

Java runs in a Java Virtual Machine (JVM), a layer that translates Java code into bytecode compatible with your operating system. Get an open source JVM from openjdk.java.net or developers.redhat.com/products/openjdk/download

Java Packages	
Related classes are grouped into a package. Declare a package name at the top of your code.	When creating libraries, you can create packages within a package to provide access to each unique class.
<pre>package com.opensource.hello; /* @author your-name */</pre>	<pre>package com.opensourc.greeting; /* @author your-name */</pre>
Java Imports	Java Variables
When importing libraries needed for your Java code, use the <code>import</code> key word. Imports work based on your environment's Java path, and open source libraries can be bundled with your application (license permitting.)	Java is strongly typed, meaning all variable types must be declared when created. <i>Local</i> variables may be created inside a method, <i>instance</i> variables may be created inside a class, and <i>static</i> variables may be shared across instances.
<pre>package com.opensource.hello; /* @author your-name */ import java.lang.System; import javax.swing.*; //code...</pre>	<pre>//code... public class Greeting { static int num = 42; public static String namer() { String name = "Java"; int number =1; double bignum = 3.1415926535897932384; float smallnum = 3.141592; char character = 'a'; boolean toggle = true; return name; } }</pre>
Java Classes	Java Methods
A class file must contain one <code>public</code> class. Other classes may exist elsewhere and may not be public. By convention, class names start with a capital letter.	Java methods may be <code>public</code> (accessed by any other class), <code>private</code> (known only within a class), <code>protected</code> (unavailable to an unrelated class), or default. Provide the type of returned data, such as <code>void</code> , <code>int</code> , <code>float</code> , etc.
<pre>package com.opensource.hello; /* @author your-name */ import java.lang.System; public class Greeting { public static String namer() { //code... } }</pre>	<pre>//code... public static String namer() { String name = "Java"; return name; } } public static void main(String[] args) { String player = greeting.namer(); System.out.printf("%s,%s", named, num); } }</pre>

Try and Catch

To catch errors in Java, start with `try`, fall back on `catch`, and end with `finally`. Should the `try` clause fail, then `catch` is invoked, and in the end, there's `finally` to perform some action regardless of the results.

```
try {
    cmd = parser.parse(opt, args);

    if(cmd.hasOption("help")) {
        HelpFormatter helper = new HelpFormatter();
        helper.printHelp("Hello <options>", opt);
        System.exit(0);
    }
    else {
        if(cmd.hasOption("shell") || cmd.hasOption("s")) {
            String target = cmd.getOptionValue("tgt");
        } // else
    } // if
} catch (ParseException err) {
    System.out.println(err);
    System.exit(1);
} //catch
finally {
    new Hello().helloWorld(opt);
} //finally
} //try
```

Arguments	Run a .java file
<p>Passing arguments into a Java application is done with the <code>args</code> keyword, preceded by the type of acceptable arguments (such as <code>String</code>, <code>int</code>, and so on).</p>	<p>Java files, usually ending in <code>.java</code>, can be run from your IDE or in a terminal using the <code>java</code> command. If an application is complex, however, running a single file may not be useful. Add arguments as appropriate.</p>
<pre>//code... public static void main (String[] args) { System.out.printf("You rolled a "); DiceRoller App = new DiceRoller(Integer.parseInt(args[0])); App.Roller(); } }</pre>	<pre>\$ java ./Example.java \$ java ./diceroller.java 20 You rolled a 17</pre>
	Run a JAR file
	<p>Usually, Java applications are distributed as Java Archives (JAR) files, ending in <code>.jar</code>. To run a JAR file, double-click its icon or launch it from a terminal.</p>
	<pre>\$ java -jar /path/to/Example.jar</pre>