Troubleshooting

The **Troubleshooting** trail attempts to point out common problems that new Java programmers encounter while learning to program in the Java language. Currently, this trail contains two pages that discuss problems that you may experience using the Java compiler and the Java interpreter. We anticipate more pages for this trail in the future which will cover common problems with other areas in the Java development environment such as the debugger and working with applets.

Troubleshooting Compiler Problems

This page explains some common problems programmers have with the java compiler and how to fix them.

Troubleshooting Interpreter Problems

This page explains some common problems programmers have with the java interpreter and how to fix them.

Troubleshooting Compiler Problems

Can’t Locate the Compiler

[PENDING: do platform−specific information here for all platforms] On UNIX systems, you may see the following error message if your path isn’t set properly.

```
javac: Command not found
```

Use `setenv` or similar command to modify your path variable so that it includes the directory where the Java compiler lives.

Syntax Errors

If you mistype part of a program, the compiler may issue a *syntax* error. The message usually displays the type of the error, the line number where the error was detected, the code on that line, and the position of the error within the code. Here’s an error caused by omitting a **`';'`** at the end of a statement:

```
testing.java:14: ';' expected.
    System.out.println("Input has " + count + " chars.")
                                                       ^
1 error
```

Sometimes the compiler can’t guess your intent and prints a confusing error message or multiple error messages if the error cascades over several lines. For example, this code snippet omits a **`';'`** from the bold line.

```
while (System.in.read() != -1)
    count++
System.out.println("Input has " + count + " chars.");
```

When processing this code, the compiler issues two error messages:

```
testing.java:13: Invalid type expression.
        count++
                ^
testing.java:14: Invalid declaration.
    System.out.println("Input has " + count + " chars.");
```

```
                                ^
2 errors
```

The compiler issued two error messages because after it processed `count++`, the compiler's state indicates that it's in the middle of an expression. Without the `';'` the compiler has no way of knowing that the statement is complete.

If you see any compiler errors, then your program did not successfully compile, and the compiler did not create a `.class` file. Carefully verify the program, fix any errors that you detect and try again.

Semantic Errors

In addition to verifying that your program is syntactically correct, the compiler checks for other basic correctness. For example, the compiler warns you *each time* you use a variable that has not been initialized:

```
testing.java:13: Variable count may not have been initialized.
        count++;
        ^
testing.java:14: Variable count may not have been initialized.
    System.out.println("Input has " + count + " chars.");
                                      ^
2 errors
```

Again, your program did not successfully compile, and the compiler did not create a `.class` file. Fix the error and try again.

See Also

The Java Development Environment

Compiler Man Page

Troubleshooting

Troubleshooting Interpreter Problems

Can’t Find Class

A common error of beginner Java programmers is to try to interpret the `.class` file created by the compiler. For example, in *The Nuts and Bolts of the Java Language* the compiler creates a file called `Count.class`. If you try to interprete *the file* rather than the class, the interpreter displays this error message:

```
Can’t find class Count.class
```

The argument to the Java interpreter is the *name of the class* that you want to use, ***not*** *the filename*.

The main() Method is Not Defined

The Java interpreter requires that the class you execute with it have a method named `main()`, because the interpreter must have somewhere to begin execution of your Java application. The main() Method discusses the `main()` method in detail.

If you try to run a class with the Java interpreter that does not have a `main()` method, the interpreter prints this error message

```
In class classname: void main(String argv[]) is not defined
```

where *classname* is the name of the class that you tried to run.

See Also

The Java Development Environment

Interpreter Man Page

Troubleshooting