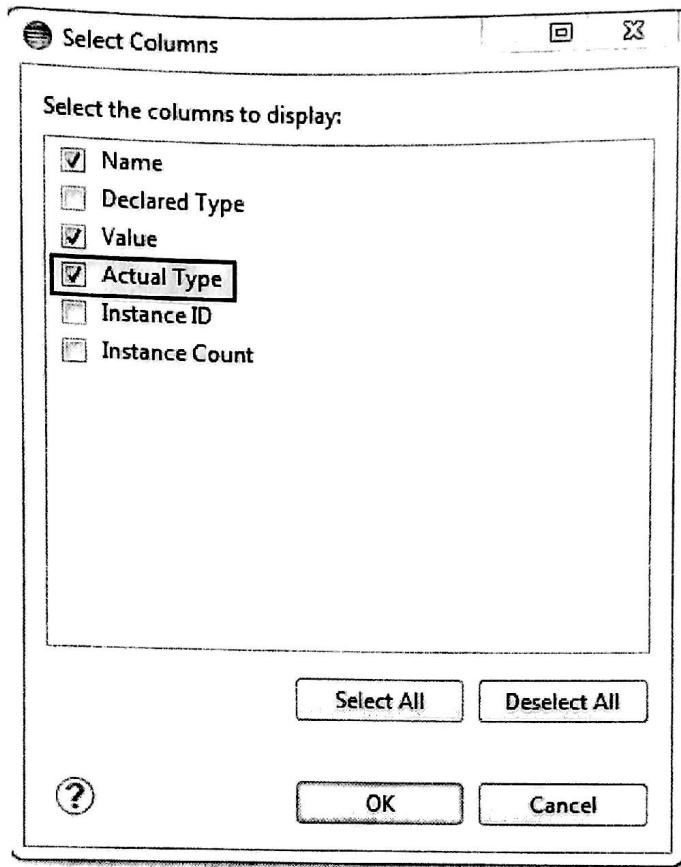


When selecting upon Select Columns..., will get one 'Select Columns' window like below. Here you can choose what values required.

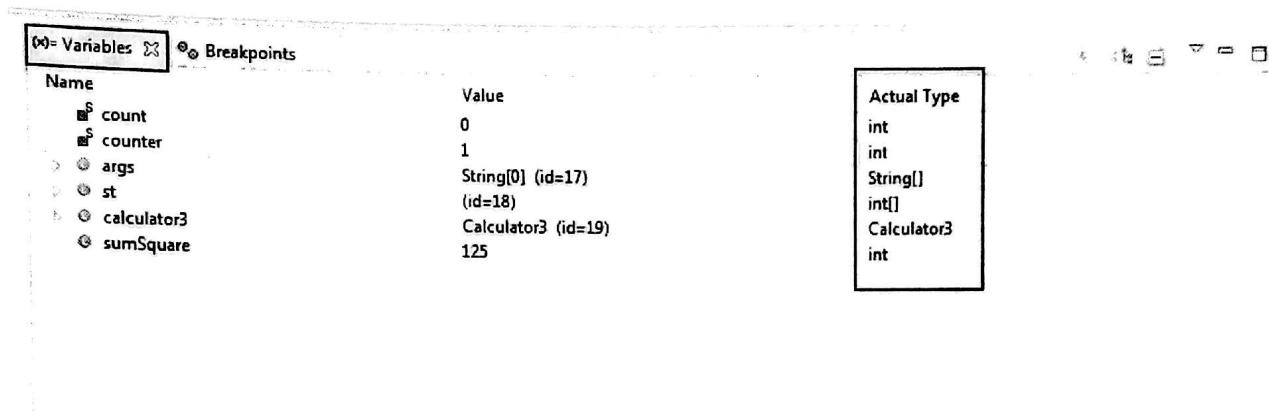


From the above screen select required checkbox types. So, that the selected value added as a column in the variables view. So, I am going to select the 'Actual Type' from the **Select columns window**.

#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
www. Sreenutech.com

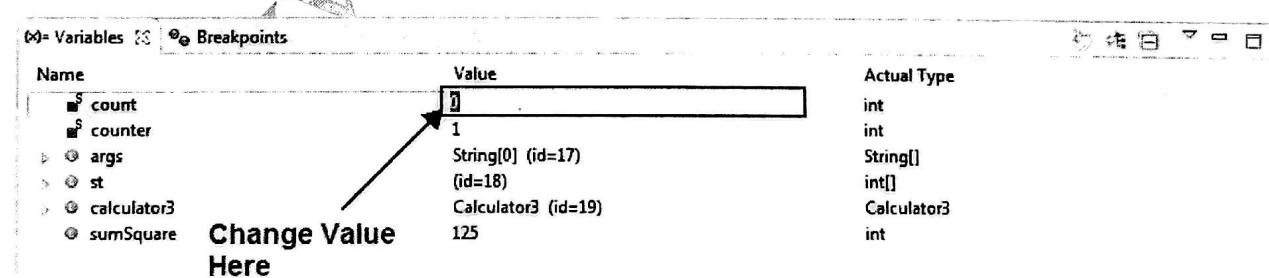


#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
[www.Sreenutech.com](http://www.Sreenutech.com)



### 3.6. Changing variable assignments in the debugger

The Variables view allows you to change the values assigned to your variable at runtime. This is depicted in the following screenshot.



### 3.7. Controlling the display of the variables with Detail Formatter

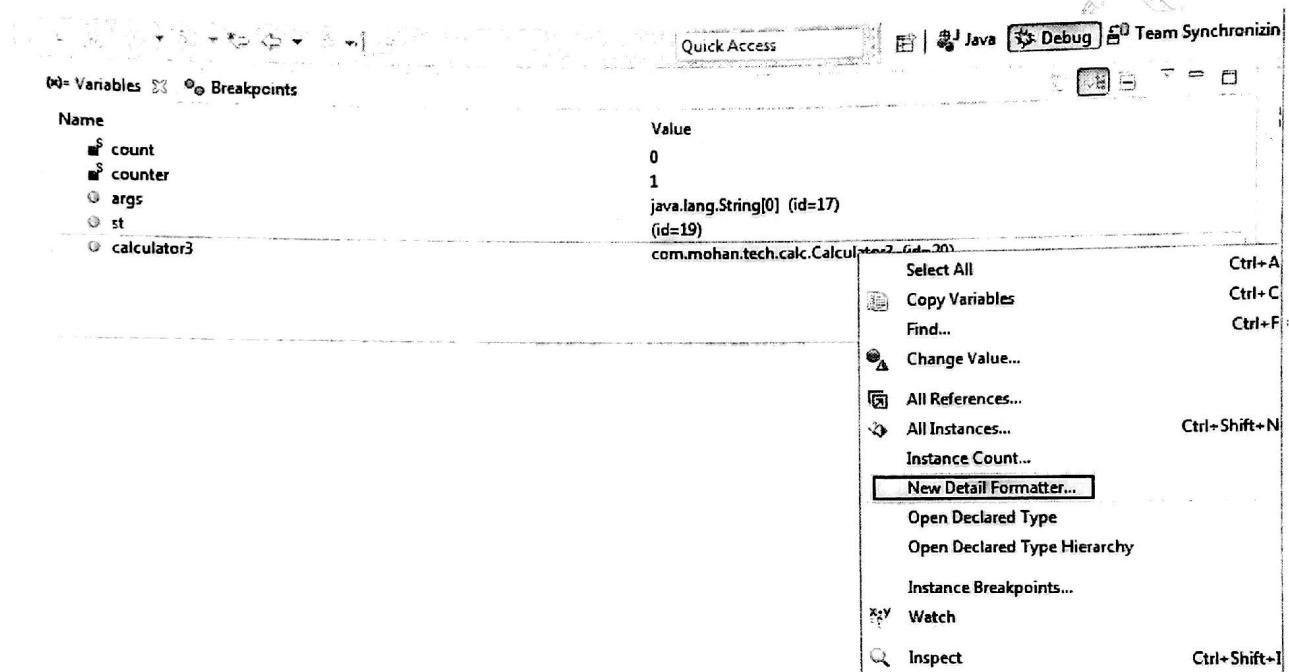
#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
 Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
[www.Sreenutech.com](http://www.Sreenutech.com)

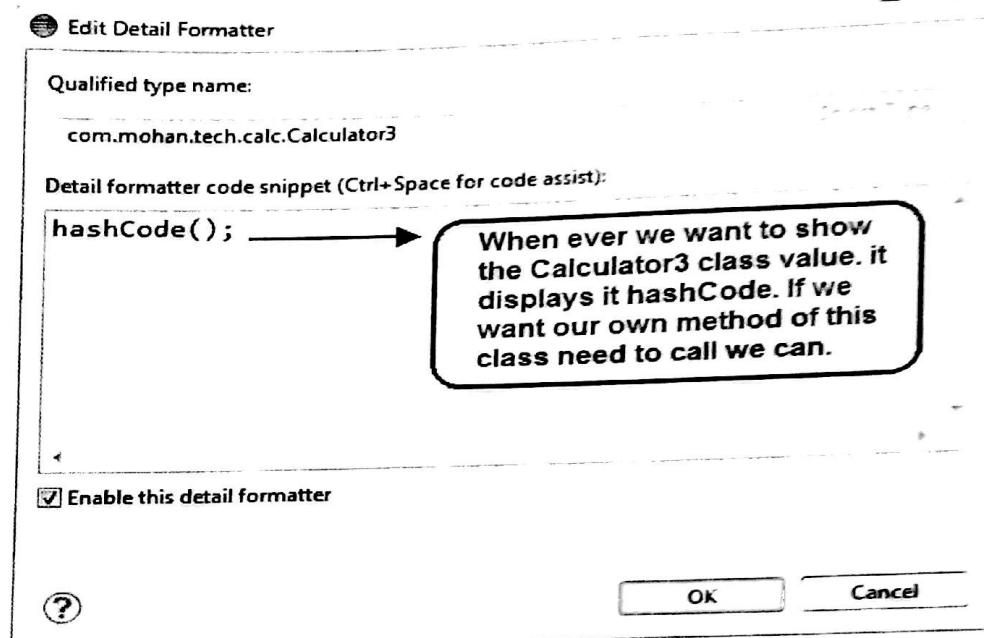


By default the Variables view uses the `toString()` method to determine how to display the variable.

You can define a Detail Formatter in which you can use Java code to define how a variable is displayed.

For example the `toString()` method in the Counter class may show meaningless information, e.g. `com.mohan.tech.calc.Calculator3@7e0c2ff5`. To make this output more readable you can right-click on the corresponding variable and select the New Detail Formater entry from the context menu.





#### 4.1. Breakpoint properties

After setting a breakpoint you can select the properties of the breakpoint, via right-click → Breakpoint Properties. Via the breakpoint properties you can define a condition that restricts the activation of this breakpoint.

You can for example specify that a breakpoint should only become active after it has reached 12 or more times via the Hit Count property.

You can also create a conditional expression. The execution of the program only stops at the breakpoint, if the condition evaluates to true. This mechanism can also be used for additional logging, as the code that specifies the condition is executed every time the program execution reaches that point.

The following screenshot depicts this setting.



# SREENU TECHNOLOGIES

The screenshot shows the Eclipse IDE interface with the title bar "Debug - cube-calc-proj/src/com/mohan/tech/calc/Counter.java - Eclipse SDK". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has icons for New, Open, Save, Cut, Copy, Paste, Find, Replace, Select All, Undo, Redo, and More. The left sidebar shows three open files: CalculatorClient.java, CalculatorB.java, and Counter.java. The Counter.java code is displayed:

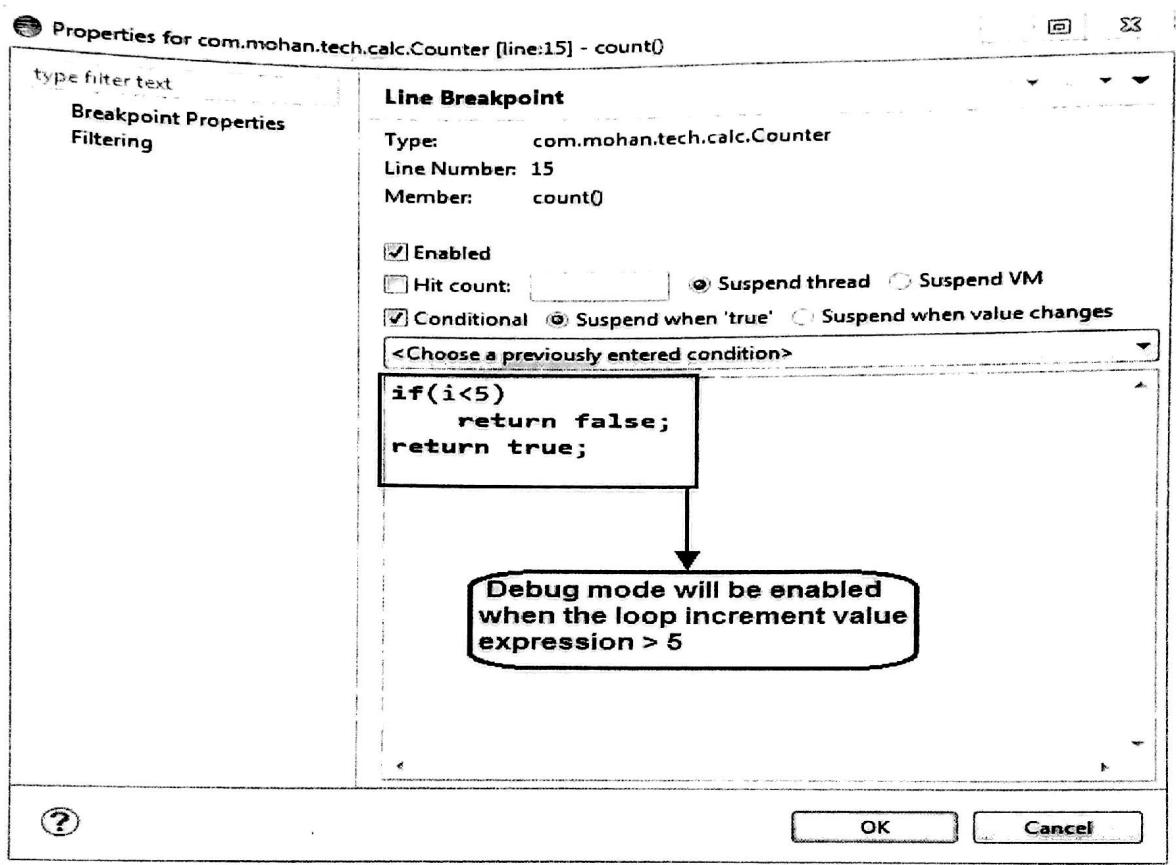
```
1 package com.mohan.tech.calc;
2
3 public class Counter {
4     private int result=0;
5     private int getResult(){
6         return result;
7     }
8     public void count(){
9         for(int i =0; i < 12; i ++){
10             result+=i;
11         }
12     }
13 }
14 
```

A context menu is open over the code, listing options such as Toggle Breakpoint, Disable Breakpoint, Go to Annotation, Show Bug Info, Show Bug in Bug Explorer, Add Bookmark..., Add Task..., Show Quick Diff, Show Line Numbers, Folding, Preferences..., and Breakpoint Properties... with their respective keyboard shortcuts.

ng[] args) {  
unter();  
lt :: "+counter.getResult());

WWW.SREENUTECH.COM

#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
www. Sreenutech.com



## 4.2. Watchpoint

A watchpoint is a breakpoint set on a field. The debugger will stop whenever that field is read or changed.

You can set a watchpoint by double-clicking on the left margin, next to the field declaration. In the properties of a watchpoint you can configure if the execution should stop during read access (Field Access) or during write access (Field Modification) or both.



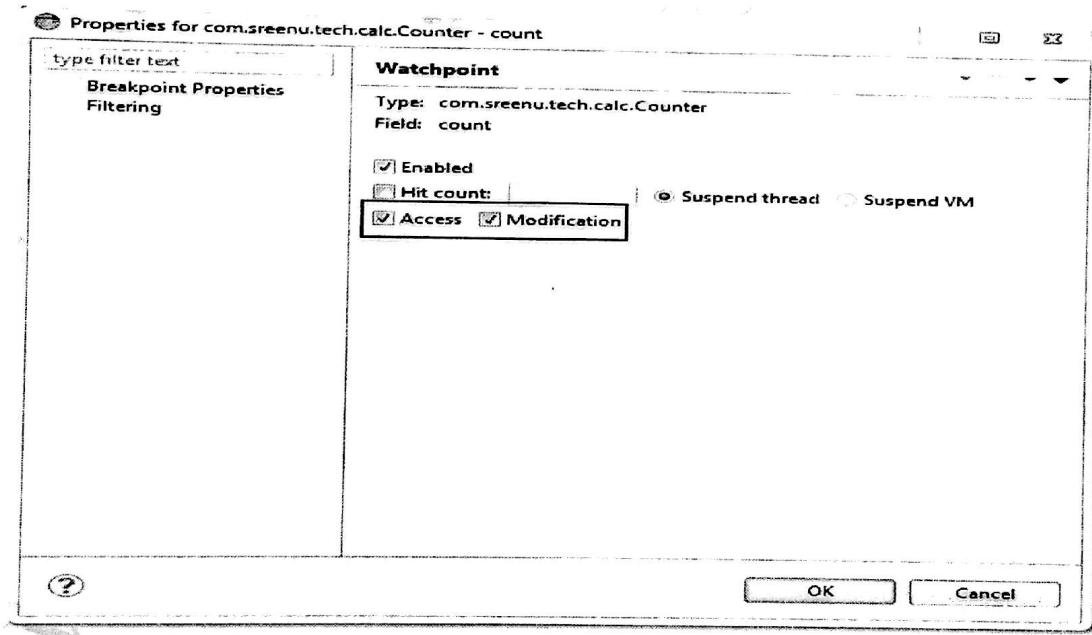
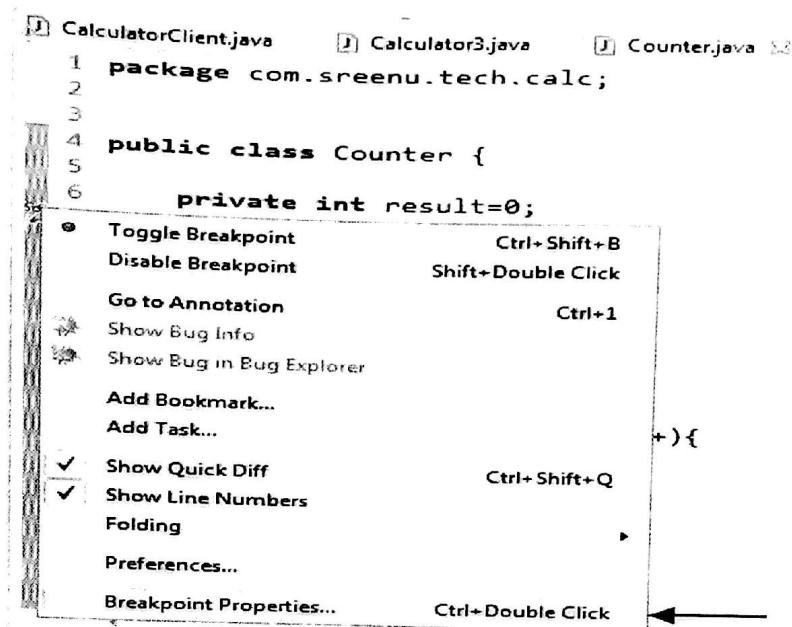
## SREENU TECHNOLOGIES

[J] CalculatorClient.java [J] Calculator3.java [J] Counter.java X

```
1 package com.sreenu.tech.calc;
2
3
4 public class Counter {
5
6     private int result=0;
7     int count =0;
8
9     private int getResult(){
10         return result;
11     }
12
13     public void count(){
14
15         for(int i =0; i< 12; i ++){
16             result+=i;
17             if(i == 8)
18                 count++;
19
20         }
21     }
22 }
```



# SREENU TECHNOLOGIES



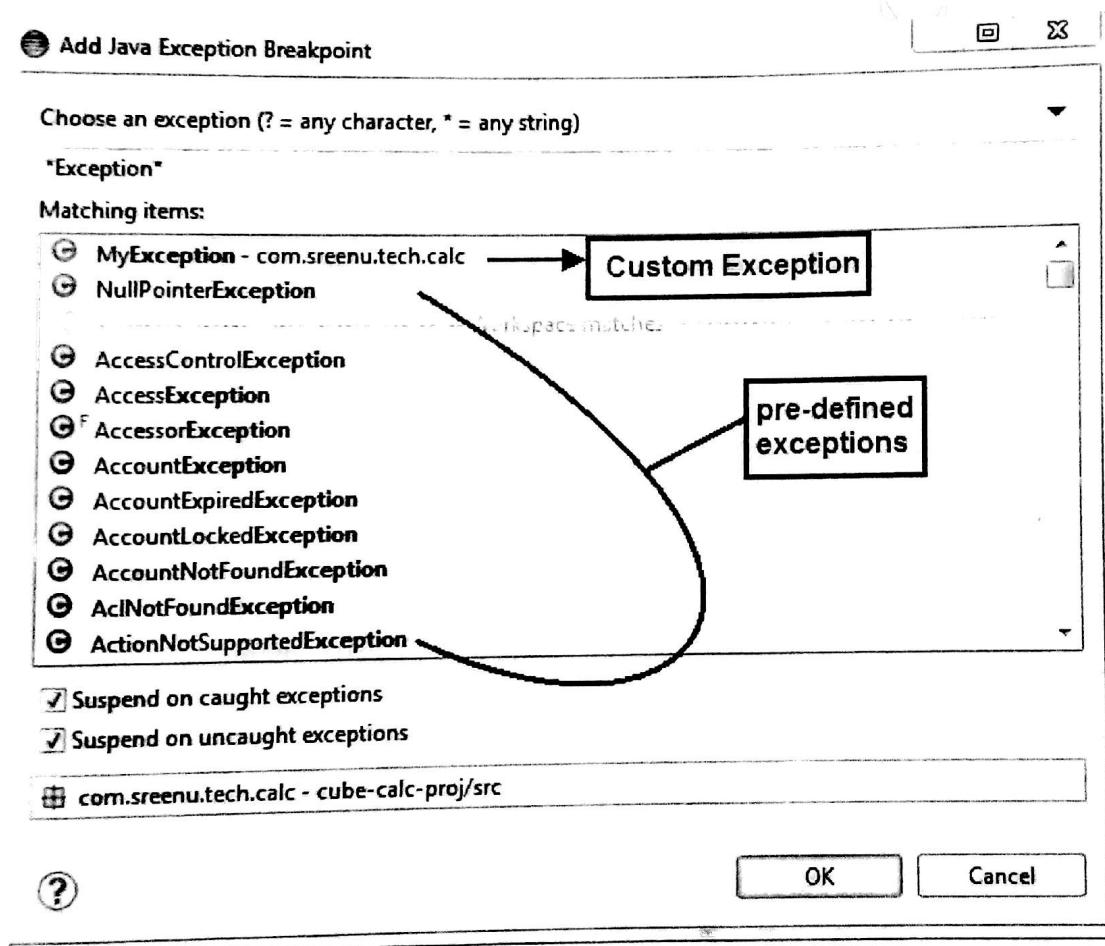
### 4.3. Exception breakpoints

You can set breakpoints which are triggered when exceptions in your Java source code are thrown. To define an exception breakpoint click on the Add Java Exception Breakpoint button icon in the Breakpoints view toolbar.

#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
www. Sreenutech.com



When we click on the above marked Icon then, the below window will be populated. From this window we can choose any of the pre-defined exceptions or custom exceptions which are configured in the application.





# SREENU TECHNOLOGIES

```
[1] CalculatorClient.java [2] CalculatorB.java [3] Counter.java [4] MyException.java
1 package com.sreenu.tech.calc;
2
3 public class Counter {
4     private int result=0;
5     int count =0;
6     private int getResult(){
7         return result;
8     }
9     public void count(){
10        od begin..."); 
11        +){
12            od leaving..."); 
13        ng[] args) {
14            unter();
15            String value:: "+str.toString());
16            ion n){
17                n(n);
18            }
19        System.out.println("Result :: "+counter.getResult());
20    }
21 }
```

A context menu is open over the code editor, showing various options for the selected line:

- Toggle Breakpoint
- Disable Breakpoint
- Go to Annotation
- Show Bug Info
- Show Bug in Bug Explorer
- Add Bookmark...
- Add Task...
- Show Quick Diff
- Show Line Numbers
- Folding
- Preferences...
- Breakpoint Properties...

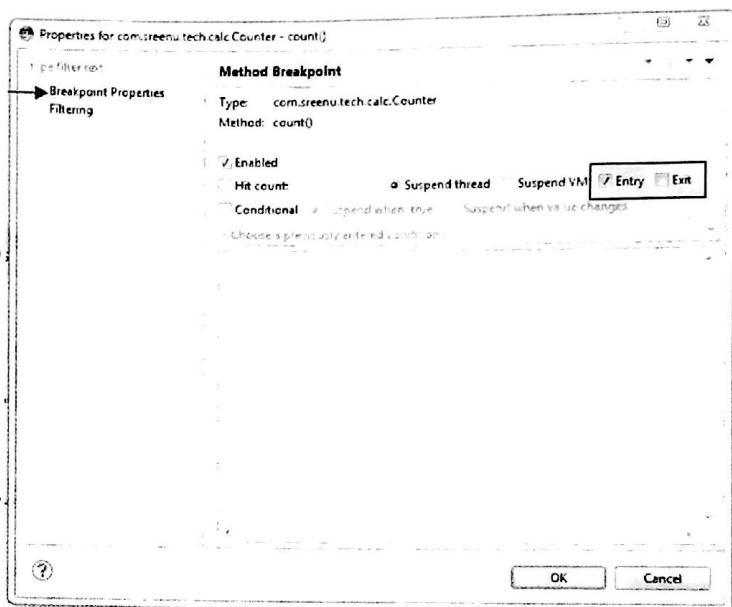
Keyboard shortcuts are listed next to each option: Ctrl+Shift+B, Shift+Double Click, Ctrl+1, Ctrl+Shift+Q, Ctrl+Shift+Q, Ctrl+Double Click.

```

CalculatorClient.java   Calculator3.java   Counter.java   MyException.java

1 package com.sreenu.tech.calc;
2
3
4 public class Counter {
5
6     private int result=0;
7     int count =0;
8
9     private int getResult(){
10         return result;
11     }
12
13     public void count(){
14         System.out.println("method begin...");
15         for(int i =0; i < 12; i ++){
16             result+=i;
17         }
18         System.out.println("method leaving...");
19     }
20
21     public static void main(String[] args) {
22         String str = null;
23         Counter counter = new Counter();
24         counter.count();
25         try{
26             System.out.println("String value::");
27         }catch(NullPointerException n){
28             throw new MyException(n);
29         }
30         System.out.println("Result :: "+counter.getResult());
31     }
32 }
33

```



#### 4.5. Breakpoints for loading classes

A class load breakpoint stops when the class is loaded.

#64/3RT, SR Nagar, Near Community Hall, Hyderabad  
 Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
[www.Sreenutech.com](http://www.Sreenutech.com)



SREENU TECHNOLOGIES

To set a class load breakpoint, right-click on a class in the Outline view and choose the **Toggle Class Load Breakpoint** option.

The screenshot shows an IDE interface with a Java file named `MyException.java` open. The code defines a class `MyException` that extends `RuntimeException`. The class has four constructors: a no-argument constructor, one taking a message and cause, one taking only a message, and one taking only a cause.

```
1 package com.sreenu.tech.calc;
2
3 public class MyException extends RuntimeException {
4
5     private static final long serialVersionUID = -8177537259814938431L;
6
7     public MyException() {
8     }
9
10    public MyException(String message, Throwable cause) {
11        super(message, cause);
12    }
13
14    public MyException(String message) {
15        super(message);
16    }
17
18    public MyException(Throwable cause) {
19        super(cause);
20    }
21
22
23
24 }
```

A context menu is open over the `MyException` class declaration, showing various options like Open Type Hierarchy, Show In, Copy, Delete, and Run As.

Now corresponding to the class name 'Toggle class load Breakpoint' appeared.

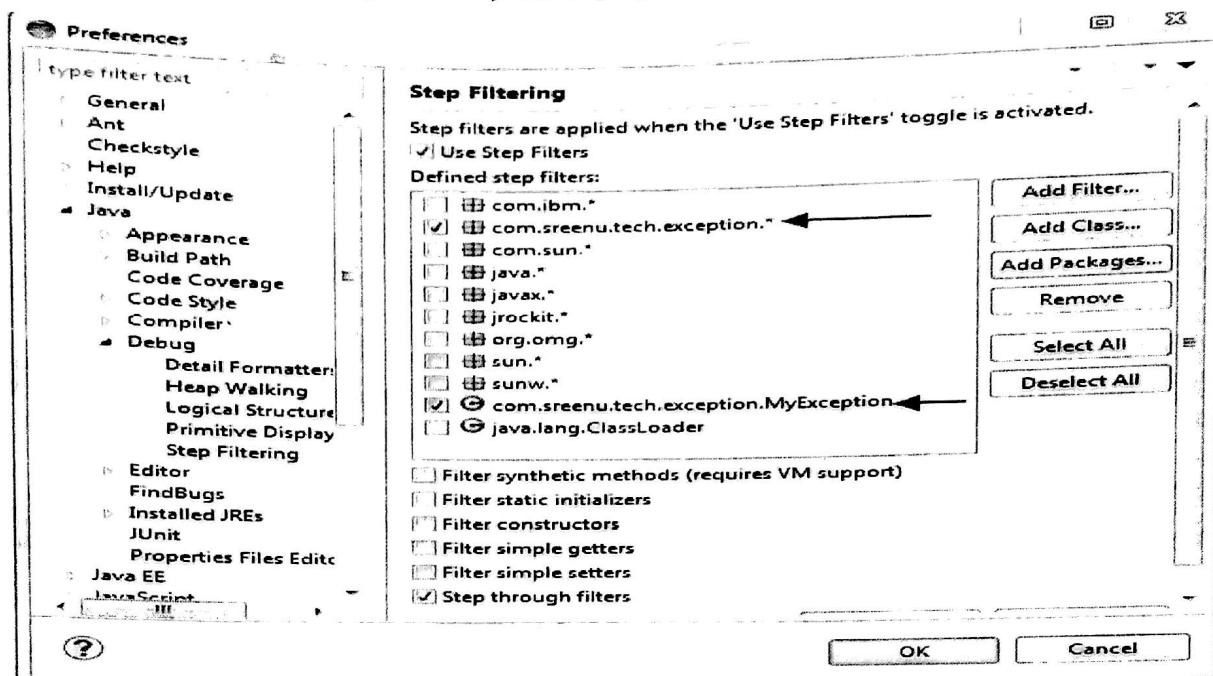


```
CalculatorClient.java    Calculator3.java    Counter.java    MyException.java  20
1 package com.sreenu.tech.calc;
2
3 public class MyException extends RuntimeException {
4     private static final long serialVersionUID = -8177537259814938431L;
5
6     public MyException() {
7
8 }
9
10    public MyException(String message, Throwable cause) {
11        super(message, cause);
12    }
13
14    public MyException(String message) {
15        super(message);
16    }
17
18    public MyException(Throwable cause) {
19        super(cause);
20    }
21
22
23 }
24
```

Alternative you can double-click in the left border of the Java editor beside the class definition.

#### 4.6. Step Filter

You can define that certain packages should be skipped in debugging. This is for example useful if you use a framework for testing but don't want to step into the test framework classes. These packages can be configured via the Window → Preferences → Java → Debug → Step Filtering menu path.

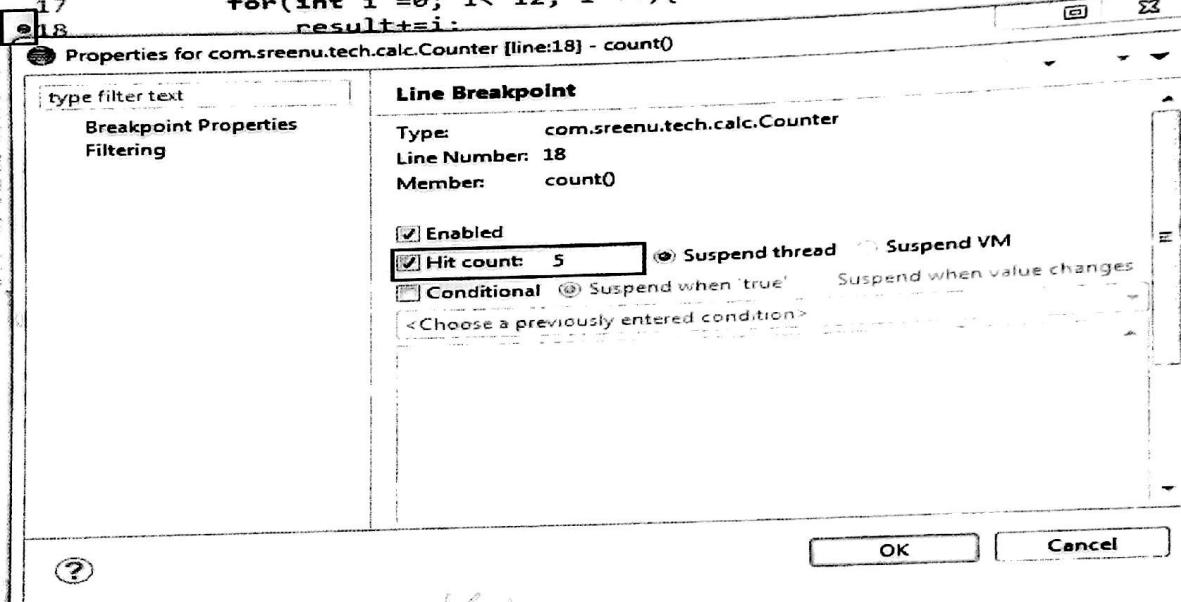


#### 4.7. Hit Count

For every breakpoint you can specify a hit count in its properties. The application is stopped once the breakpoint has been reached the number of times defined in the hit count.

```

[1] CalculatorClient.java [2] CalculatorB.java [3] Counter.java [4] MyException.java
  public class Counter {
    7
    8     private int result=0;
    9     int count =0;
   10
   11     private int getResult(){
   12         return result;
   13     }
   14
   15     public void count(){
   16         System.out.println("method begin...");
   17         for(int i =0; i < 12; i ++){
   18             result+=i;
  
```



#### 4.8. Remote debugging

Eclipse allows you to debug applications which runs on another Java virtual machine or even on another machine.

To enable remote debugging you need to start your Java application with certain flags, as demonstrated in the following code example.

```

java -Xdebug -Xnoagent \
-Djava.compiler=NONE \
-xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005
  
```

In you Eclipse IDE you can enter the hostname and port to connect for debugging via the Run

#64/3RT, SR Nagar, Near Community Hall, Hyderabad

Contacts: +91-8019697596, 040-40061799 Email-id: sreenutechnologies@gmail.com  
www. Sreenutech.com



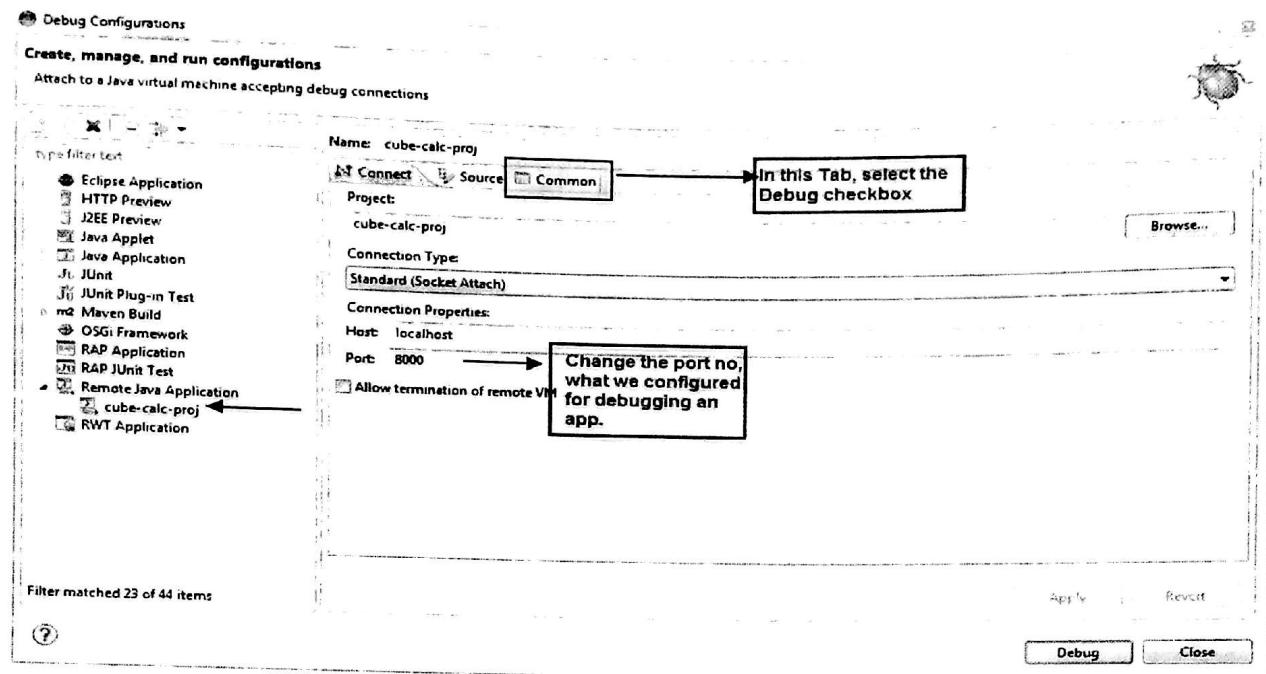
# SREENU TECHNOLOGIES

→ Debug Configuration... menu.

Here you can create a new debug configuration of the Remote Java Application type. This configuration allows you to enter the hostname and port for the connection as depicted in the following screenshot.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Structure:** The left sidebar shows a project named "cube-calc-proj" with a "src" folder containing several "Cal" files and a "com.sr" package.
- File Menu:** The "File" menu is open, showing options like New, Go Into, Open in New Window, Open Type Hierarchy, Show In, Copy, Paste, Delete, Build Path, Source, Refactor, Import..., Export..., Find Bugs, Build Project, Refresh, Close Project, Assign Working Sets..., Run As, Debug As, Profile As, Coverage As, Validate.
- Code Editor:** The main editor area displays Java code for a "Counter" class. The code includes a constructor, a "getResult()" method, a "count()" method that prints "method begin..." and loops from 0 to 12, and a "main()" method that creates a Counter object and calls its count() method.
- Context Menu:** A context menu is open over the code editor, with the "Debug Configurations..." option highlighted.
- Bottom Bar:** The bottom bar shows three options: 1 Java Applet, 2 Java Application, and Debug Configurations..., each with a keyboard shortcut.



### Note

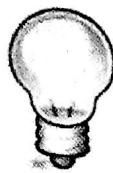
Remote debugging requires that you have the source code of the application which is debugged available in your Eclipse IDE.

### 4.9. Drop to frame

Eclipse allows you to select any level (frame) in the call stack during debugging and set the JVM to restart from that point.

This allows you to rerun a part of your program. Be aware that variables which have been modified by code that already run will remain modified.

To use this feature, select a level in your stack and press the Drop to Frame button in the toolbar of the Debug view .



## Note

Fields and external data may not be affected by the reset. For example if you write a entry to the database and afterward drop to a previous frame, this entry is still in the database.

The following screenshot depicts such a reset. If you restart your for loop, the field **result** is not set to its initial value and therefore the loop is not executed as without resetting the execution to a previous point.

Debug - cube-calc-proj/src/com/sreenu/tech/calc/Counter.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Debug Server

Counter [Java Application]

com.sreenu.tech.calc.Counter at localhost:54074

Thread [main] (Suspended (breakpoint at line 18 in Counter))

Counter.count() line: 18

Counter.main(String[]) line: 27

C:\Program Files\Eclipse 4.2.2\jdk1.6.0\_38\bin\javaw.exe (Sep 25, 2014 5:39:41 AM)

Variables Breakpoints

Name	Value
this	com.sreenu.tech.calc.Counter (id=16)
i	4

CalculatorClient.java    Counter.java    Exception.java

```
10
11  private int getResult(){
12      return result;
13  }
14
15  public void count(){
16      System.out.println("method begin...");
17      for(int i =0; i < 12; i ++){
18          result+=i*1;
19      }
20      System.out.println("method leaving1...");
21      System.out.println("method leaving2...");
22  }
23
24  public static void main(String[] args) {
25      String str = null;
26      Counter counter = new Counter();
```

