Remote z/OS Debugging
8355 The Debug Perspective

Exercise 1

1. Creating the Project, package and java classes (from Lab 1)

Create a Project, **File**>**New**>**Project**,

1. Select Project type 'Java Project'. Select **Next**.
2. Project name is 'Lab'. Select **Finish**.

Create a Package, **File**>**New**>**Package**,

1. Enter 'Lab' in 'Source Folder' text window.
2. Enter 'eclipseLab' in 'name' text window. Select **Finish**.

Create a Class, **File**>**New**>**Class**,

1. Enter 'Lab' in 'Source Folder' text window.
2. Enter 'eclipseLab' in 'Package' window.
3. Enter 'CountPrinter' in 'name' window.
4. Unselect all of the 'tick boxes' in the bottom 3 boxes.
5. Select **Finish**

Enter the following code in the generated 'CountPrinter' class template:

```
package eclipseLab;

public class CountPrinter {
      public CountPrinter(int num)
      {
            System.out.println("Count = " + num);
      }
}
```

Create another Class, **File**>**New**>**Class**,

1. Enter 'Lab' in 'Source Folder' text window.
2. Enter 'eclipseLab' in 'Package' window.
3. Enter 'RunIt' in 'name' window.
4. Tick the 'public static void main(String[] args)' 'tick box'
5. Select in **Finish**

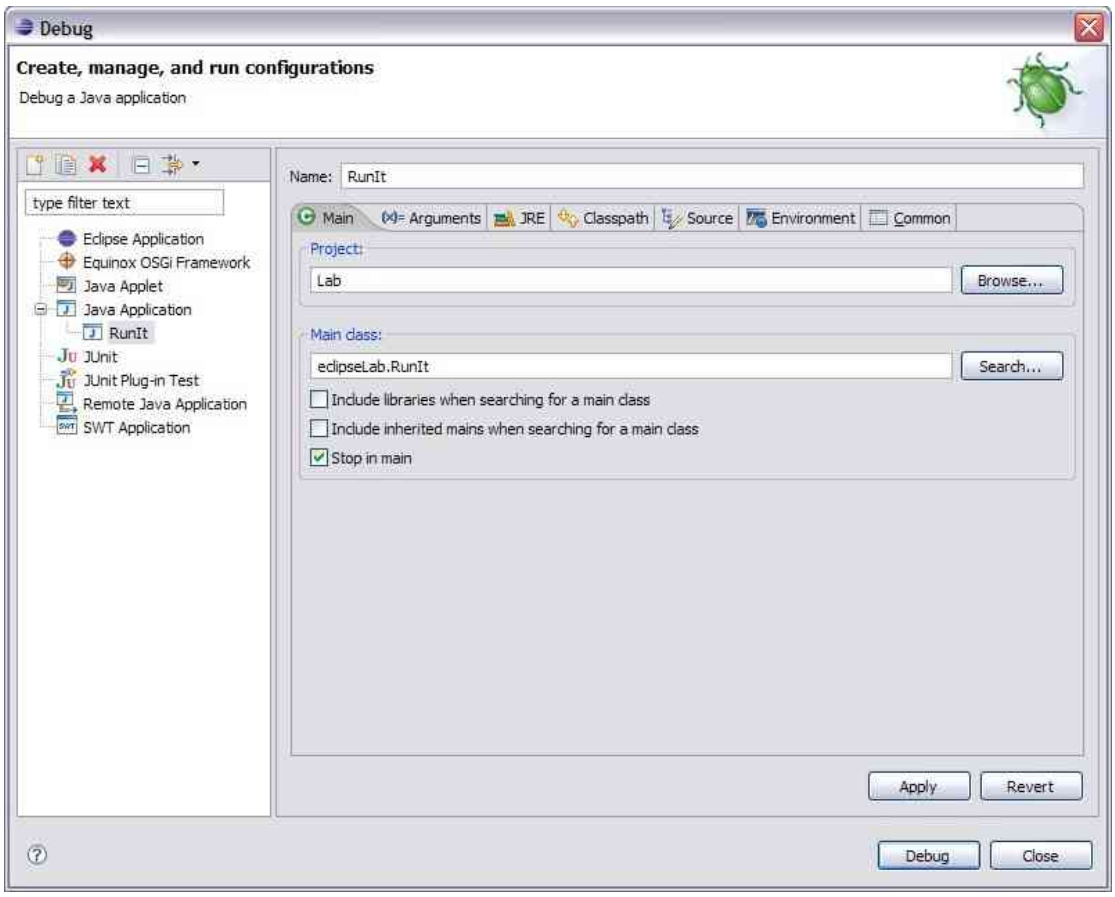
Enter the following code in the generated 'RunIt' class template. Note this time we'll count i to 1000:

```
package eclipseLab;

public class RunIt {

      public static void main(String[] args) {
            int i=0;
```

```
                System.out.println("Lab1");
                while (i++ < 1000) {
                      CountPrinter cp = new CountPrinter(i);
                }
          }
}
```

2. Debugging functionality

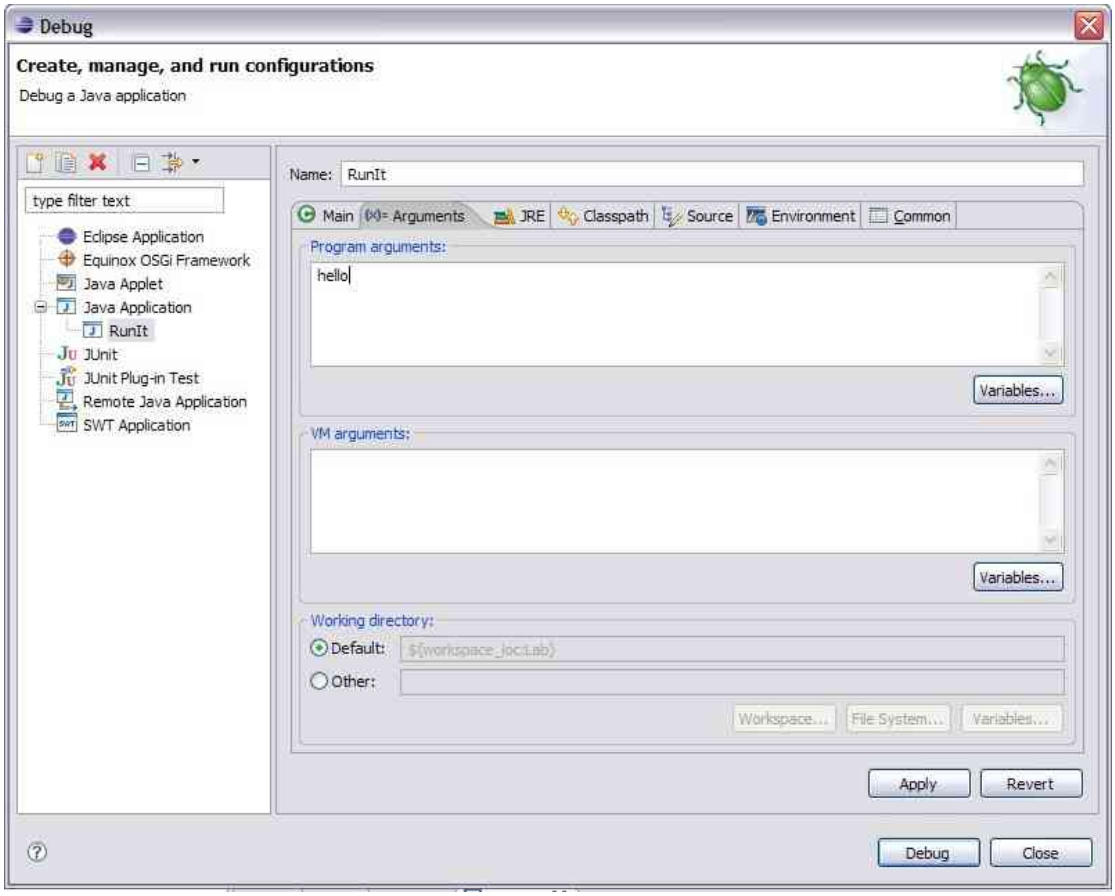
3. In the Java perspective open the RunIt class in the java window.
2. Enter **Run**>**Debug …** and the following window is displayed :



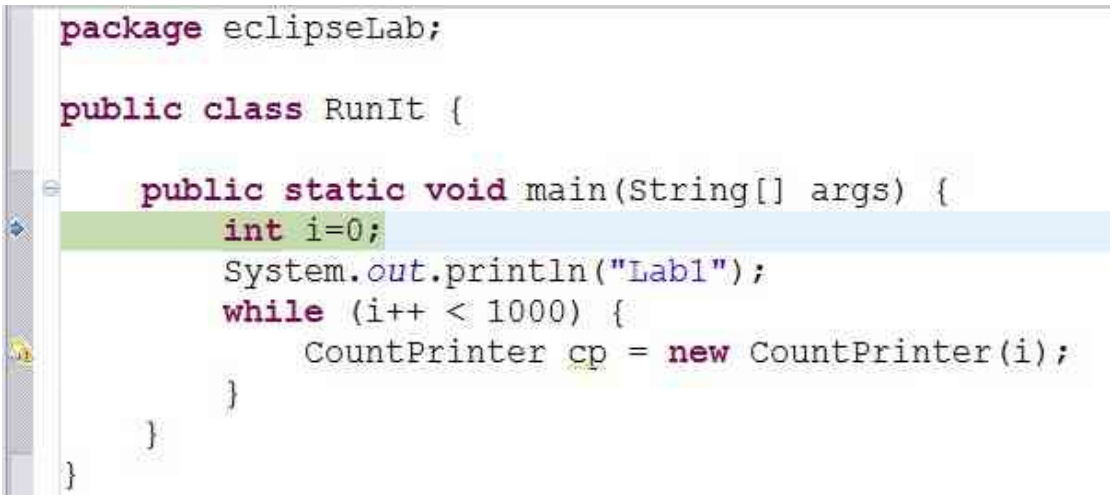
This is the debug configuration

Check that the Stop in main check box is selected.

5. Select the **Argument** tab and enter ‘hello’ in the ‘Program Arguments’ panel. See below.



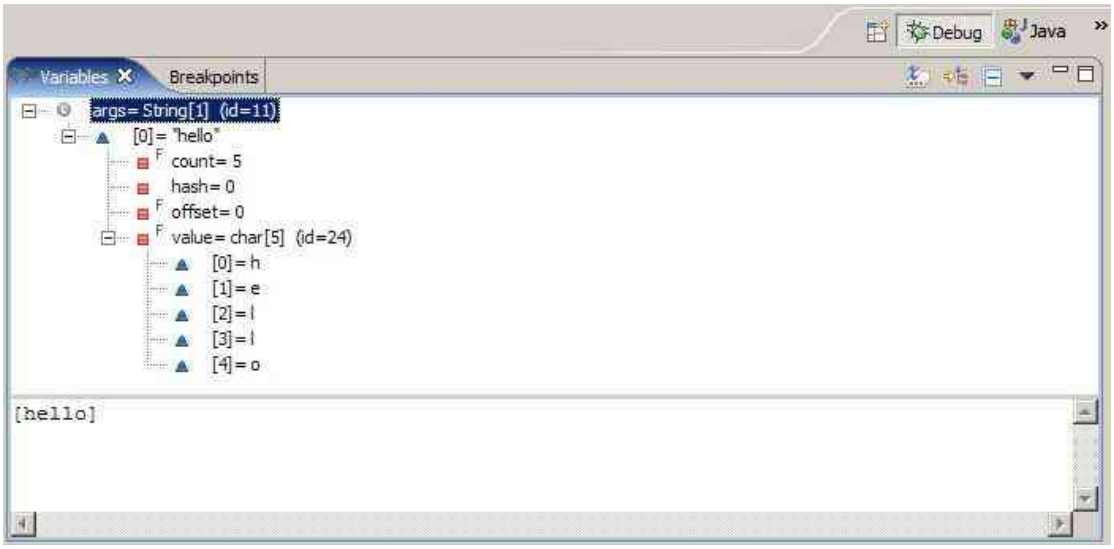
6. Click the **Debug** button.
   This opens the ‘Debug’ Perspective shown in the current slide in the presentation.
   The RunIt application is being executed and is suspended at the entry into method main.
   The blue arrow in the margin and the greyed line indicate where the programs is suspended.



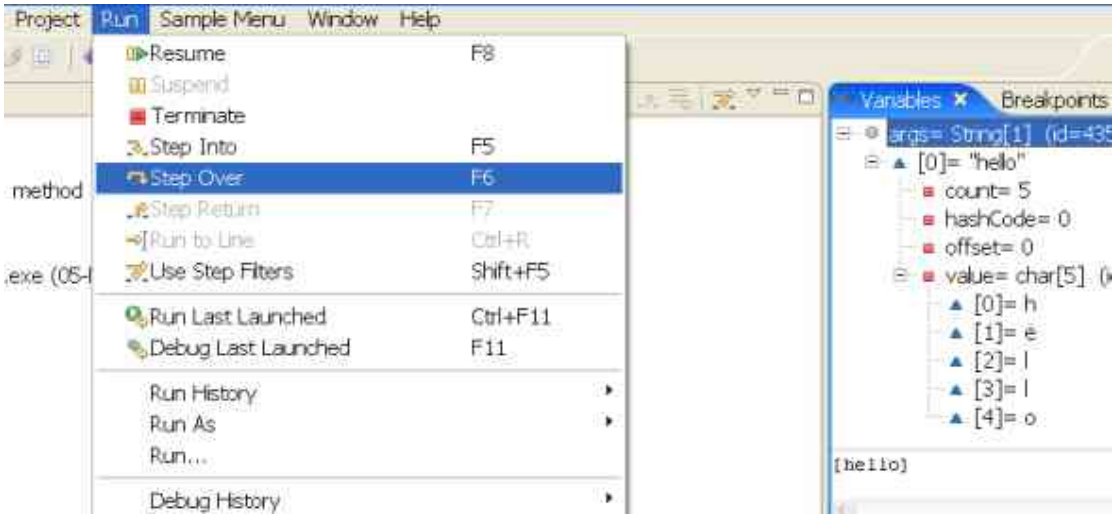
```
package eclipseLab;

public class RunIt {

    public static void main(String[] args) {
        int i=0;
        System.out.println("Lab1");
        while (i++ < 1000) {
            CountPrinter cp = new CountPrinter(i);
        }
    }
}
```

7. Look at the other windows in the debug perspective

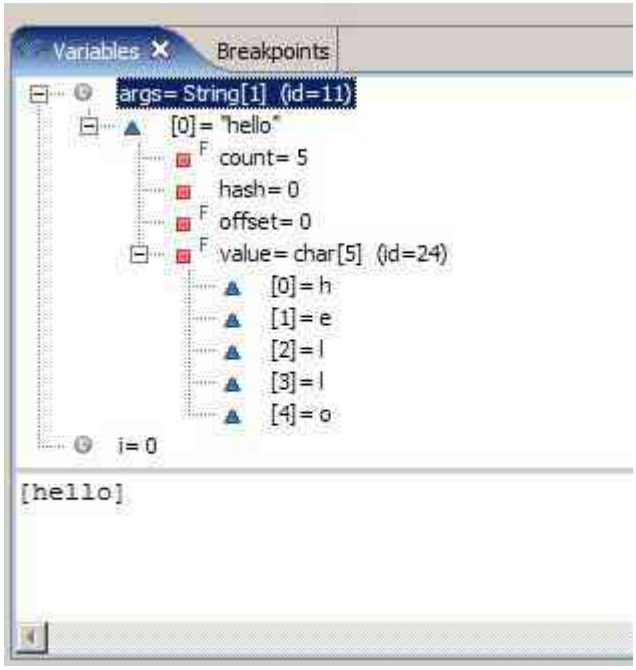
   In the attributes window's **variables** tab the only known 'variable' the program argument 'hello' is displayed.
   Try 'opening' the 'args' variable and see the various components of a String array exposed.



8. Click on the run menu option and see the debugging functionality now enabled:
   Resume, terminate, step with filters, step into …



7. Select **Step Over**. The variable ‘i’ now appears in the variable tab:

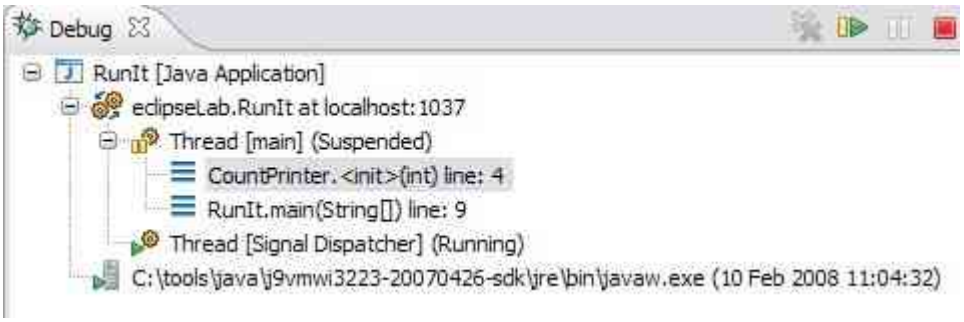


8. Select **Step Over** or press **F6** until ‘i’ is 10.
   See how the program is executing 1 line at a time.

9. Step through the program until the execution is suspended at line

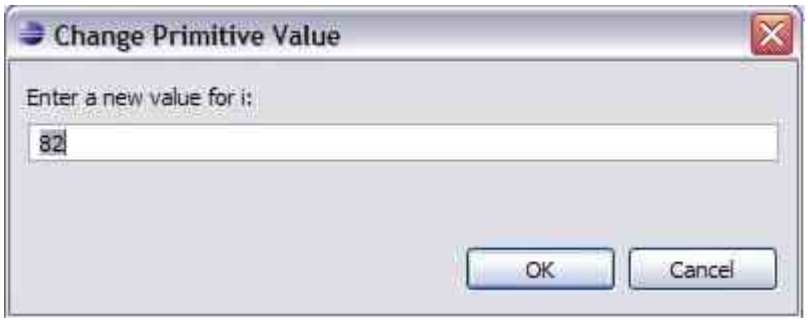
```
CountPrinter cp = new CountPrinter(i);
```

   Now enter **F5** or **Step Into** and see the program suspend in the CountPrinter() constructor. In the thread window the java stack can be seen. RunIt.main(String[]) method calls CountPrinter.<init>(int) method,



10. Now enter **F7** or **Step Return** to return to the RunIt.main method.

11. Place the cursor over ‘i’ in the **variable** tab and ‘right click’ and select **Change Value** to bring up the following window. ‘i’ can now be changed. Change it to **82** and click on **OK**. This enables the debugger to skip loop iterations.

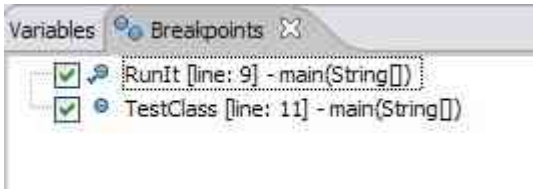


12. Step through the program (using **F6**) until the execution is suspended at line

```
CountPrinter cp = new CountPrinter(i);
```

Now depress the **Ctrl** + **Shift** + **b** keys. A breakpoint has been set at this source code line. Now enter **F8** or **Resume**, program execution now proceeds to the next breakpoint. Enter **F8** a few times and each time see ‘i’ incremented to indicate that a whole loop Iteration has been executed.

13. Click on the **breakpoints** tab to see the breakpoint added in section 12.

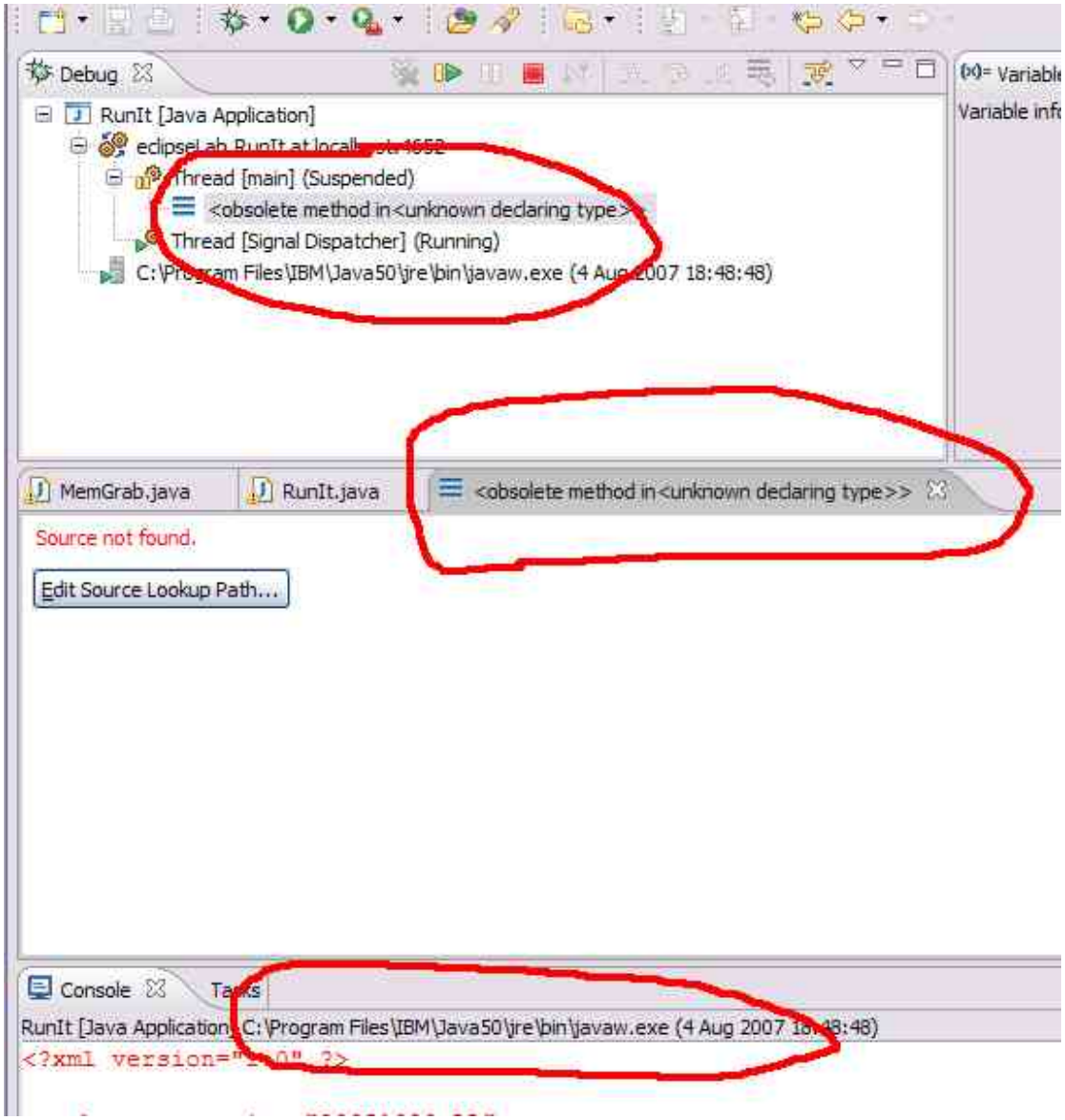


14. Try ‘hot code replacement’. Suspend the java application on a breakpoint in CountPrinter (use F7 to step-in). Modify the code, for example, change RunIt.java with the following 1 line addition:

```
int i=0;
System.out.println("Lab1");
while (i++ < 1000) {
      CountPrinter cp = new CountPrinter(i);
      i++;
}
```

Hit ‘Ctrl s’ to save RunIt.java. If you have automatic compilation enabled (default) RunIt.java will be recompiled. If you get an error, see below. Continue editing the application. You have changed the application code during an execution. Stepping through notice how i is incremented twice in each loop body as a result of this code modification.

15. I suspect you will see the following screen:



You have tried to modify a method while it is on the thread stack and the jvm has deemed that it is unsafe to continue. I say ‘suspect’ because another Vendor’s jvm might attempt to continue in this scenario. As you can see, this test was run using IBM’s Java 5 release.

Try hot code replacement with the following modified testcase.
Step through, in debug mode, exercising both the countPrinterRedirect and main methods. Stop on a break point in the main method and enable the commented out line, ‘ctrl s’ to save and recompile the RunIt class, now continue stepping through. It works! The method changed in flight was not in the call stack when the changes were made so the method could be recompiled and ‘safely’ executed on its next invocation.

```
package eclipseLab;

public class RunIt {
	public static void main(String[] args) {
		int i=0;
		System.out.println("Lab1");
		while (i++ < 100) {
```

```
                        countPrinterRedirect();
                }
        }

        public static void countPrinterRedirect () {
                int j =0;
                while (j++ < 20) {
                        CountPrinter cp = new CountPrinter(j);
                        // enable the following line in flight
                        // j++;
                }
        }
}
```

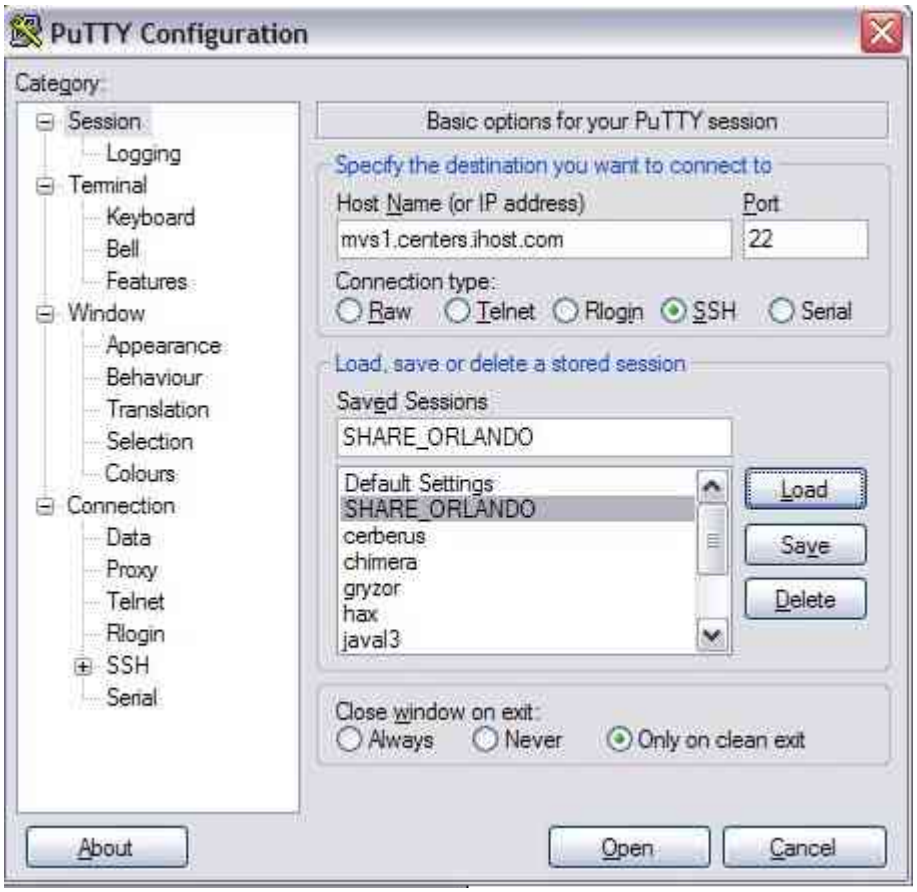
16. To end the lab select the terminate debug menu option (**Run**>**Terminate**).

Exercise 2

1. Compiling and running the CountPrinter application on z/OS

Log on to the z/OS machine using putty, compile and start the remote application

1. On your Windows desktop, open the "Java Putty" application (double-click)
2. Ensure the Host Name is `mvs1.centers.ihost.com` and click Open



3. The instructor will allocate you a username and password to login with

Review the remote RunIt application in the eclipseLab directory

4. Move to the eclipselab directory:
   `cd eclipselab[ENTER]`

5. Type the following from your home directory to view RunIt.java:
   `cat eclipseLab/RunIt.java[ENTER]`

6. Type the following from your home directory to view CountPrinter.java
   `cat eclipseLab/CountPrinter.java[ENTER]`

Compile and start the remote application

In the eclipselab directory is a file called runMeRemote. This file contains a script to setup the Java environment, compile the RunIt and CountPrinter Java files with the debug extensions on, and finally to start the application suspended waiting on debug instructions from a port.

7. Type the following from the eclipselab directory to view the runMeRemote script
   `cat runMeRemote[ENTER]`

8. Run the runMeRemote script
   runMeRemote

9. This will output the following line. Note the address as we'll need that in Eclipse:
   `Listening for transport dt_socket at address: 80nn`

The RunIt application is now waiting on input from our debugger. To debug we need to return to our Eclipse environment.

2. Remote Debugging in Eclipse

The version of the RunIt application on mvs1 is the same as the original version created in the last lab. To avoid inconsistencies, you must change the application back to how it was. You should have the following files:

CountPrinter.java

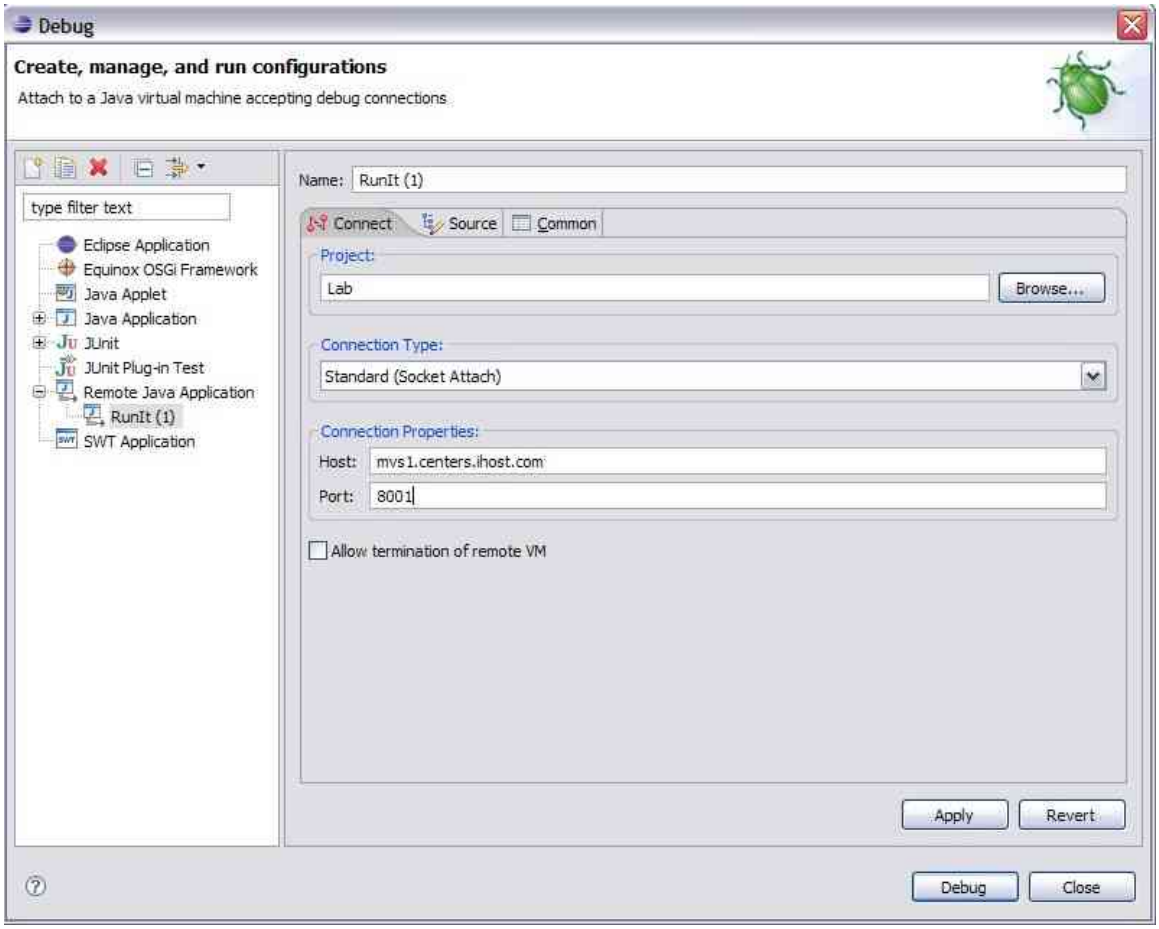
```
package eclipseLab;

public class CountPrinter {
	public CountPrinter(int num)
	{
		System.out.println("Count = " + num);
	}
}
```

RunIt.java

```
package eclipseLab;

public class RunIt {

	public static void main(String[] args) {
		int i=0;
		System.out.println("Lab1");
		while (i++ < 1000) {
			CountPrinter cp = new CountPrinter(i);
		}
	}
}
```

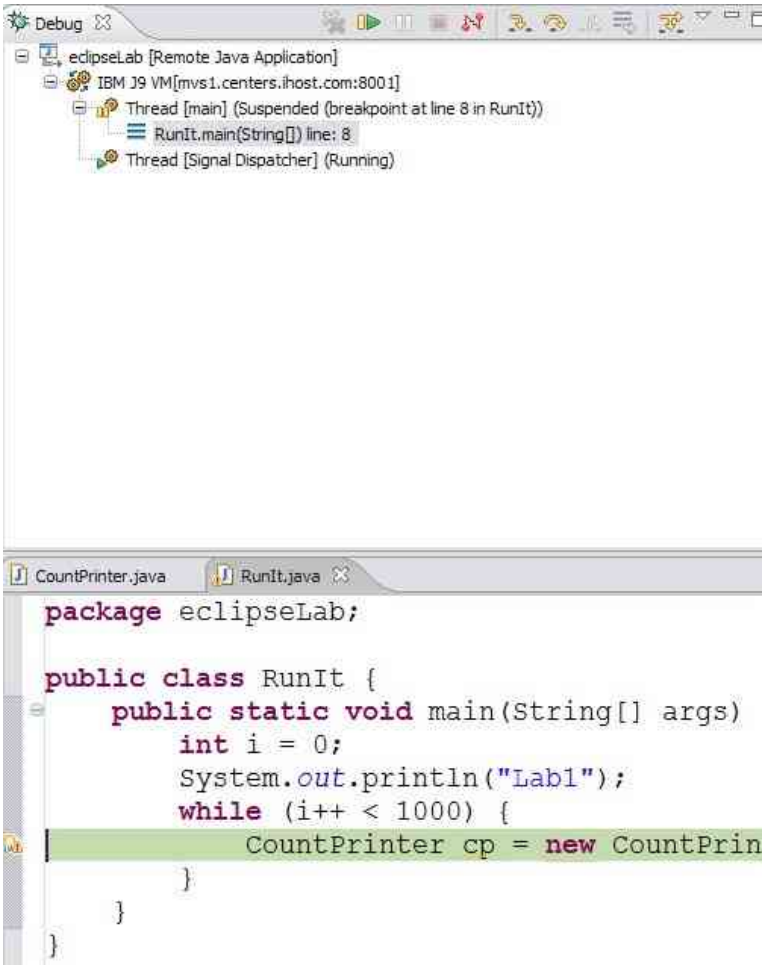
Start the debugger

1. Select **Run>Debug**, and ensure the Port setting is the one you noted down above:



Debug the application

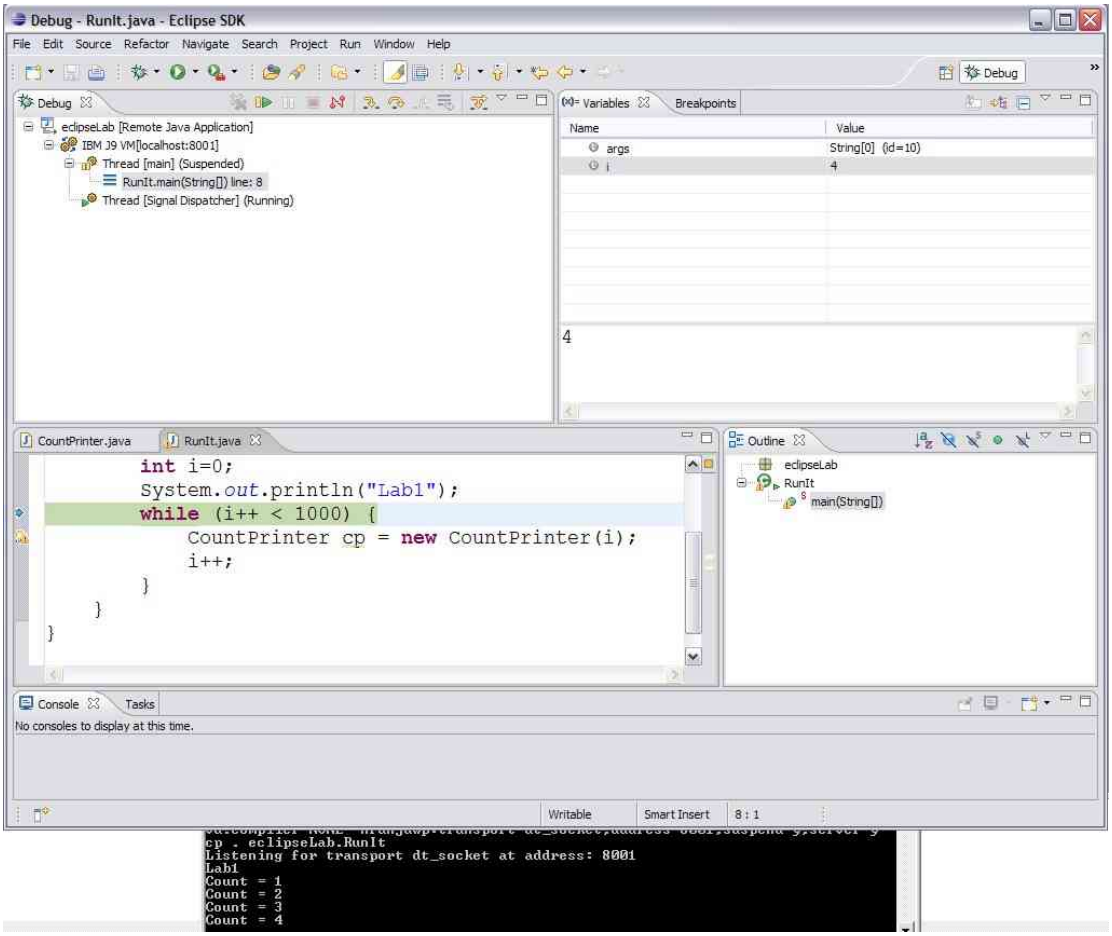
2. Eclipse will change to the debug perspective and allow you to debug as before. A breakpoint was set at the start of the main method and execution is suspended here:



3. Try using with some of the features we worked with in the previously lab:
   - step in, over
   - modifying variables in flight
   - breakpoints
   - hot code replacement

   If the remote application completes, restart it by simply rerunning the runMeRemote application.

   Note, any output will now appear on the remote machine. Check out this in-flight snapshot:



4. To end the lab select the terminate debug menu option (**`Run>Terminate`**).