

Chapter 1: Introduction to Java

About the Java Technology

Java technology is both a programming language and a platform.

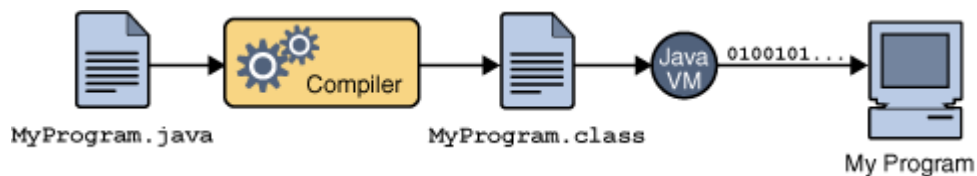
The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

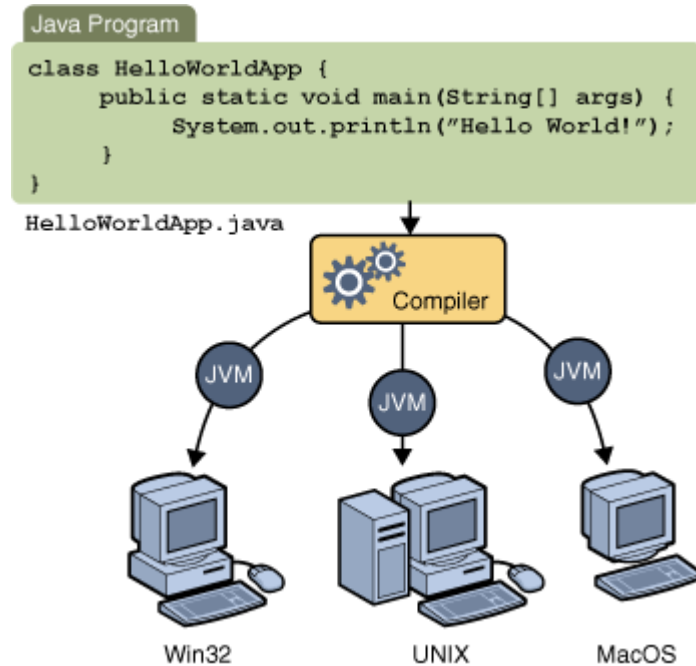
Each of the preceding buzzwords is explained in *The Java Language Environment*, a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the `.java` extension. Those source files are then compiled into `.class` files by the `javac` compiler. A `.class` file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine¹ (Java VM). The `java` launcher tool then runs your application with an instance of the Java Virtual Machine.



An overview of the software development process.

Because the Java VM is available on many different operating systems, the same `.class` files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java HotSpot virtual machine, perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.



Through the Java VM, the same application is capable of running on multiple platforms.

The Java Platform

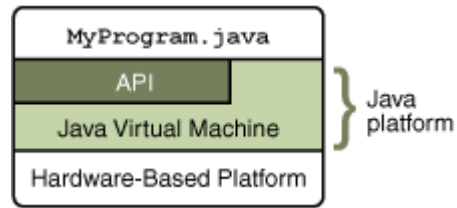
A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine*
- The *Java Application Programming Interface (API)*

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, *What Can Java Technology Do?* highlights some of the functionality provided by the API.



The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

What Can Java Technology Do?

The general-purpose, high-level Java programming language is a powerful software platform. Every full implementation of the Java platform gives you the following features:

- **Development Tools:** The development tools provide everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the `javac` compiler, the `java` launcher, and the `javadoc` documentation tool.
- **Application Programming Interface (API):** The API provides the core functionality of the Java programming language. It offers a wide array of useful classes ready for use in your own applications. It spans everything from basic objects, to networking and security, to XML generation and database access, and more. The core API is very large; to get an overview of what it contains, consult the Java SE Development Kit 6 (JDK™ 6) documentation.
- **Deployment Technologies:** The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits:** The Swing and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries:** Integration libraries such as the Java IDL API, JDBC™ API, Java Naming and Directory Interface™ ("J.N.D.I.") API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

How Will Java Technology Change My Life?

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program written in C++.
- **Write better code:** The Java programming language encourages good coding practices, and automatic garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans™ component architecture, and its wide-ranging, easily extendible API let you reuse existing, tested code and introduce fewer bugs.
- **Develop programs more quickly:** The Java programming language is simpler than C++, and as such, your development time could be up to twice as fast when writing in it. Your programs will also require fewer lines of code.
- **Avoid platform dependencies:** You can keep your program portable by avoiding the use of libraries written in other languages.
- **Write once, run anywhere:** Because applications written in the Java programming language are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily:** With Java Web Start software, users will be able to launch your applications with a single click of the mouse. An automatic version check at startup ensures that users are always up to date with the latest version of your software. If an update is available, the Java Web Start software will automatically update their installation.

Design Goals of the Java Programming Language

The design requirements of the Java™ programming language are driven by the nature of the computing environments in which software must be deployed.

The massive growth of the Internet and the World-Wide Web leads us to a completely new way of looking at development and distribution of software. To live in the world of electronic commerce and distribution, Java technology must enable the development of *secure, high performance*, and highly *robust* applications on *multiple platforms* in *heterogeneous, distributed networks*.

Operating on multiple platforms in heterogeneous networks invalidates the traditional schemes of binary distribution, release, upgrade, patch, and so on. To survive in this jungle, the Java programming language must be *architecture neutral*, *portable*, and *dynamically adaptable*.

The system that emerged to meet these needs is *simple*, so it can be easily programmed by most developers; *familiar*, so that current developers can easily learn the Java programming language; *object oriented*, to take advantage of modern software development methodologies and to fit into distributed client-server applications; *multithreaded*, for high performance in applications that need to perform multiple concurrent activities, such as multimedia; and *interpreted*, for maximum portability and dynamic capabilities.

Together, the above requirements comprise quite a collection of buzzwords, so let's examine some of them and their respective benefits before going on.

1.2.1 Simple, Object Oriented, and Familiar

Primary characteristics of the Java programming language include a *simple* language that can be programmed without extensive programmer training while being attuned to current software practices. The fundamental concepts of Java technology are grasped quickly; programmers can be productive from the very beginning.

The Java programming language is designed to be *object oriented* from the ground up. Object technology has finally found its way into the programming mainstream after a gestation period of thirty years. The needs of distributed, client-server based systems coincide with the encapsulated, message-passing paradigms of object-based software. To function within increasingly complex, network-based environments, programming systems must adopt object-oriented concepts. Java technology provides a clean and efficient object-based development platform.

Programmers using the Java programming language can access existing libraries of tested objects that provide functionality ranging from basic data types through I/O and network interfaces to graphical user interface toolkits. These libraries can be extended to provide new behavior.

Even though C++ was rejected as an implementation language, keeping the Java programming language looking like C++ as far as possible results in it being a *familiar* language, while removing the unnecessary complexities of C++. Having the Java programming language retain many of the object-oriented features and the "look and feel" of C++ means that programmers can migrate easily to the Java platform and be productive quickly.

1.2.2 Robust and Secure

The Java programming language is designed for creating highly *reliable* software. It provides extensive compile-time checking, followed by a second level of run-time checking. Language features guide programmers towards reliable programming habits.

The memory management model is extremely simple: objects are created with a `new` operator. There are no explicit programmer-defined pointer data types, no pointer arithmetic, and automatic garbage collection. This simple memory management model eliminates entire classes of programming errors that bedevil C and C++ programmers. You can develop Java code with confidence that the system will find many errors quickly and that major problems won't lay dormant until after your production code has shipped.

Java technology is designed to operate in distributed environments, which means that *security* is of paramount importance. With security features designed into the language and run-time system, Java technology lets you construct applications that can't be invaded from outside. In the network environment, applications written in the Java programming language are secure from intrusion by unauthorized code attempting to get behind the scenes and create viruses or invade file systems.

1.2.3 Architecture Neutral and Portable

Java technology is designed to support applications that will be deployed into heterogeneous network environments. In such environments, applications must be capable of executing on a variety of hardware architectures. Within this variety of hardware platforms, applications must execute atop a variety of operating systems and interoperate with multiple programming language interfaces. To accommodate the diversity of operating environments, the Java Compiler™ product generates *bytecodes*--an *architecture neutral* intermediate format designed to transport code efficiently to multiple hardware and software platforms. The interpreted nature of Java technology solves both the binary distribution problem and the version problem; the same Java programming language byte codes will run on any platform.

Architecture neutrality is just one part of a truly *portable* system. Java technology takes portability a stage further by being strict in its definition of the basic language. Java technology puts a stake in the ground and specifies the sizes of its basic data types and the behavior of its arithmetic operators. Your programs are the same on every platform--there are no data type incompatibilities across hardware and software architectures.

The architecture-neutral and portable language platform of Java technology is known as the *Java virtual machine*. It's the specification of an abstract machine for which Java programming language compilers can generate code. Specific implementations of the Java virtual machine for specific hardware and software platforms then provide the concrete realization of the virtual machine. The Java virtual machine is based primarily

on the POSIX interface specification--an industry-standard definition of a portable system interface. Implementing the Java virtual machine on new architectures is a relatively straightforward task as long as the target platform meets basic requirements such as support for multithreading.

1.2.4 High Performance

Performance is always a consideration. The Java platform achieves superior performance by adopting a scheme by which the interpreter can run at full speed without needing to check the run-time environment. The *automatic garbage collector* runs as a low-priority background thread, ensuring a high probability that memory is available when required, leading to better performance. Applications requiring large amounts of compute power can be designed such that compute-intensive sections can be rewritten in native machine code as required and interfaced with the Java platform. In general, users perceive that interactive applications respond quickly even though they're interpreted.

1.2.5 Interpreted, Threaded, and Dynamic

The *Java interpreter* can execute Java bytecodes directly on any machine to which the interpreter and run-time system have been ported. In an interpreted platform such as Java technology-based system, the link phase of a program is simple, incremental, and lightweight. You benefit from much faster development cycles--prototyping, experimentation, and rapid development are the normal case, versus the traditional heavyweight compile, link, and test cycles.

Modern network-based applications, such as the HotJava™ Browser for the World Wide Web, typically need to do several things at the same time. A user working with HotJava Browser can run several animations concurrently while downloading an image and scrolling the page. Java technology's *multithreading* capability provides the means to build applications with many concurrent threads of activity. Multithreading thus results in a high degree of interactivity for the end user.

The Java platform supports multithreading at the language level with the addition of sophisticated synchronization primitives: the language library provides the `Thread` class, and the run-time system provides monitor and condition lock primitives. At the library level, moreover, Java technology's high-level system libraries have been written to be *thread safe*: the functionality provided by the libraries is available without conflict to multiple concurrent threads of execution.

While the Java Compiler is strict in its compile-time static checking, the language and run-time system are *dynamic* in their linking stages. Classes are linked only as needed. New code modules can be linked in on demand from a variety of sources, even from sources across a network. In the case of the HotJava Browser and similar applications,

interactive executable code can be loaded from anywhere, which enables transparent updating of applications. The result is on-line services that constantly evolve; they can remain innovative and fresh, draw more customers, and spur the growth of electronic commerce on the Internet.

1.3 The Java Platform--a New Approach to Distributed Computing

Taken individually, the characteristics discussed above can be found in a variety of software development platforms. What's completely new is the manner in which Java technology and its runtime environment have combined them to produce a flexible and powerful programming system.

Developing your applications using the Java programming language results in software that is *portable* across multiple machine architectures, operating systems, and graphical user interfaces, *secure*, and *high performance*. With Java technology, your job as a software developer is much easier--you focus your full attention on the end goal of shipping innovative products on time, based on the solid foundation of the Java platform. The *better way* to develop software is here, now, brought to you by the Java platform.

"Hello World!" for Microsoft Windows

Creating Your First Application

Your first application, `HelloWorldApp`, will simply display the greeting "Hello world!". To create this program, you will:

- **Create a source file**

A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files.

- **Compile the source file into a .class file**

The Java programming language *compiler* (`javac`) takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this file are known as *bytecodes*.

- **Run the program**

The Java application *launcher tool* (`java`) uses the Java virtual machine to run your application.

Create a Source File

To create a source file, you have two options:

- You can save the file [HelloWorldApp.java](#) on your computer and avoid a lot of typing. Then, you can go straight to [Compile the Source File into a .class File](#).
- Or, you can use the following (longer) instructions.

First, start your editor. You can launch the Notepad editor from the **Start** menu by selecting **Programs > Accessories > Notepad**. In a new document, type in the following code:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Be Careful When You Type



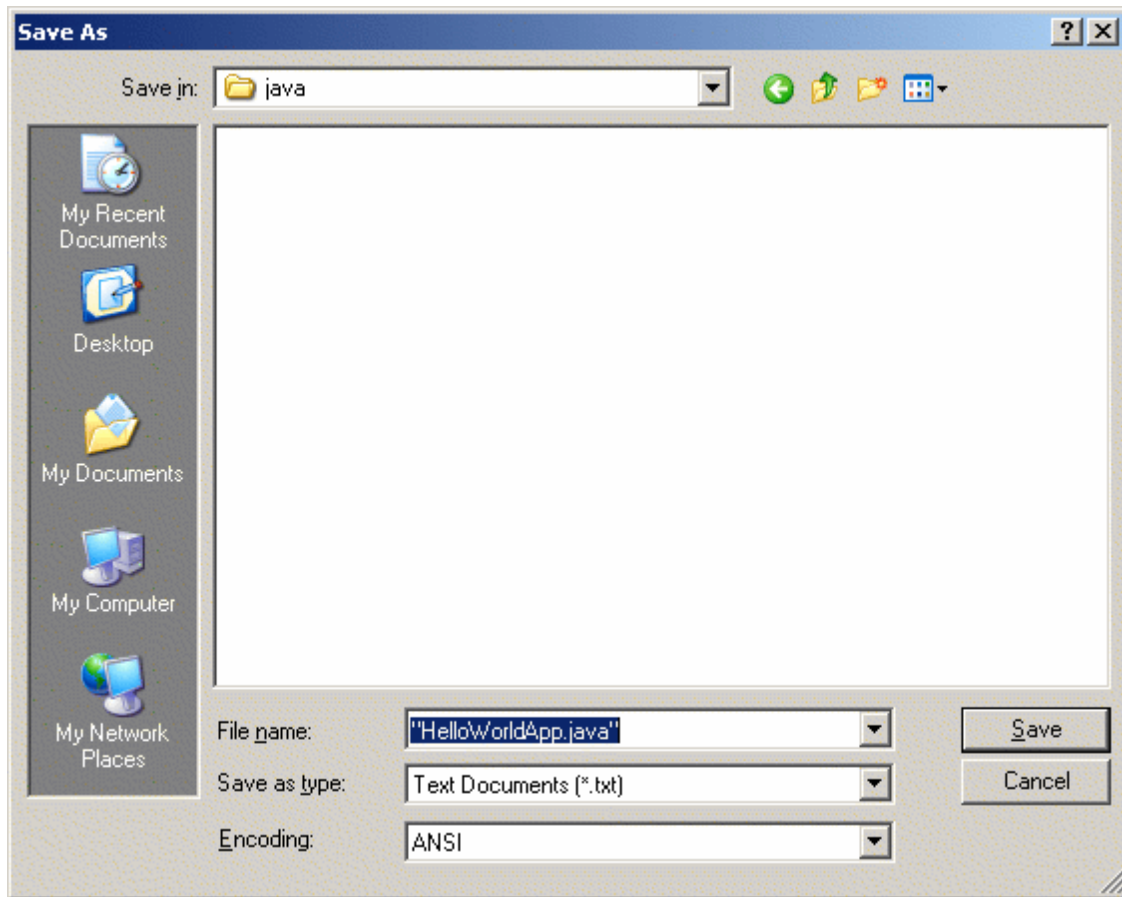
Type all code, commands, and file names exactly as shown. Both the compiler (javac) and launcher tool (java) are *case-sensitive*, so you must capitalize consistently.

HelloWorldApp  helloworldapp

Save the code in a file with the name `HelloWorldApp.java`. To do this in Notepad, first choose the **File > Save As** menu item. Then, in the **Save As** dialog box:

1. Using the **Save in** combo box, specify the folder (directory) where you'll save your file. In this example, the directory is `java` on the C drive.
2. In the **File name** text field, type `"HelloWorldApp.java"`, including the quotation marks.
3. From the **Save as type** combo box, choose **Text Documents (*.txt)**.
4. In the **Encoding** combo box, leave the encoding as ANSI.

When you're finished, the dialog box should look like this.

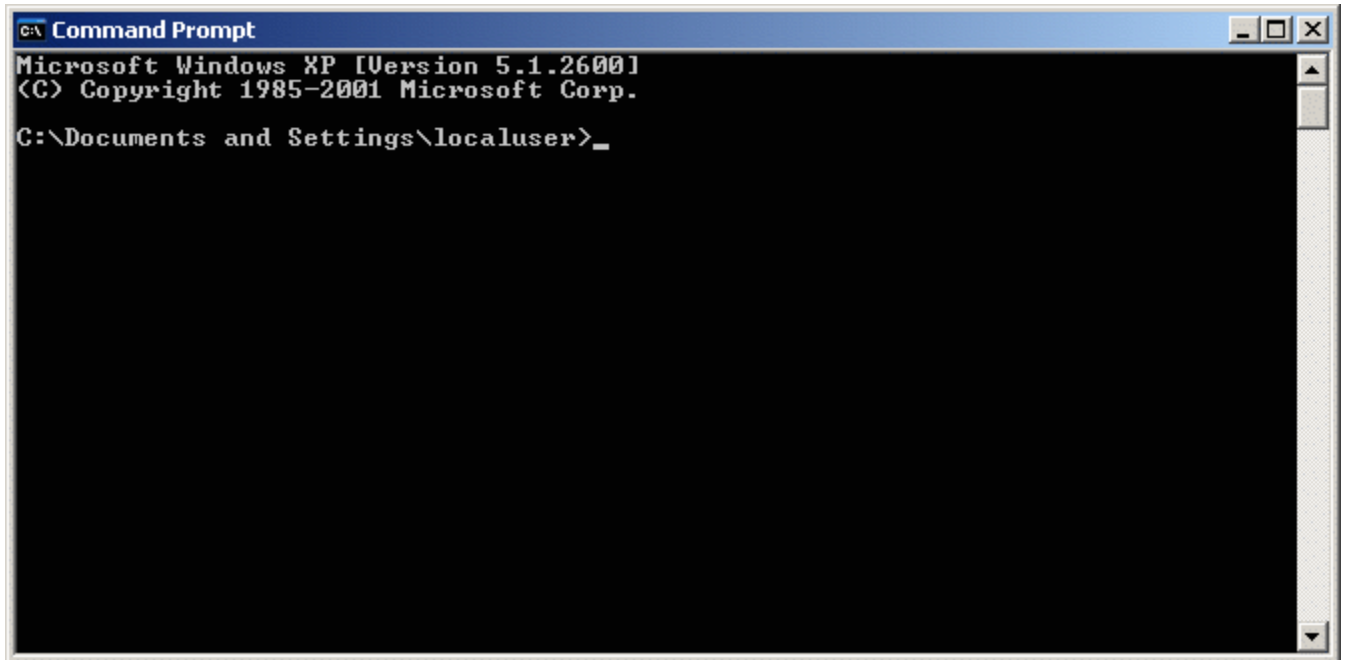


The Save As dialog just before you click **Save**.

Now click **Save**, and exit Notepad.

Compile the Source File into a .class File

Bring up a shell, or "command," window. You can do this from the **Start** menu by choosing **Command Prompt** (Windows XP), or by choosing **Run...** and then entering `cmd`. The shell window should look similar to the following figure.



A shell window.

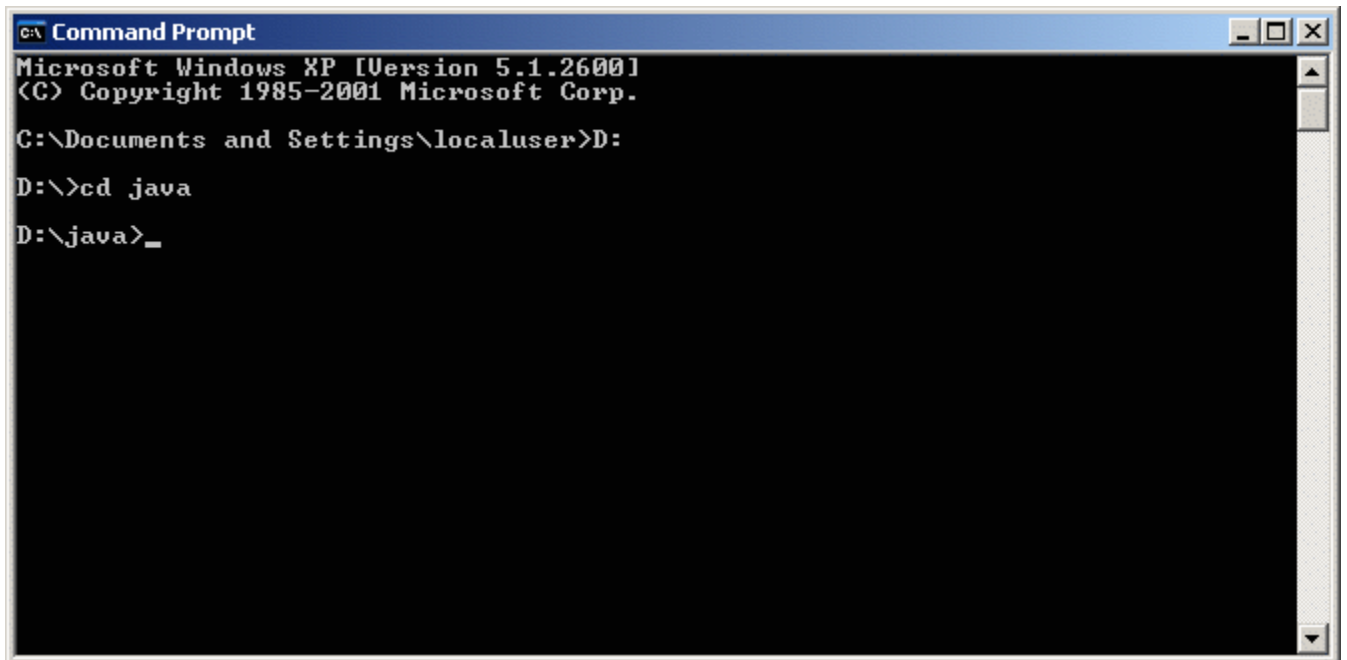
The prompt shows your *current directory*. When you bring up the prompt, your current directory is usually your home directory for Windows XP (as shown in the preceding figure).

To compile your source file, change your current directory to the directory where your file is located. For example, if your source directory is `java` on the C drive, type the following command at the prompt and press **Enter**:

```
cd C:\java
```

Now the prompt should change to `C:\java>`.

Note: To change to a directory on a different drive, you must type an extra command: the name of the drive. For example, to change to the `java` directory on the D drive, you must enter `D:`, as shown in the following figure.

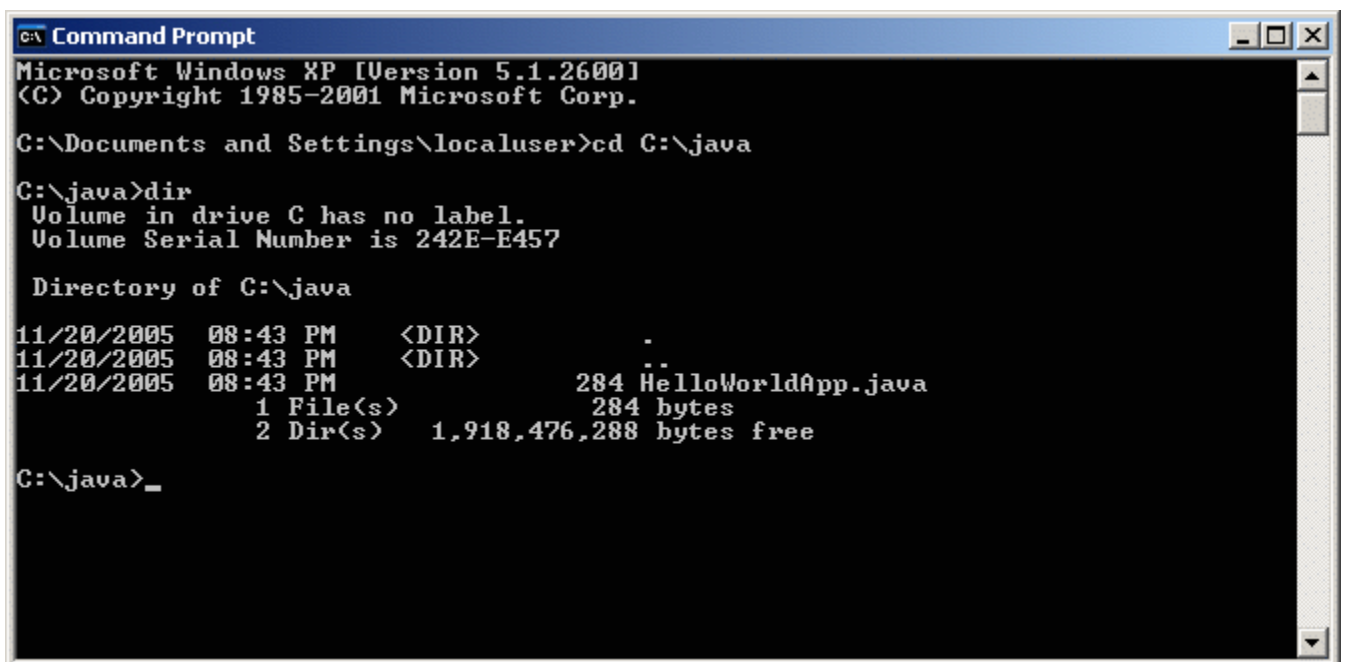


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\localuser>D:
D:\>cd java
D:\java>_
```

Changing directory on an alternate drive.

If you enter `dir` at the prompt, you should see your source file, as the following figure shows.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\localuser>cd C:\java
C:\java>dir
Volume in drive C has no label.
Volume Serial Number is 242E-E457

Directory of C:\java

11/20/2005  08:43 PM    <DIR>          .
11/20/2005  08:43 PM    <DIR>          ..
11/20/2005  08:43 PM                284 HelloWorldApp.java
                1 File(s)                284 bytes
                2 Dir(s)  1,918,476,288 bytes free

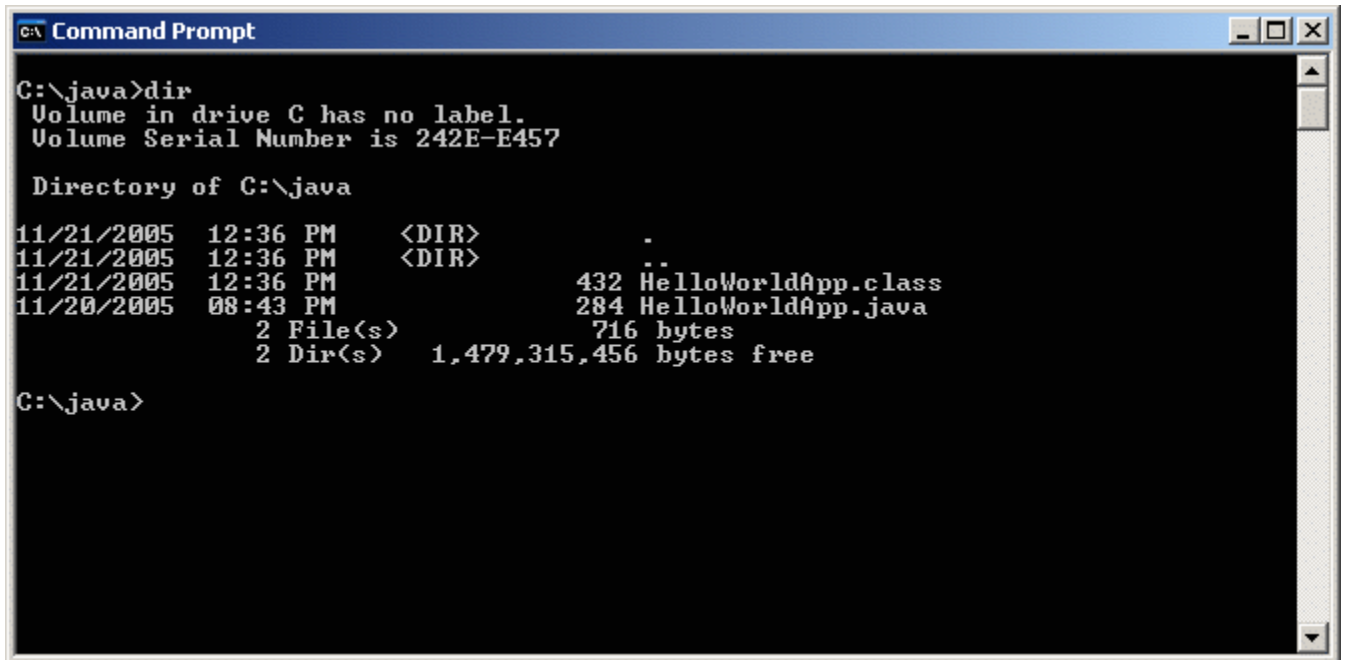
C:\java>_
```

Directory listing showing the `.java` source file.

Now you are ready to compile. At the prompt, type the following command and press **Enter**.

```
javac HelloWorldApp.java
```

The compiler has generated a bytecode file, `HelloWorldApp.class`. At the prompt, type `dir` to see the new file that was generated, as shown in the following figure.



```
C:\java>dir
Volume in drive C has no label.
Volume Serial Number is 242E-E457

Directory of C:\java

11/21/2005  12:36 PM    <DIR>          .
11/21/2005  12:36 PM    <DIR>          ..
11/21/2005  12:36 PM                432 HelloWorldApp.class
11/20/2005   08:43 PM                284 HelloWorldApp.java
                2 File(s)              716 bytes
                2 Dir(s)    1,479,315,456 bytes free

C:\java>
```

Directory listing, showing the generated `.class` file

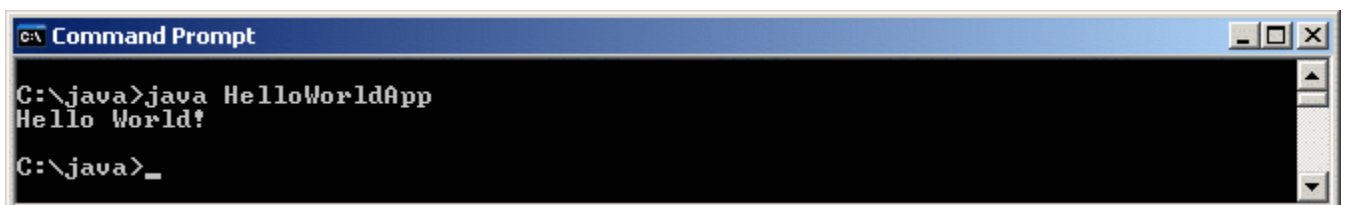
Now that you have a `.class` file, you can run your program.

Run the Program

In the same directory, enter the following command at the prompt:

```
java HelloWorldApp
```

The next figure shows what you should now see:



```
C:\java>java HelloWorldApp
Hello World!

C:\java>
```

The program prints "Hello World!" to the screen.

Congratulations! Your program works!

"Hello World!" for Solaris OS and Linux

Creating Your First Application

Your first application, `HelloWorldApp`, will simply display the greeting "Hello world!". To create this program, you will:

- **Create a source file**

A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files.

- **Compile the source file into a .class file**

The Java programming language *compiler* (`javac`) takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this `.class` file are known as *bytecodes*.

- **Run the program**

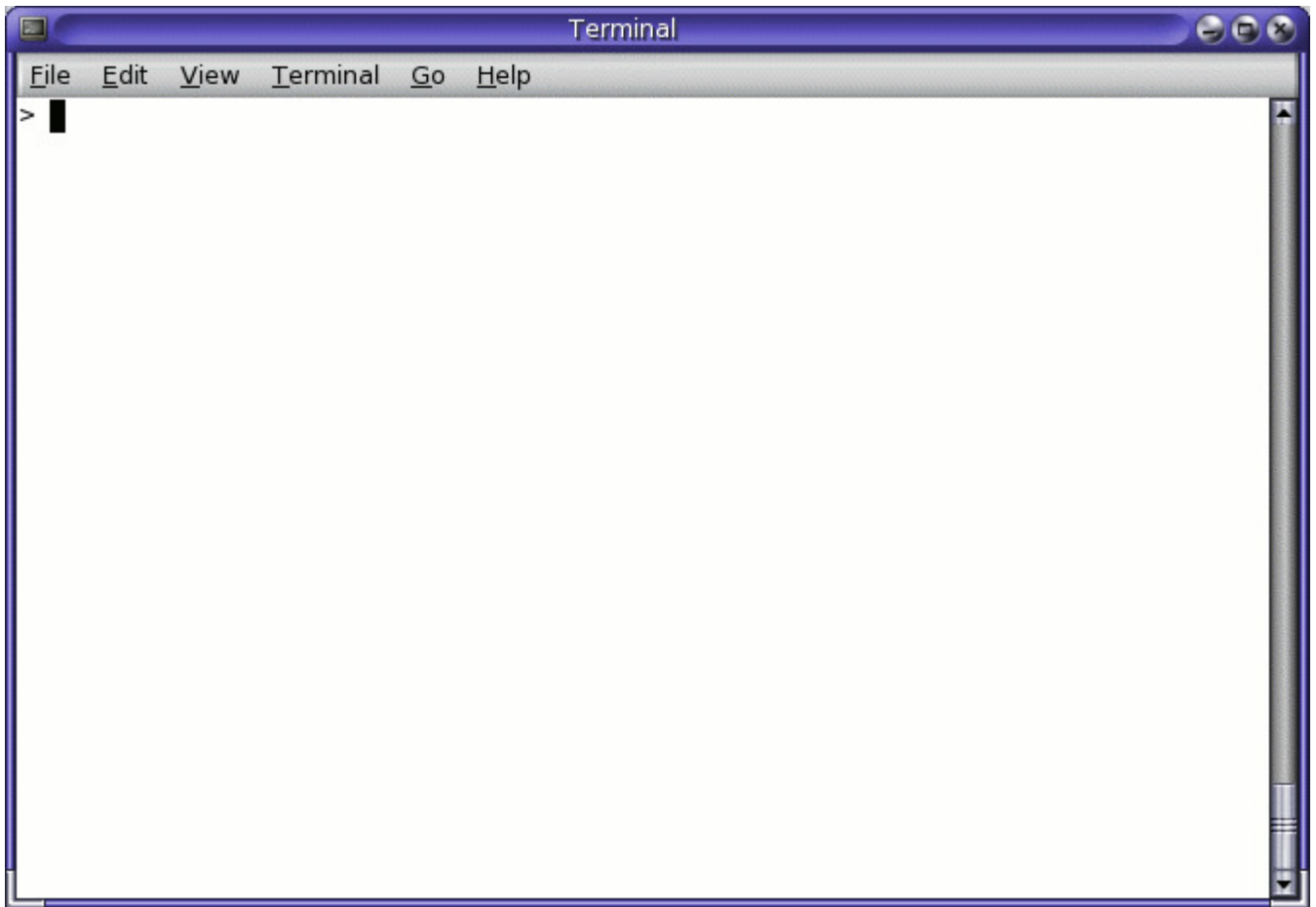
The Java application *launcher tool* (`java`) uses the Java virtual machine to run your application.

Create a Source File

To create a source file, you have two options:

- You can save the file [HelloWorldApp.java](#) on your computer and avoid a lot of typing. Then, you can go straight to [Compile the Source File](#).
- Or, you can use the following (longer) instructions.

First, open a shell, or "terminal," window.



A new terminal window.

When you first bring up the prompt, your *current directory* will usually be your *home directory*. You can change your current directory to your home directory at any time by typing `cd` at the prompt and then pressing **Return**.

The source files you create should be kept in a separate directory. You can create a directory by using the command `mkdir`. For example, to create the directory `java` in your home directory, use the following commands:

```
cd
mkdir java
```

To change your current directory to this new directory, you then enter:

```
cd java
```

Now you can start creating your source file.

Start the Pico editor by typing `pico` at the prompt and pressing **Return**. If the system responds with the message `pico: command not found`, then Pico is most likely unavailable. Consult your system administrator for more information, or use another editor.

When you start Pico, it'll display a new, blank *buffer*. This is the area in which you will type your code.

Type the following code into the new buffer:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Be Careful When You Type



Type all code, commands, and file names exactly as shown. Both the compiler (`javac`) and launcher tool (`java`) are *case-sensitive*, so you must capitalize consistently.

HelloWorldApp ~~h~~elloworldapp

Save the code in a file with the name `HelloWorldApp.java`. In the Pico editor, you do this by typing **Ctrl-O** and then, at the bottom where you see the prompt `File Name to write:`, entering the directory in which you wish to create the file, followed by `HelloWorldApp.java`. For example, if you wish to save `HelloWorldApp.java` in the directory `/home/jdoe/java`, then you type `/home/jdoe/java/HelloWorldApp.java` and press **Return**.

You can type **Ctrl-X** to exit Pico.

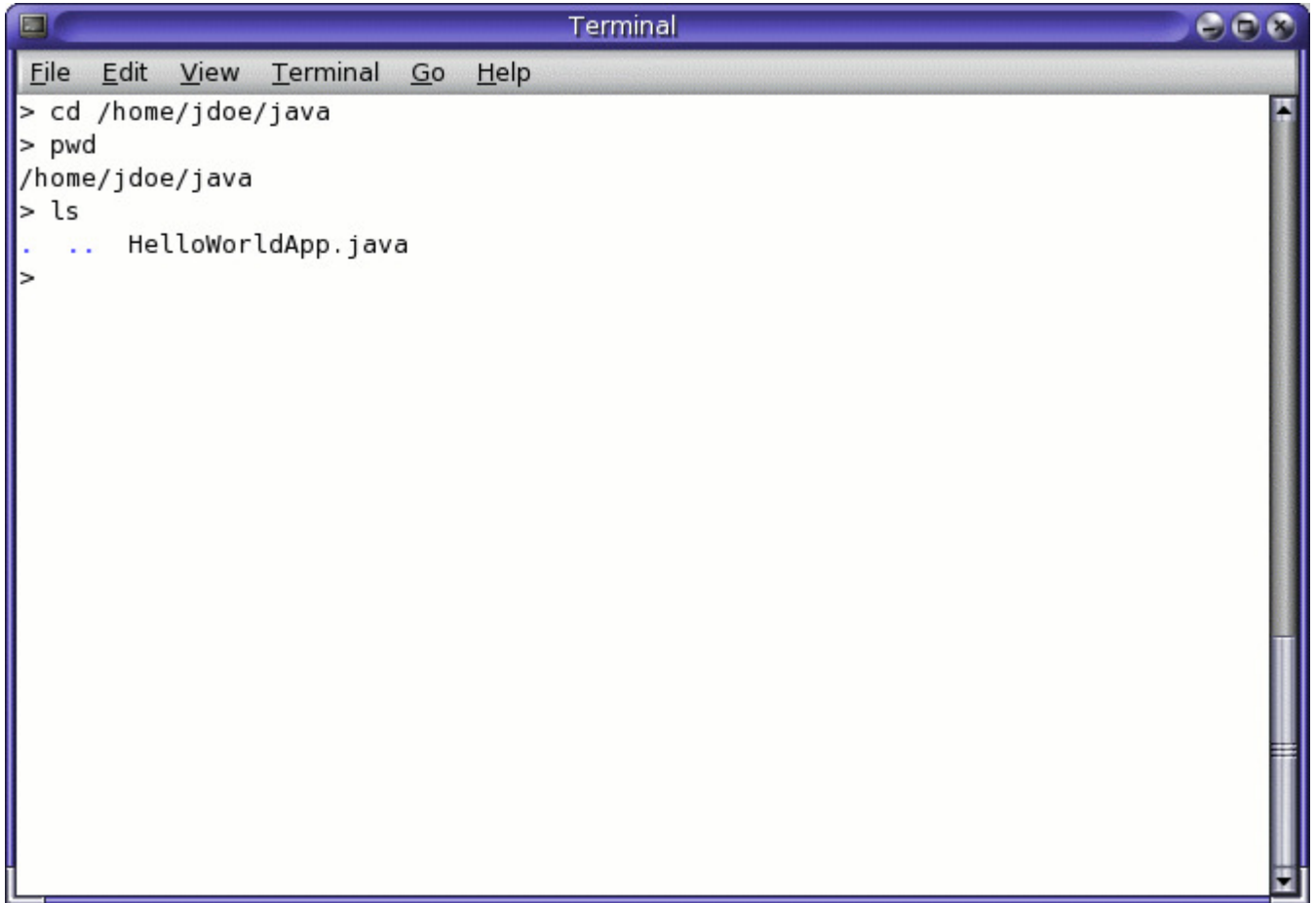
Compile the Source File into a `.class` File

Bring up another shell window. To compile your source file, change your current directory to the directory where your file is located. For example, if your source directory is `/home/jdoe/java`, type the following command at the prompt and press **Return**:

```
cd /home/jdoe/java
```

If you enter `pwd` at the prompt, you should see the current directory, which in this example has been changed to `/home/jdoe/java`.

If you enter `ls` at the prompt, you should see your file.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal content shows the following sequence of commands and their outputs:

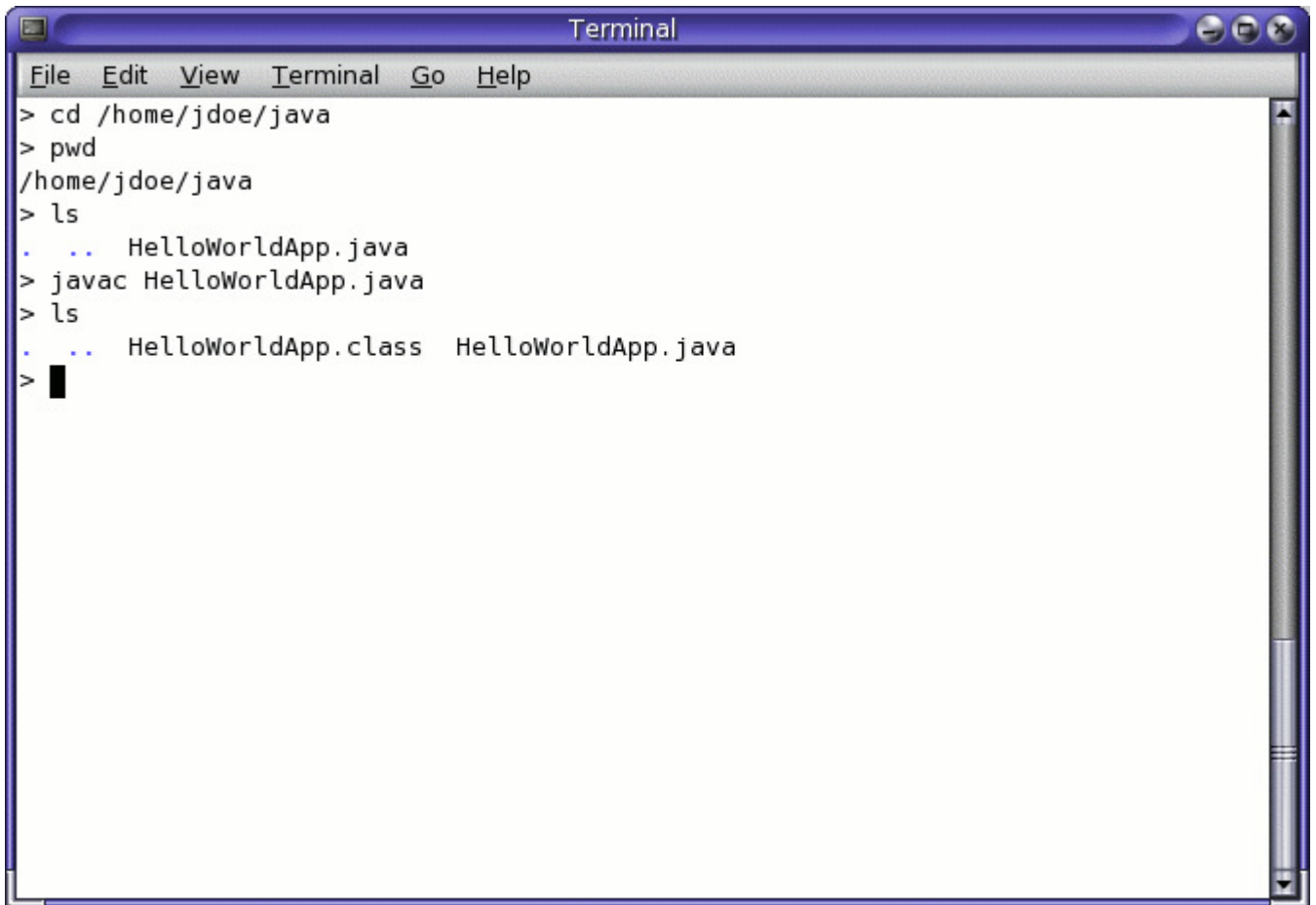
```
> cd /home/jdoe/java
> pwd
/home/jdoe/java
> ls
.  ..  HelloWorldApp.java
>
```

Results of the `ls` command, showing the `.java` source file.

Now are ready to compile the source file. At the prompt, type the following command and press **Return**.

```
javac HelloWorldApp.java
```

The compiler has generated a bytecode file, `HelloWorldApp.class`. At the prompt, type `ls` to see the new file that was generated: the following figure.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal content shows the following commands and output:

```
> cd /home/jdoe/java
> pwd
/home/jdoe/java
> ls
.  ..  HelloWorldApp.java
> javac HelloWorldApp.java
> ls
.  ..  HelloWorldApp.class  HelloWorldApp.java
> █
```

Results of the `ls` command, showing the generated `.class` file.

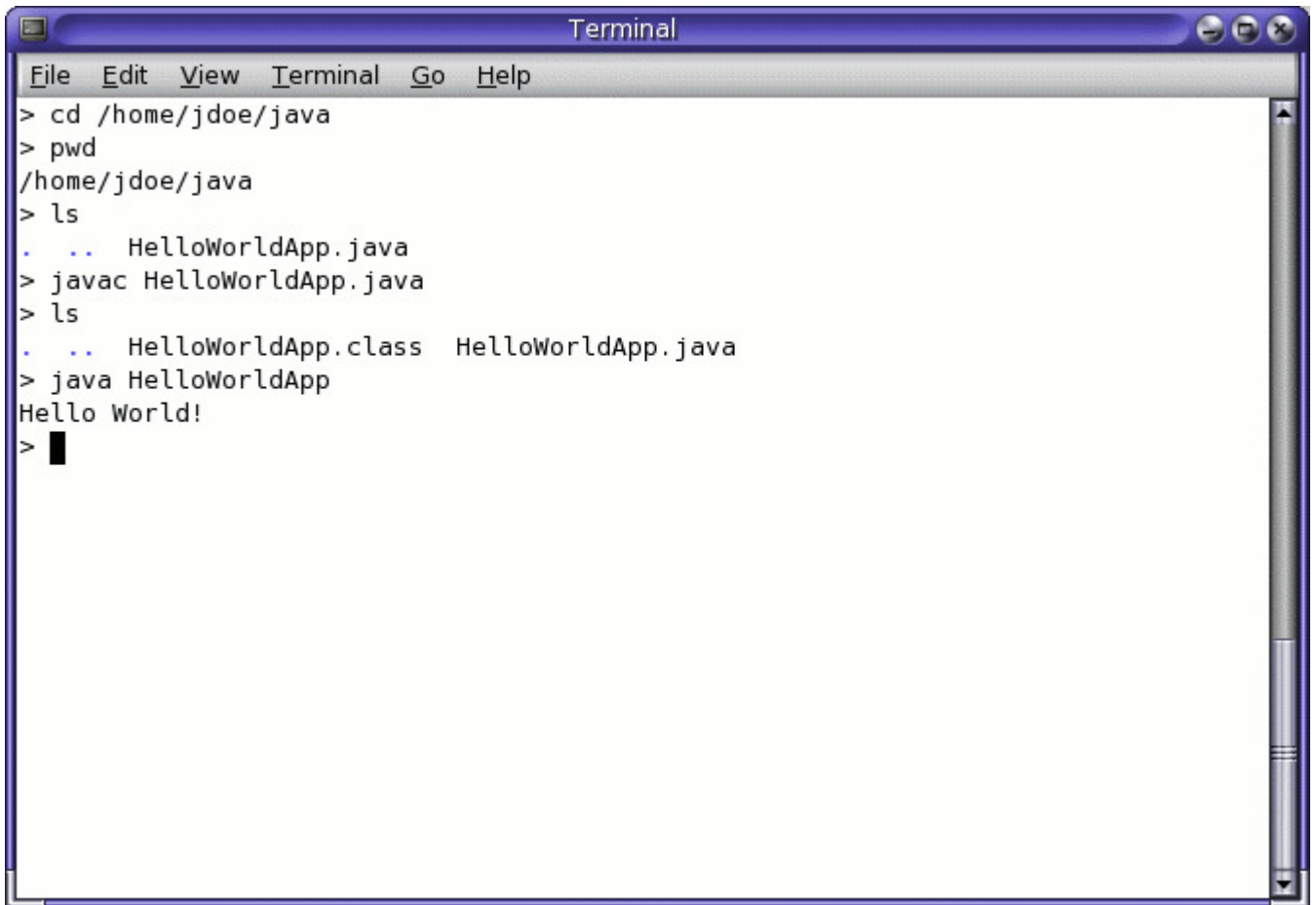
Now that you have a `.class` file, you can run your program.

Run the Program

In the same directory, enter at the prompt:

```
java HelloWorldApp
```

The next figure shows what you should now see.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal content shows a series of commands and their outputs:

```
> cd /home/jdoe/java
> pwd
/home/jdoe/java
> ls
.  ..  HelloWorldApp.java
> javac HelloWorldApp.java
> ls
.  ..  HelloWorldApp.class  HelloWorldApp.java
> java HelloWorldApp
Hello World!
> █
```

The output prints "Hello World!" to the screen.

Congratulations! Your program works!