Using the Eclipse Debugger

Java programs may contain three types of errors: syntax errors, runtime errors and logic errors. Syntax errors result from using the language rules incorrectly. Some errors are so severe that the program will not execute as a result. Others are flagged as warnings, but the program will execute in spite of them. For the present, you may ignore compiler messages that are warnings. Examples of syntax errors are forgetting to put a semicolon or brace in a block of statements. The compiler detects errors in syntax and reports them to you with problem markers and in the Problem view.

Once a program is compiled, it may have errors that occur when the program is executing that cause it to end abnormally or display strange results. The compiler cannot tell that these errors will occur because the syntax is correct. However, when the program executes, the computer will generally report these errors and stop termination of the program execution. These are called runtime errors. When runtime errors are detected, they will be reported as exceptions in a stack trace in the Console view.

Logic errors are errors that are made in developing an algorithm or problem solution. They are basically errors in thinking. Logic errors are not detected by the compiler or as runtime errors. They result in erroneous behavior in a program or incorrect output. Logic errors are the most difficult errors in a program to correct.

One of the benefits that Eclipse provides is the ability to run code interactively by using its integrated debugger. Examining variables and expressions while executing code step-by-step is an invaluable tool for investigating problems with your code.
Syntax Errors
These are displayed in the Problems view and annotated in the vertical ruler of your source code.

A problem is indicated in several ways:

- Listed in the Problems view

![Problems View]

- In the Package Explorer view, the Type Hierarchy or the Outline view, problem ticks appear on the affected Java elements and their parent elements.

![Package Explorer View]

- In the editor, there are 3 places that indicate syntax errors:
  - Squiggly lines appear under the word which might have caused the error.
  - The editor tab is annotated with a problem marker.
  - In the editor's vertical ruler, a problem marker is displayed near the affected line.
Notice that there are two types of red X's in the above illustrations:

- Hover over the red X or the marked word in the editor to view a description of the problem. (This is the same statement that is in the Problems view).
  - The problem in the above illustration is caused by a missing semicolon.

- The light bulb signals that correction proposals are available for this problem. Click on the light bulb X to see the suggestions. Be careful when accepting a suggestion that it is what you know is the correct choice to fix the problem. Double click on the statement that you want to use or make the correction manually.

---

**Eclipse Debug Perspective**

![Eclipse Debug Perspective](image)

- **Debug controls**
- **Debug view**
- **Variables view**
- **active line**
- **breakpoint**

---

```java
10          */
11          The declared package "edu.ilstu" does not match the expected package "edu.ilstu".
12          import java;
13          Move 'Problem4.java' to package 'edu.ilstu';
14          Change package declaration to 'edu.ilstu.itk177';
```
Debugging a Java Language Program

Before you are able to debug your project, the code needs to compile and run cleanly.

- Launch the application using Debug instead of Run
  o From the menu choose Run -> Debug As -> Java Application
  o Right-click on the class name in the Package Explorer and choose Debug As -> Java Application
  o Click on the bug icon instead of the "play" button.

Note: For both the Debug icon and the Run icon, if the class you want to run was the last class you ran (you can tell by clicking on the arrow to see if it is on the top of the list), you only need to click on the icon to debug or run again.
• If you are not there, you will be prompted to switch to the Debug perspective
  o Click the check box: ☑ Remember my decision and then Yes

![Confirm Perspective Switch](image)

**Setting Breakpoints**
The most common debugging procedure is to set breakpoints that will allow the
inspection of variables and the values inside conditional statements or loops. A
breakpoint suspends the execution of a program at the location where the
breakpoint is set. This allows you to see the value of variables in the Variables view
at that particular point of the program.

To set breakpoints:
• Be sure the class that you want to debug is open in the editor and that it is
  selected if there is more than one file open.
• Go to the line in the code where the breakpoint needs to be set.
• Set the breakpoint in one of these ways:
  o Double-click on the left margin next to the line that needs a
    breakpoint.
    ▪ The breakpoint will be indicated by a blue dot in the margin.
  o From the menu choose Run -> Toggle Breakpoint
• A breakpoint can be removed in the same way:
  o Double-click on the breakpoint
  o Run -> Toggle Breakpoint
• Breakpoints can also be managed in the Breakpoint View in the Debug Perspective
Stepping Through the Code using the Debug Controls

- Resume
  - Continue execution until the program ends or the next breakpoint.
- Terminate
  - Stop the debug process
- Step Into
  - Go to the next line of code. If it is a method call, go into that method.
- Step Over
  - Go to the next line of code. If it is a method call, execute that method and move to the next line.
- Step Return
  - Return from a Stepped-Into method back to the caller.
- Use Step Filters
  - Click on this button to keep from going into the code for any Core API classes.
    - This is generally considered "noise"
  - Usually you just want to step into methods of classes that you wrote, so remember to click this.