Java Tutorials

Updated for Java SE 8

ORACLE

Java Tutorials

The Java Tutorials are a one stop shop for learning the Java language. The tutorials are practical guides for programmers who want to use the Java programming language to create applets and applications. They also serve as a reference for the experienced Java programmer.

Groups of related lessons are organized into "trails"; for example, the "Getting Started" trail, or the "Learning the Java Language" trail. This ebook contains a single trail. You can download other trails via the link on the Java Tutorials home page.

To read an ebook file that you have downloaded, transfer the document to your device.

We would love to hear what you think of this ebook, including any problems that you find. Please send us your feedback.

Legal Notices

This tutorial is a guide to developing applications for the Java Platform, Standard Edition and contains documentation (Tutorial) and sample code. The sample code made available with this Tutorial is licensed separately to you by Oracle under the Berkeley license. If you download any such sample code, you agree to the terms of the Berkeley license.

This Tutorial is provided to you by Oracle under the following license terms containing restrictions on use and disclosure and is protected by intellectual property laws. Oracle grants to you a limited, non-exclusive license to use this Tutorial for information purposes only, as an aid to learning about the Java SE platform. Except as expressly permitted in these license terms, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means this Tutorial. Reverse engineering, disassembly, or decompilation of this Tutorial is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If the Tutorial is licensed on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This Tutorial is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this Tutorial in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use.

THE TUTORIAL IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND. ORACLE FURTHER DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT.

IN NO EVENT SHALL ORACLE BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, DATA OR DATA USE, INCURRED BY YOU OR ANY THIRD PARTY, WHETHER IN AN ACTION IN CONTRACT OR TORT, EVEN IF ORACLE HAS BEEN

ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. ORACLES ENTIRE LIABILITY FOR DAMAGES HEREUNDER SHALL IN NO EVENT EXCEED ONE THOUSAND DOLLARS (U.S. $1,000).

No Technical Support

Oracles technical support organization will not provide technical support, phone support, or updates to you.

The sample code and Tutorial may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Supported Platforms

The `mobi` file format works best on the following devices:

- Kindle Fire
- Kindle DX

The `ePub` file format works best on the following devices:

- iPad
- Nook
- Other eReaders that support the `ePub` format.

For best results when viewing preformatted code blocks, adjust the landscape/portrait orientation and font size of your device to enable the maximum possible viewing area.

Trail: Getting Started

This trail provides everything you'll need to know about getting started with the Java programming language.

 **The Java Technology Phenomenon** Provides an overview of Java technology as a whole. It discusses both the Java programming language and platform, providing a broad overview of what this technology can do and how it will make your life easier.

 **The "Hello World!" Application** This hands-on approach describes what to download, what to install, and what to type, for creating a simple "Hello World!" application. It provides separate instructions for the NetBeans integrated development environment (NetBeans IDE), Microsoft Windows, Solaris Operating System (Solaris OS), Linux, and Mac users.

 **A Closer Look at "Hello World!"** Discusses the "Hello World!" application, describing each section of code in detail. It covers source code comments, the `HelloWorldApp` class definition block, and the `main` method.

 **Common Problems (and Their Solutions)** This is the place to go if you have trouble compiling or running the programs in this trail.

Trail: Getting Started: Table of Contents

Lesson: The Java Technology Phenomenon

Talk about Java technology seems to be everywhere, but what exactly is it? The following sections explain how Java technology is both a programming language and a platform, and provide an overview of what this technology can do for you.

Note: See online version of topics in this ebook to download complete source code.

About the Java Technology

Java technology is both a programming language and a platform.

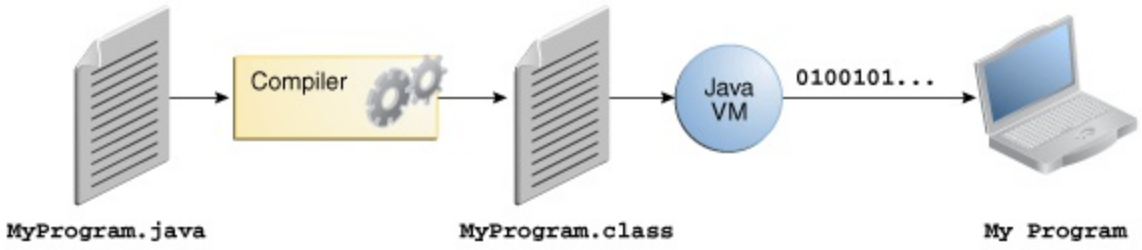
The Java Programming Language

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

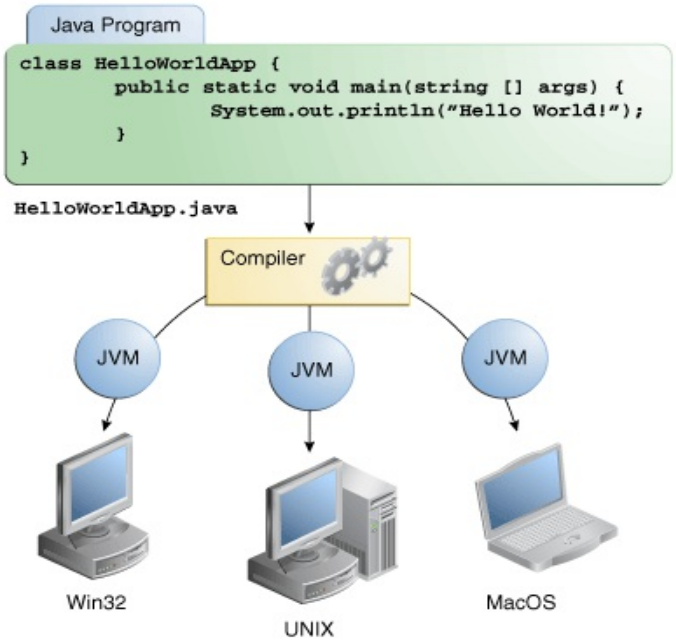
Each of the preceding buzzwords is explained in *The Java Language Environment* , a white paper written by James Gosling and Henry McGilton.

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 Machine[1] (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 SE HotSpot at a Glance, 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

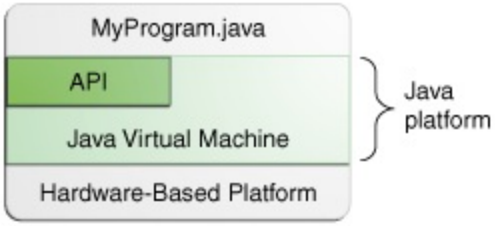
A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform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

The terms"Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform.

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 Platform Standard Edition 7 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 JavaFX, 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 (JNDI) 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.

Lesson: The "Hello World!" Application

The sections listed below provide detailed instructions for compiling and running a simple "Hello World!" application. The first section provides information on getting started with the NetBeans IDE, an integrated development environment that greatly simplifies the software development process. The NetBeans IDE runs on all of the platforms listed below. The remaining sections provide platform-specific instructions for getting started without an integrated development environment. If you run into problems, be sure to consult the common problems section; it provides solutions for many issues encountered by new users.

"Hello World!" for the NetBeans IDE These instructions are for users of the NetBeans IDE. The NetBeans IDE runs on the Java platform, which means that you can use it with any operating system for which there is a JDK 7 available. These operating systems include Microsoft Windows, Solaris OS, Linux, and Mac OS X. We recommend using the NetBeans IDE instead of the command line whenever possible.

"Hello World!" for Microsoft Windows These command-line instructions are for users of Windows XP Professional, Windows XP Home, Windows Server 2003, Windows 2000 Professional, and Windows Vista.

"Hello World!" for Solaris OS and Linux These command-line instructions are for users of Solaris OS and Linux. Common Problems (and Their Solutions) Consult this page if you have problems compiling or running your application.

Note: See online version of topics in this ebook to download complete source code.

"Hello World!" for the NetBeans IDE

It's time to write your first application! These detailed instructions are for users of the NetBeans IDE. The NetBeans IDE runs on the Java platform, which means that you can use it with any operating system for which there is a JDK available. These operating systems include Microsoft Windows, Solaris OS, Linux, and Mac OS X.

A Checklist 

To write your first program, you'll need:

1. The Java SE Development Kit (JDK 7 has been selected in this example)

   - For Microsoft Windows, Solaris OS, and Linux: Java SE Downloads Index page
   - For Mac OS X: developer.apple.com

2. The NetBeans IDE

   - For all platforms: NetBeans IDE Downloads Index page

Creating Your First Application

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

- Create an IDE project
  When you create an IDE project, you create an environment in which to build and run your applications. Using IDE projects eliminates configuration issues normally associated with developing on the command line. You can build or run your application by choosing a single menu item within the IDE.
- Add code to the generated source file
  A source file contains code, written in the Java programming language, that you and other programmers can understand. As part of creating an IDE project, a skeleton source file will be

automatically generated. You will then modify the source file to add the "Hello World!" message.

- Compile the source file into a .class file
  The IDE invokes the Java programming language *compiler* (`javac`), which 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 IDE invokes the Java application *launcher tool* (`java`), which uses the Java virtual machine to run your application.

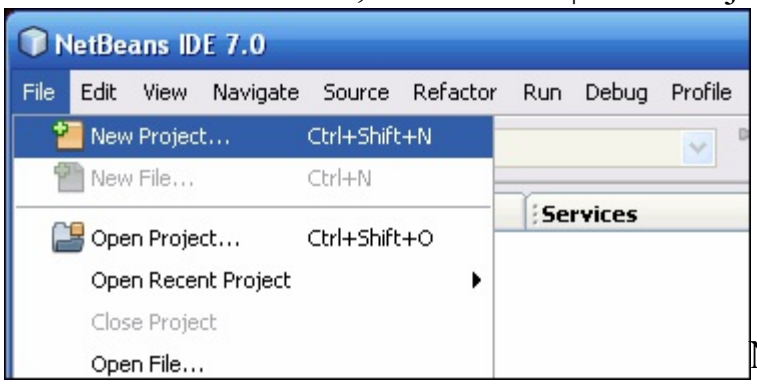
Create an IDE Project

To create an IDE project:

1. Launch the NetBeans IDE.

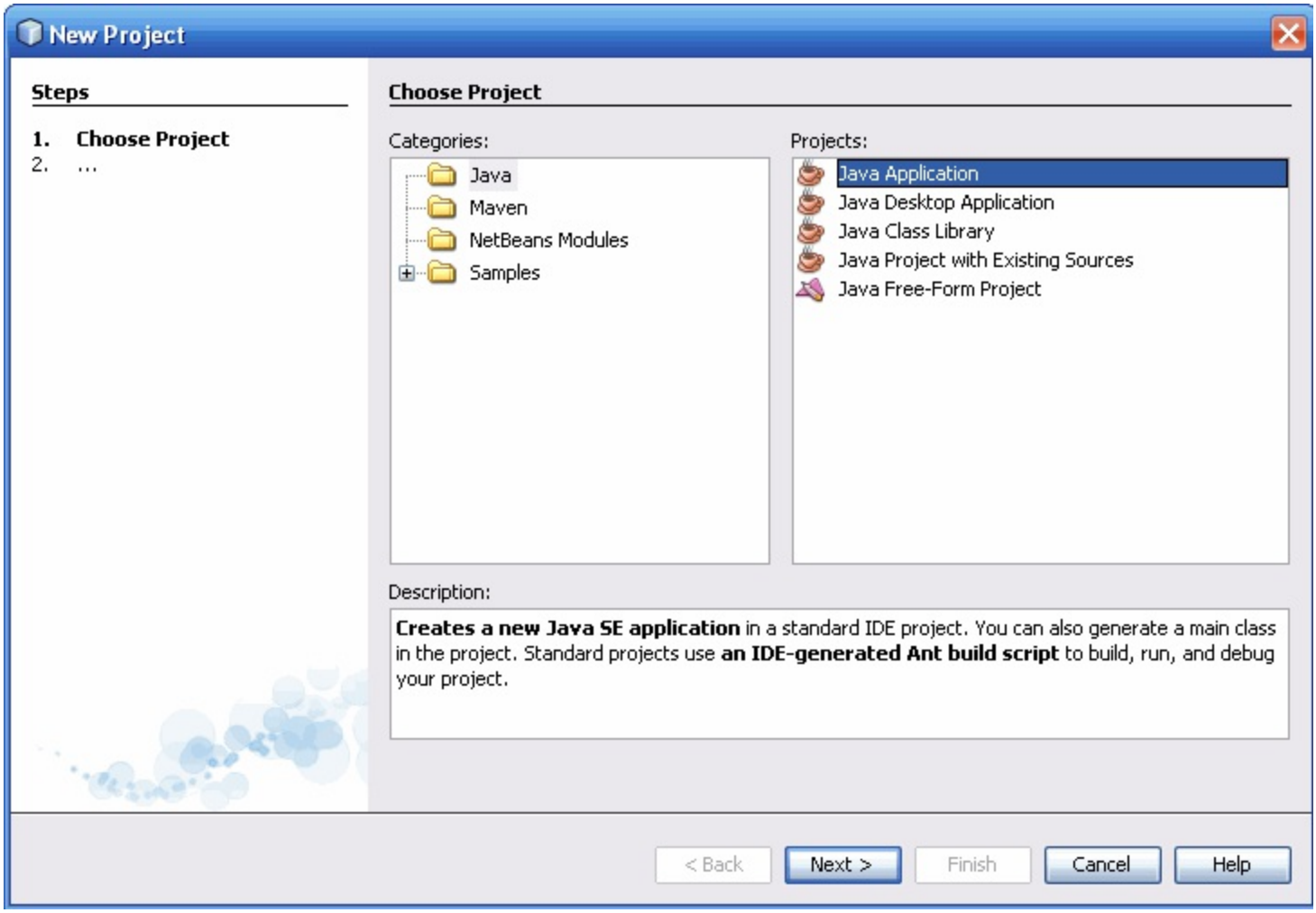
- On Microsoft Windows systems, you can use the NetBeans IDE item in the Start menu.
- On Solaris OS and Linux systems, you execute the IDE launcher script by navigating to the IDE's `bin` directory and typing `./netbeans`.
- On Mac OS X systems, click the NetBeans IDE application icon.

2. In the NetBeans IDE, choose File | New Project.



NetBeans IDE with the File | New Project menu item selected.

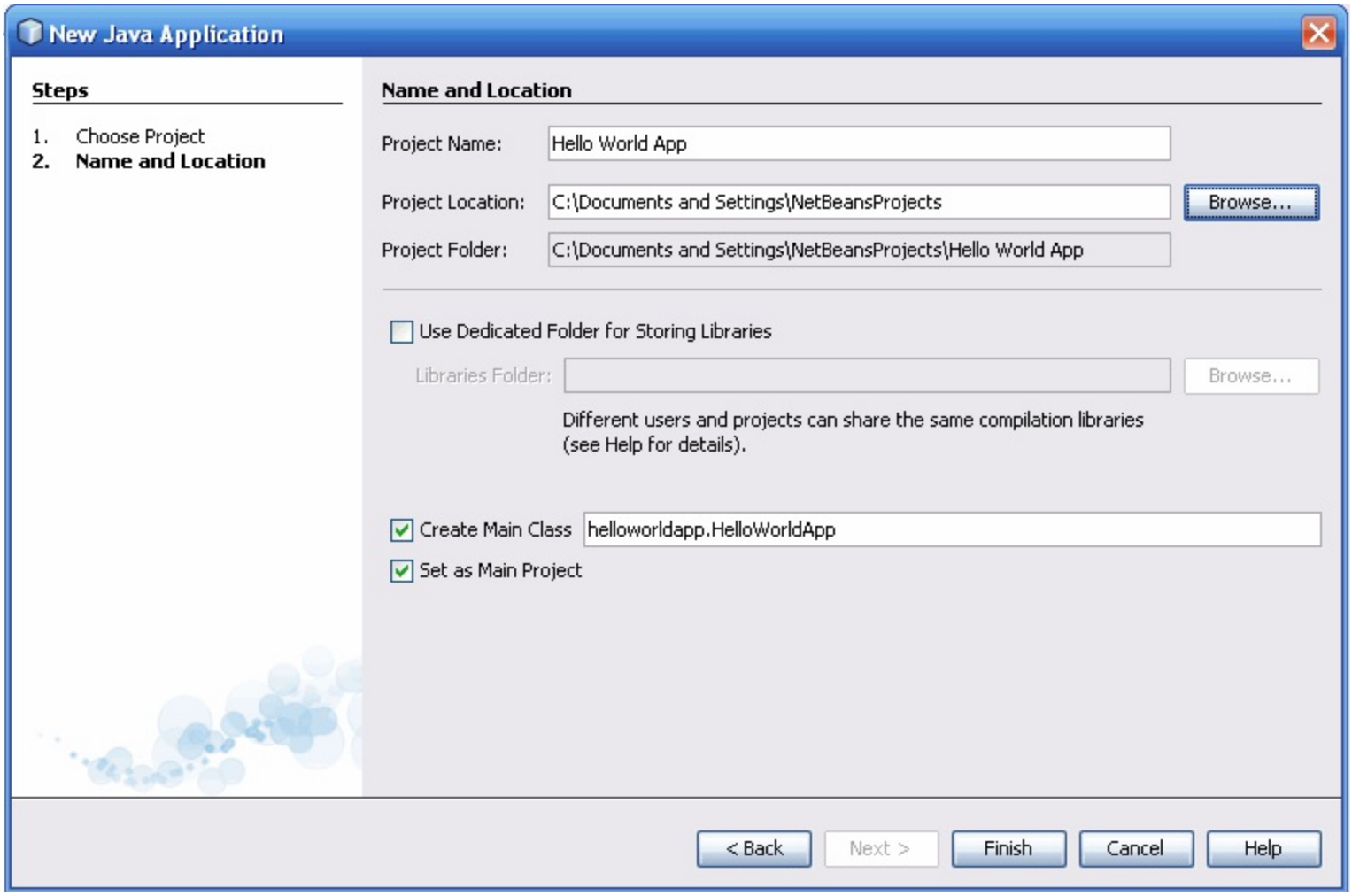
3. In the New Project wizard, expand the Java category and select Java Application as shown in the following figure:



NetBeans IDE, New Project wizard, Choose Project page.

4. In the Name and Location page of the wizard, do the following (as shown in the figure below):

- In the Project Name field, type `Hello World App`.
- In the Create Main Class field, type `helloworldapp.HelloWorldApp`.
- Leave the Set as Main Project checkbox selected.

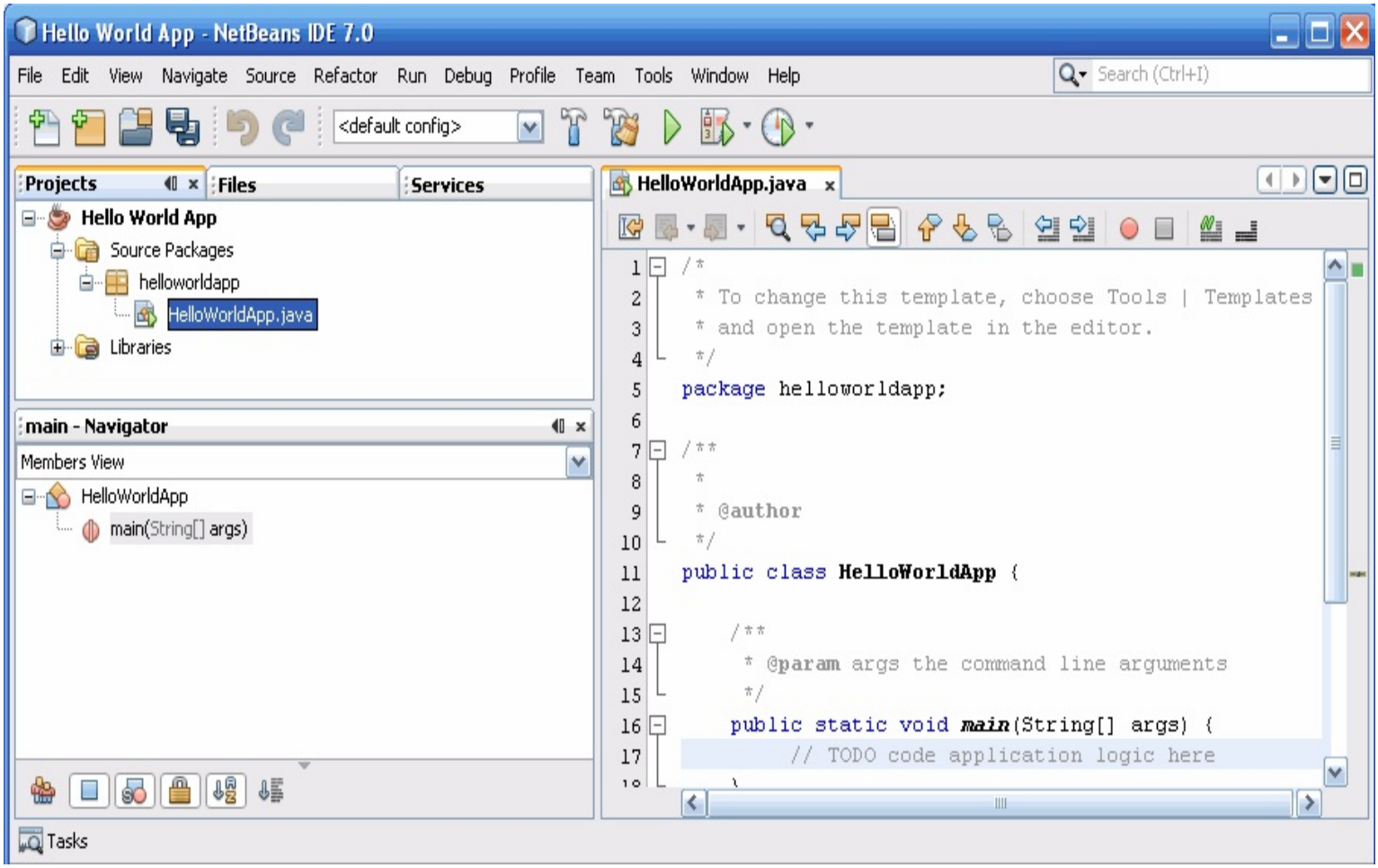


NetBeans IDE, New Project wizard, Name and Location page.

5. Click Finish.

The project is created and opened in the IDE. You should see the following components:

- The Projects window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.
- The Source Editor window with a file called `HelloWorldApp` open.
- The Navigator window, which you can use to quickly navigate between elements within the selected class.



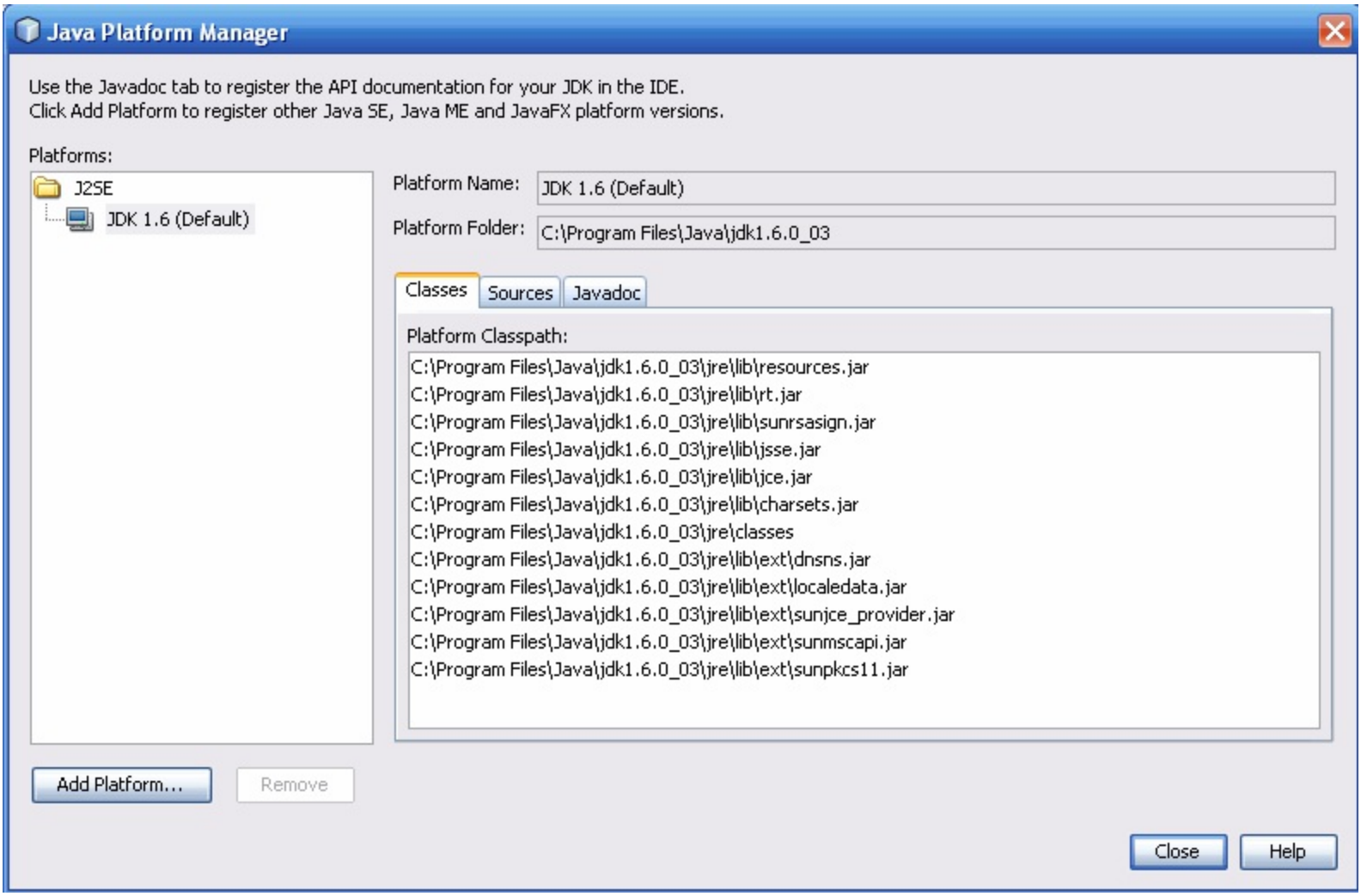
NetBeans IDE with the HelloWorldApp project open.

Add JDK 7 to the Platform List (if necessary)

It may be necessary to add JDK 7 to the IDE's list of available platforms. To do this, choose Tools | Java Platforms as shown in the following figure:

Selecting the Java Platform Manager from the Tools Menu

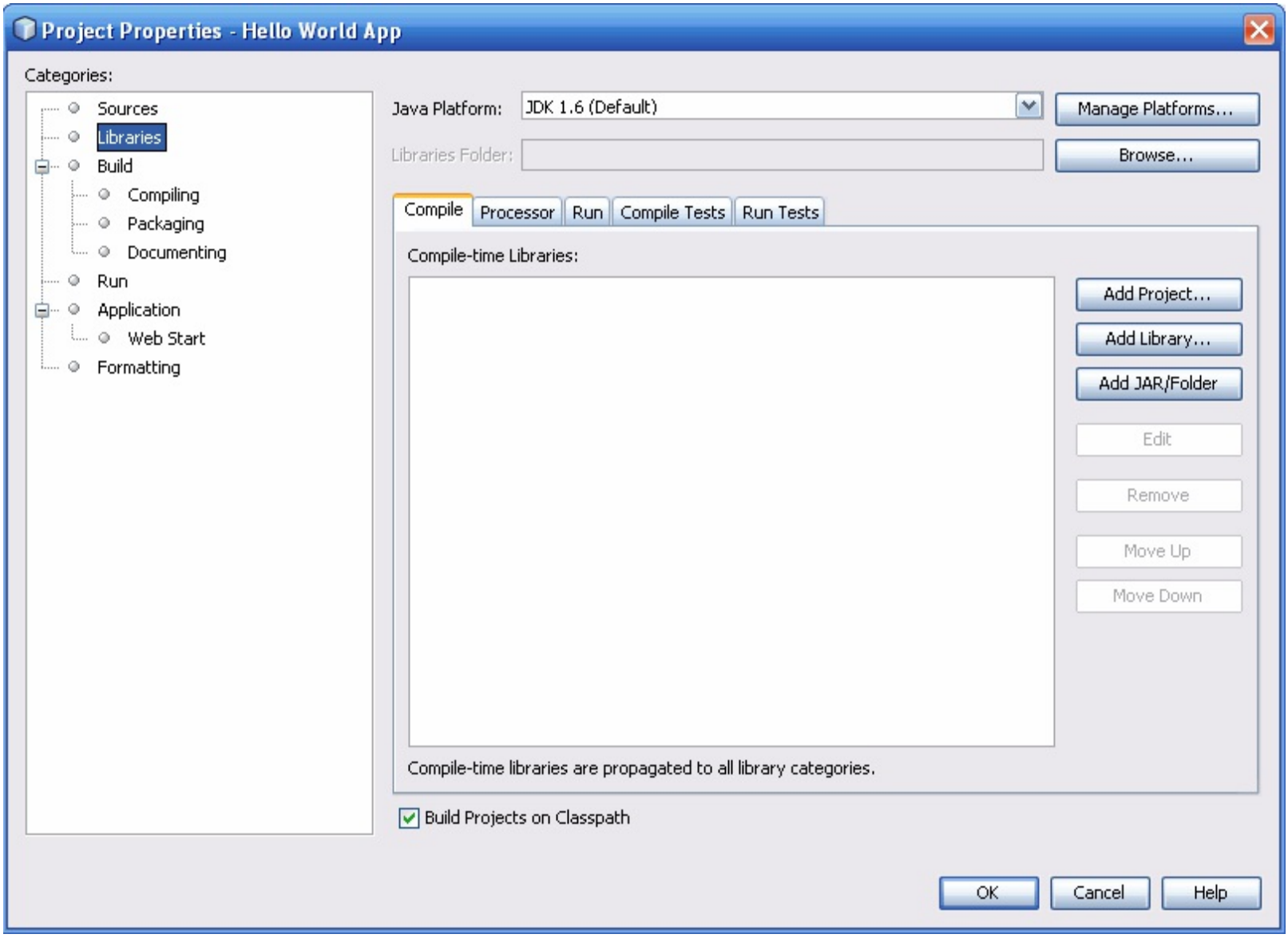
If you don't see JDK 7 (which might appear as 1.7 or 1.7.0) in the list of installed platforms, click "Add Platform", navigate to your JDK 7 install directory, and click "Finish". You should now see this newly added platform:



The Java Platform Manager

To set this JDK as the default for all projects, you can run the IDE with the `--jdkhome` switch on the command line, or by entering the path to the JDK in the `netbeans_j2sdkhome` property of your `INSTALLATION_DIRECTORY/etc/netbeans.conf` file.

To specify this JDK for the current project only, select Hello World App in the Projects pane, choose File | Project Properties (Hello World App), click on Libraries, then select JDK 7 under the Java Platform pulldown menu. You should see a screen similar to the following:



The IDE is now configured for JDK 7.

Add Code to the Generated Source File

When you created this project, you left the Create Main Class checkbox selected in the New Project wizard. The IDE has therefore created a skeleton class for you. You can add the "Hello World!" message to the skeleton code by replacing the line:

```
// TODO code application logic here
```

with the line:

```
System.out.println("Hello World!"); // Display the string.
```

Optionally, you can replace these four lines of generated code:

```
/**
 *
```

```
 * @author
 */
```

with these lines:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
```

These four lines are a code comment and do not affect how the program runs. Later sections of this tutorial explain the use and format of code comments.

Be Careful When You Type 

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

`HelloWorldApp` is *not* the same as `helloworldapp`.

Save your changes by choosing File | Save.

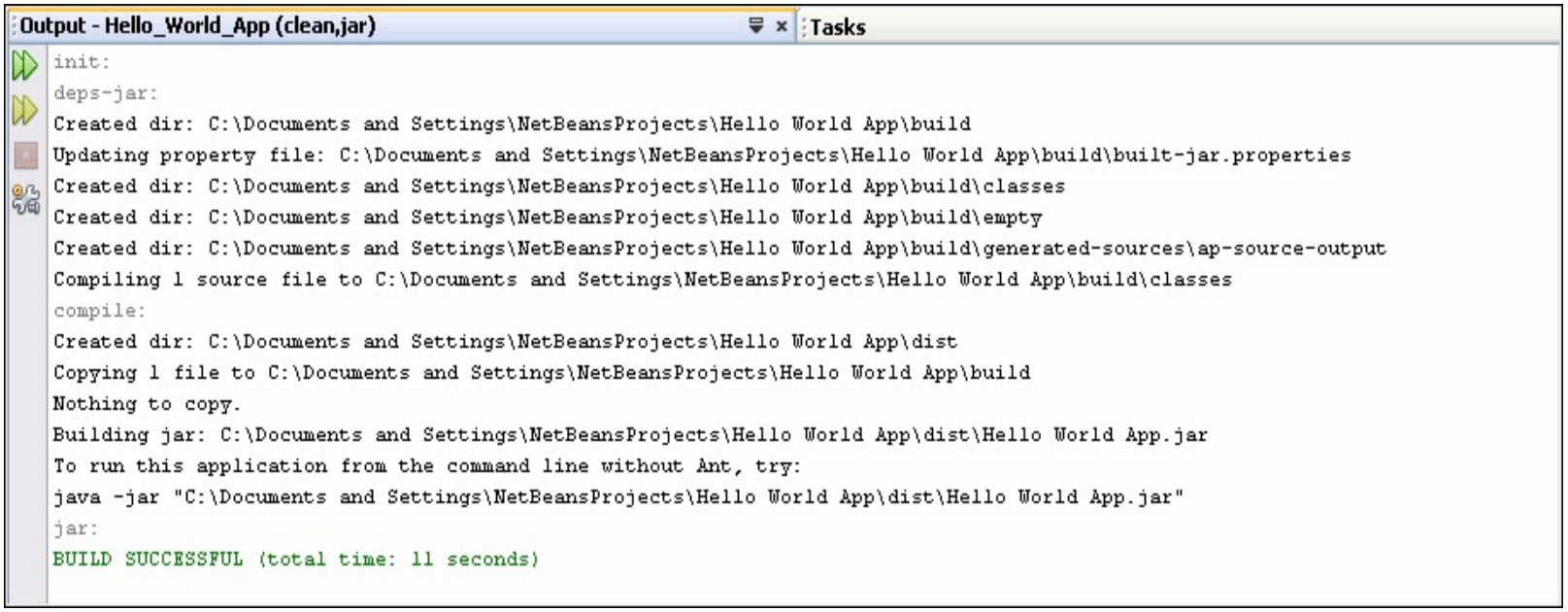
The file should look something like the following:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package helloworldapp;

/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
public class HelloWorldApp {


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }

}
```

Compile the Source File into a .class File

To compile your source file, choose Run | Build Main Project from the IDE's main menu.

The Output window opens and displays output similar to what you see in the following figure:

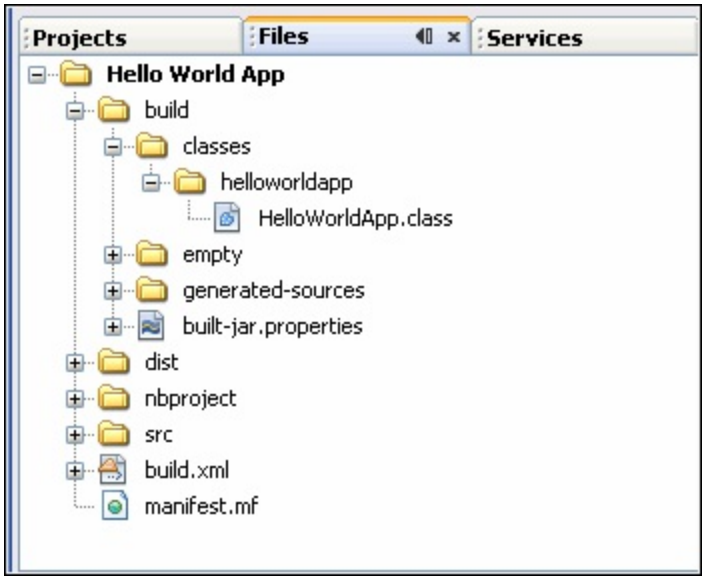


Output window showing results of building the HelloWorld project.

If the build output concludes with the statement `BUILD SUCCESSFUL`, congratulations! You have successfully compiled your program!

If the build output concludes with the statement `BUILD FAILED`, you probably have a syntax error in your code. Errors are reported in the Output window as hyper-linked text. You double-click such a hyper-link to navigate to the source of an error. You can then fix the error and once again choose Run | Build Main Project.

When you build the project, the bytecode file `HelloWorldApp.class` is generated. You can see where the new file is generated by opening the Files window and expanding the `HelloWorldApp/build/classes/helloworldapp` node as shown in the following figure.



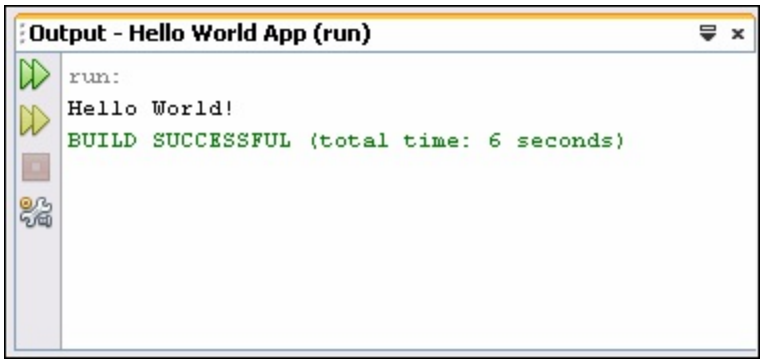
Files window, showing the generated .class file.

Now that you have built the project, you can run your program.

Run the Program

From the IDE's menu bar, choose Run | Run Main Project.

The next figure shows what you should now see.



The program prints "Hello World!" to the Output window (along with other output from the build script).

Congratulations! Your program works!

Continuing the Tutorial with the NetBeans IDE

The next few pages of the tutorial will explain the code in this simple application. After that, the lessons go deeper into core language features and provide many more examples. Although the rest of the tutorial does not give specific instructions about using the NetBeans IDE, you can easily use the IDE to write and run the sample code. The following are some tips on using the IDE and explanations of some IDE behavior that you are likely to see:

- Once you have created a project in the IDE, you can add files to the project using the New File wizard. Choose File | New File, and then select a template in the wizard, such as the Empty Java File template.
- You can compile and run an individual file (as opposed to a whole project) using the IDE's Compile File (F9) and Run File (Shift-F6) commands. If you use the Run Main Project command, the IDE will run the file that the IDE associates as the main class of the main project. Therefore, if you create an additional class in your HelloWorldApp project and then try to run that file with the Run Main Project command, the IDE will run the `HelloWorldApp` file instead.
- You might want to create separate IDE projects for sample applications that include more than one source file.
- As you are typing in the IDE, a code completion box might periodically appear. You can either ignore the code completion box and keep typing, or you can select one of the suggested expressions. If you would prefer not to have the code completion box automatically appear, you can turn off the feature. Choose Tools | Options | Editor, click the Code Completion tab and clear the Auto Popup Completion Window checkbox.
- If you want to rename the node for a source file in the Projects window, choose Refactor from IDE's main menu. The IDE prompts you with the Rename dialog box to lead you through the options of renaming the class and the updating of code that refers to that class. Make the changes and click Refactor to apply the changes. This sequence of clicks might seem unnecessary if you have just a single class in your project, but it is very useful when your changes affect other parts of your code in larger projects.

- For a more thorough guide to the features of the NetBeans IDE, see the NetBeans Documentation page.

"Hello World!" for Microsoft Windows

It's time to write your first application! The following instructions are for users of Windows XP Professional, Windows XP Home, Windows Server 2003, Windows 2000 Professional, and Windows Vista. Instructions for other platforms are in "Hello World!" for Solaris OS and Linux and "Hello World!" for the NetBeans IDE.

If you encounter problems with the instructions on this page, consult the Common Problems (and Their Solutions).

A Checklist 

To write your first program, you'll need:

1. The Java SE Development Kit 7 (JDK 7)
You can download the Windows version now. (Make sure you download the **JDK**, *not* the JRE.)
Consult the installation instructions.
2. A text editor
In this example, we'll use Notepad, a simple editor included with the Windows platforms. You can easily adapt these instructions if you use a different text editor.

These two items are all you'll need to write your first application.

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 

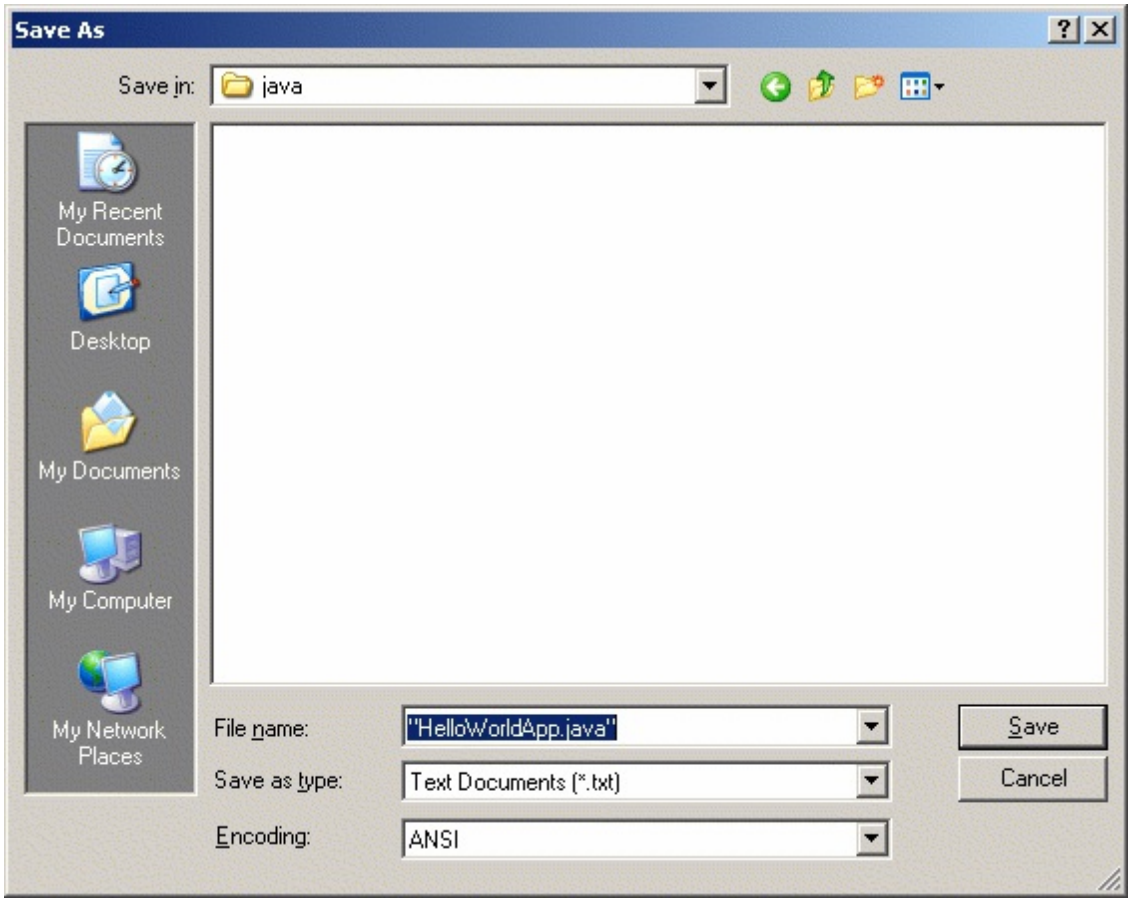
Note: Type all code, commands, and file names exactly as shown. Both the compiler (`javac`) and launcher (`java`) are *case-sensitive*, so you must capitalize consistently.

`HelloWorldApp` is *not* the same as `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.

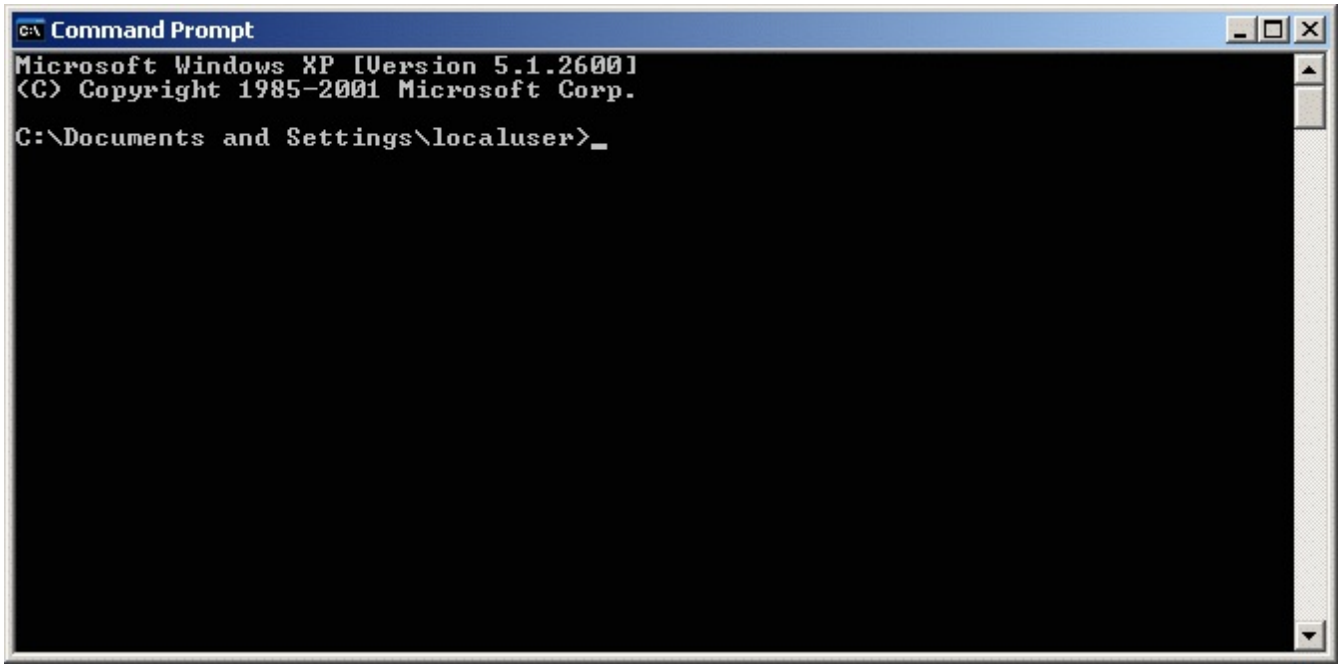


The Save As dialog just before you click **Save**.

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.

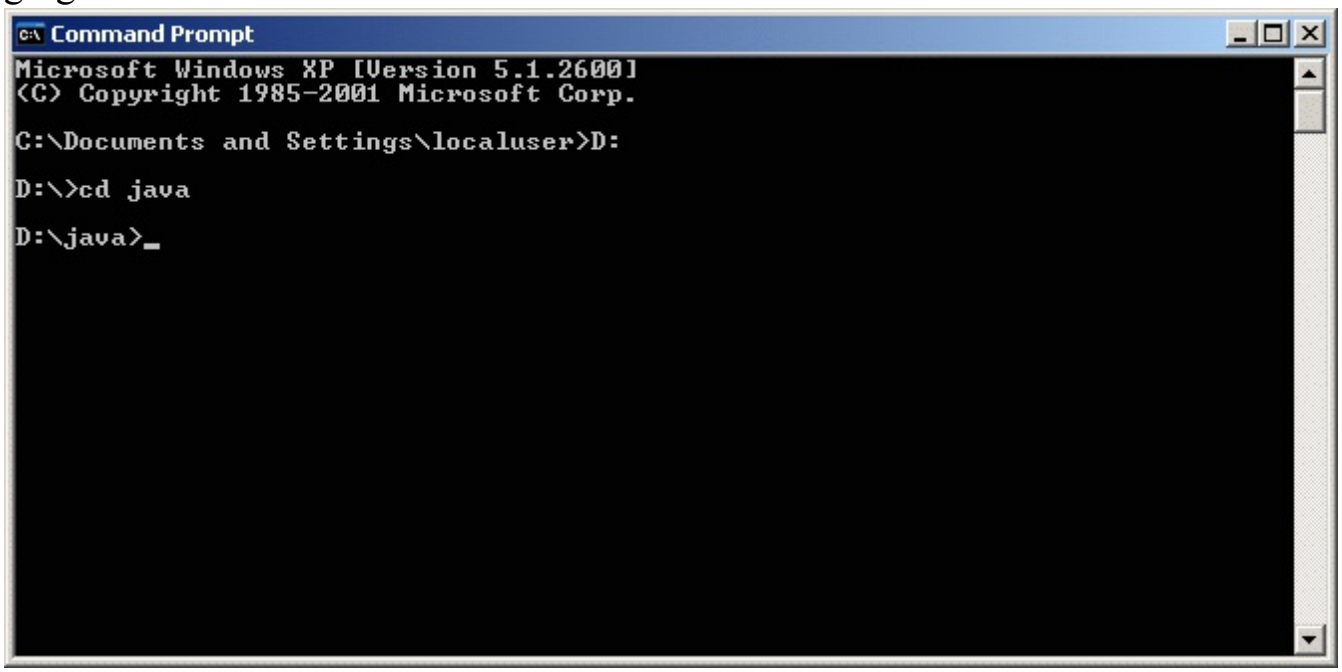
The prompt shows your *current directory*. When you bring up the prompt, your current directory is usually your home directory for Windows XP (as shown in the preceding figure.

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
```

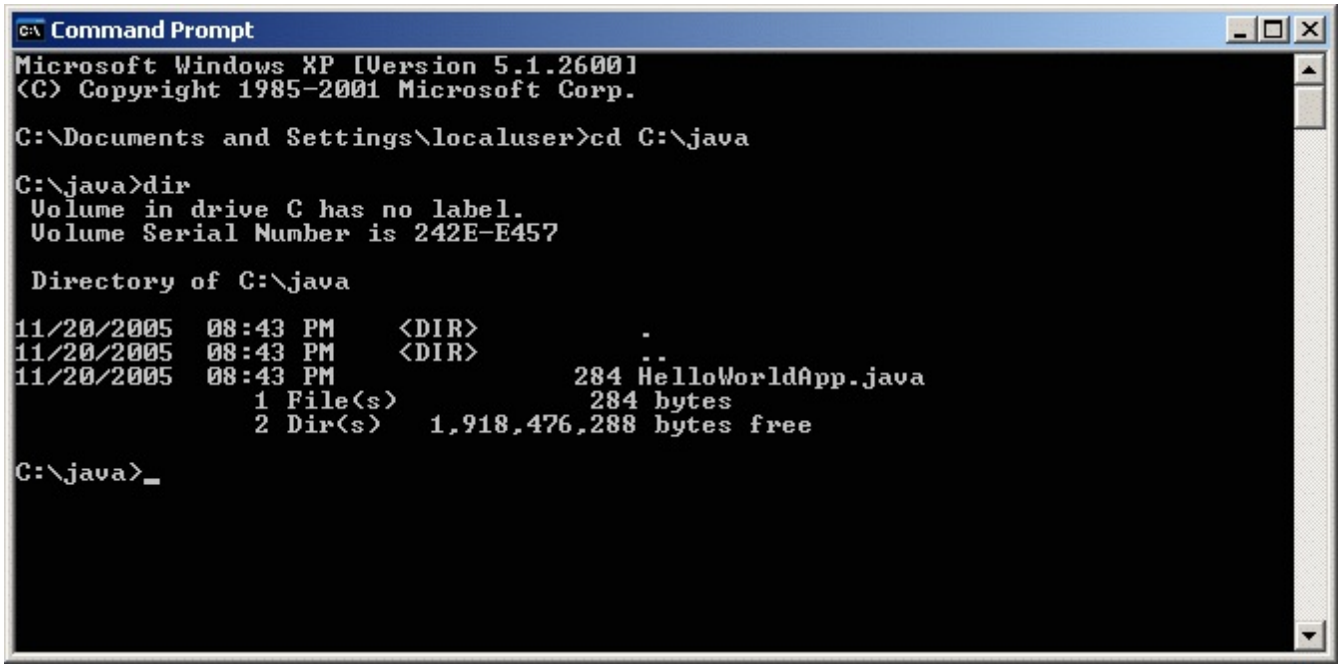
Now the prompt should change to `C:\java>`.

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.



Changing directory on an alternate drive.

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

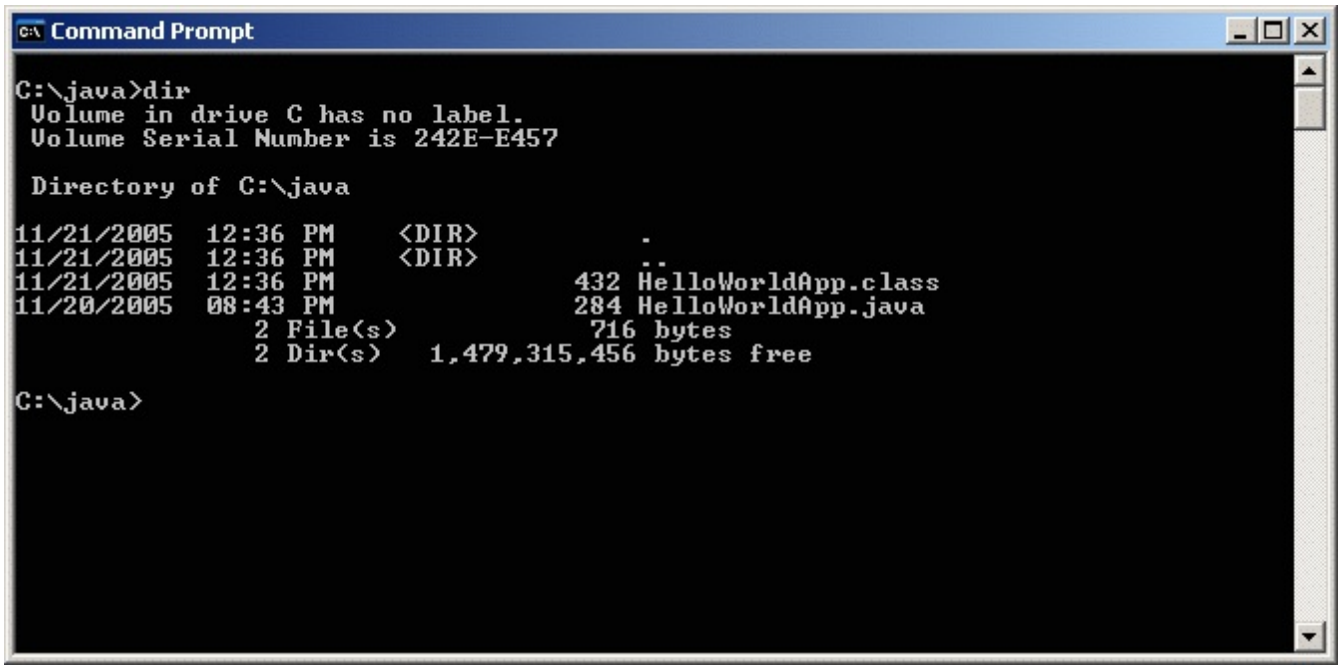


Directory listing showing the `.java` source file.

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.



Directory listing, showing the generated `.class` file

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

If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions).

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:

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

Congratulations! Your program works!

If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions).

"Hello World!" for Solaris OS and Linux

It's time to write your first application! These detailed instructions are for users of Solaris OS and Linux. Instructions for other platforms are in "Hello World!" for Microsoft Windows and "Hello World!" for the NetBeans IDE.

If you encounter problems with the instructions on this page, consult the Common Problems (and Their Solutions).

A Checklist 

To write your first program, you'll need:

1. The Java SE Development Kit 7 (JDK 7)
You can download the Solaris OS or Linux version now. (Make sure you download the **JDK**, *not* the JRE.) Consult the installation instructions.
2. A text editor
In this example, we'll use Pico, an editor available for many UNIX-based platforms. You can easily adapt these instructions if you use a different text editor, such as `vi` or `emacs`.

These two items are all you'll need to write your first application.

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 `examples/java` in the `/tmp` directory, use the following commands:

```
cd /tmp
mkdir examples
cd examples
mkdir java
```

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

```
cd /tmp/examples/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 

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

`HelloWorldApp` is *not* the same as `helloworldapp`.

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 `/tmp/examples/java`, then you type `/tmp/examples/java/HelloWorldApp.java` and press **Return**.

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

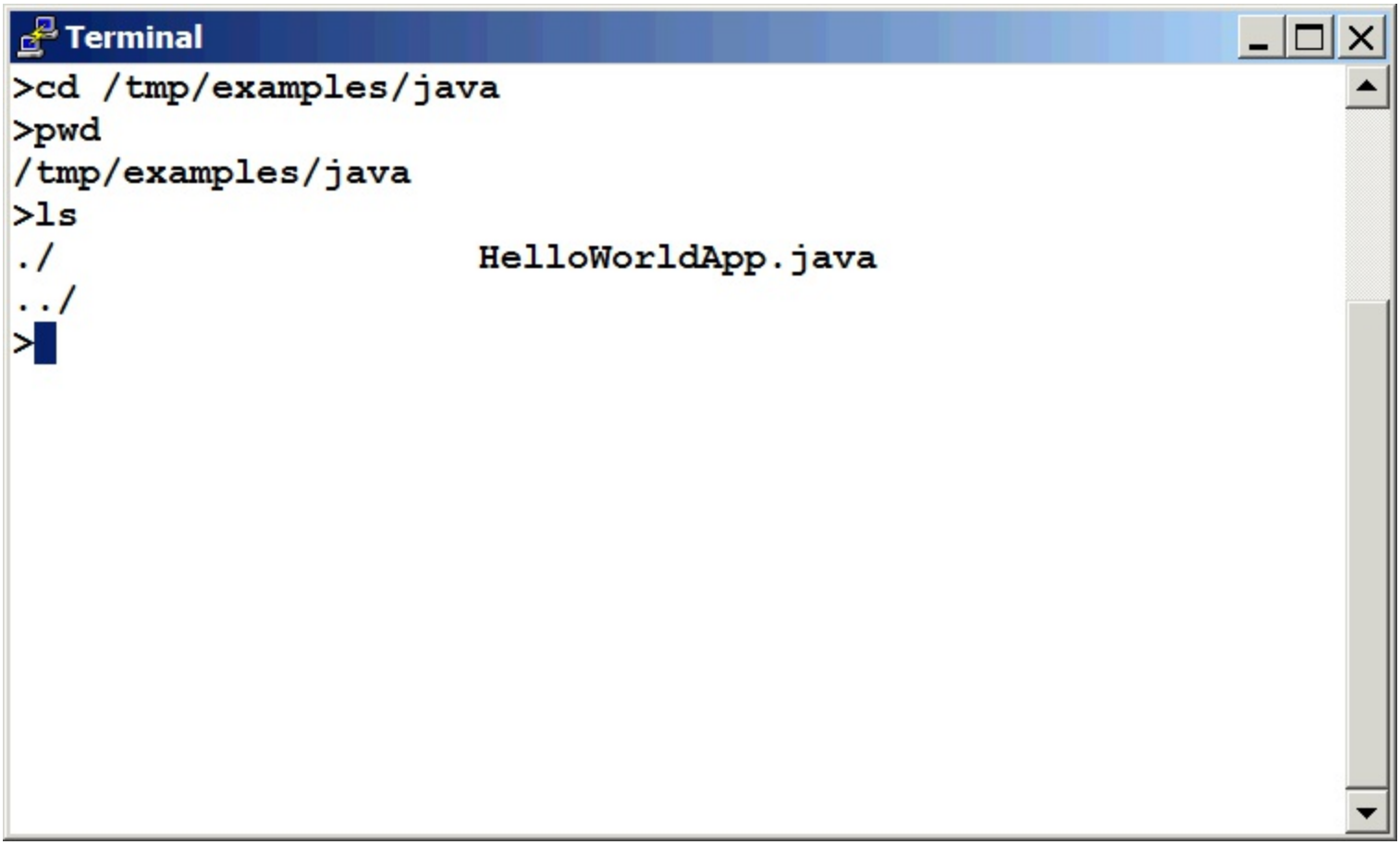
Compile the Source File into a `.class` File

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 `/tmp/examples/java`, type the following command at the prompt and press **Return**:

```
cd /tmp/examples/java
```

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

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

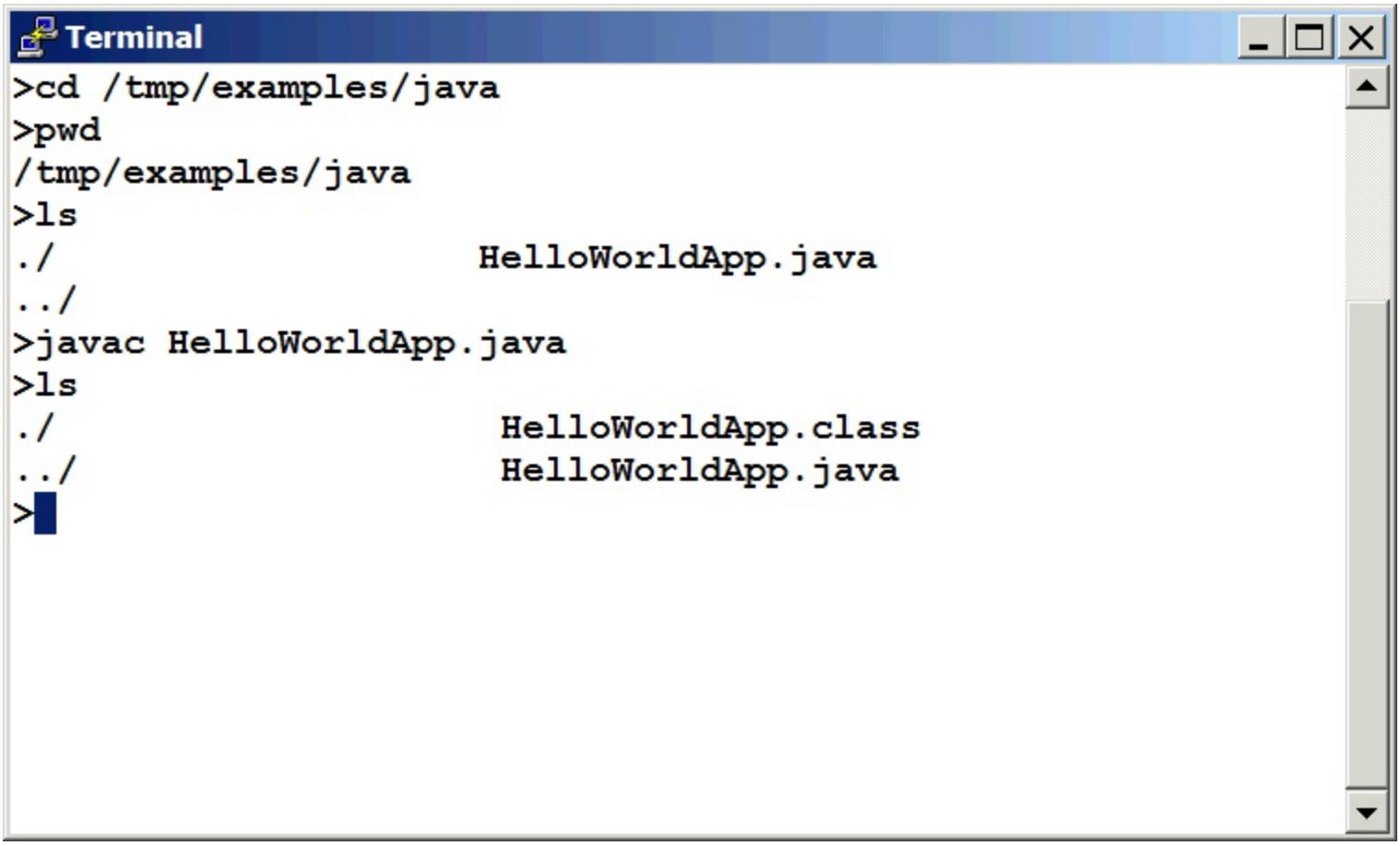


Results of the `ls` command, showing the `.java` source file.

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.



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

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

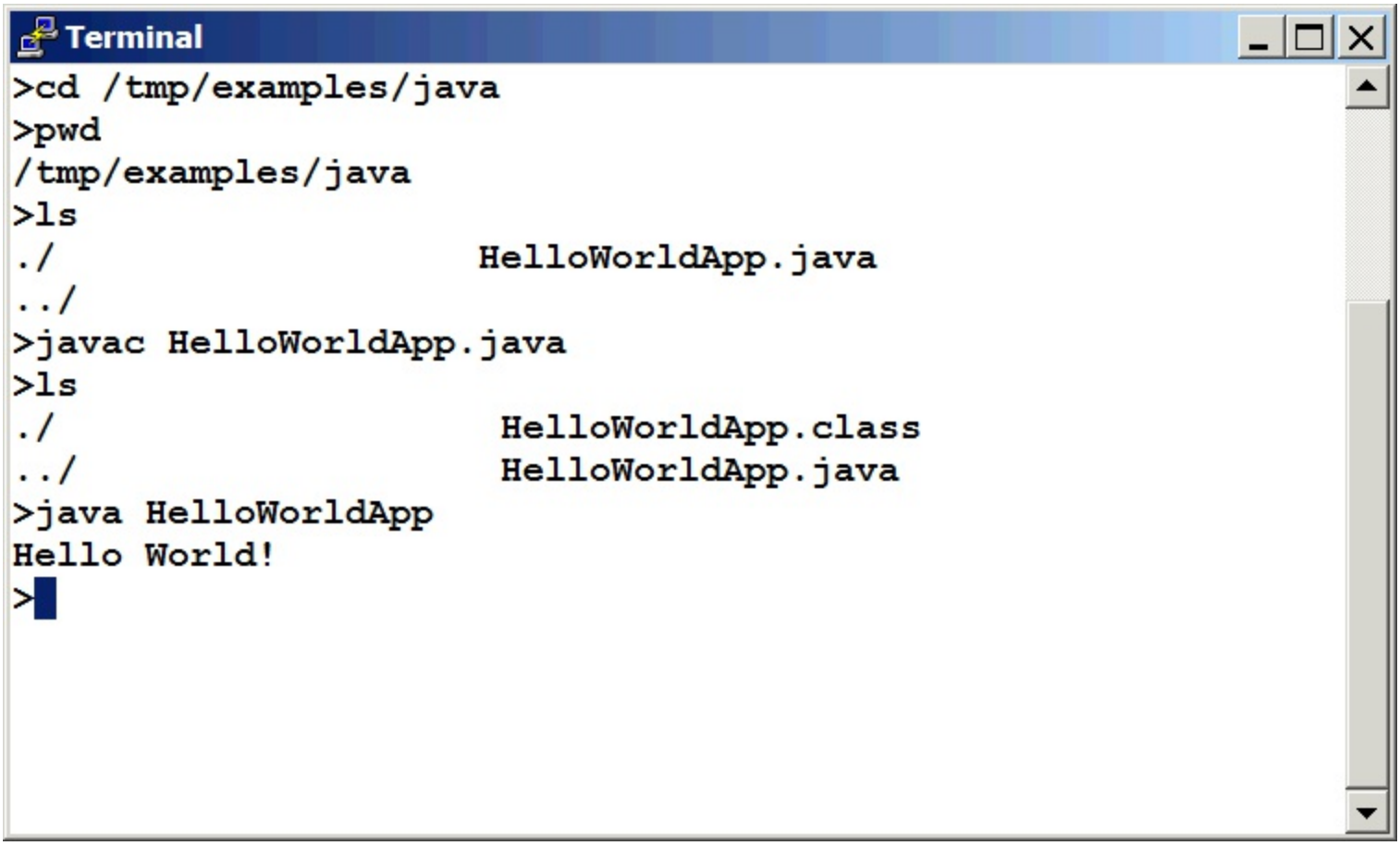
If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions).

Run the Program

In the same directory, enter at the prompt:

```
java HelloWorldApp
```

The next figure shows what you should now see.



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

Congratulations! Your program works!

If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions).

Lesson: A Closer Look at the "Hello World!" Application

Now that you've seen the "Hello World!" application (and perhaps even compiled and run it), you might be wondering how it works. Here again is its code:

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

The "Hello World!" application consists of three primary components: source code comments, the `HelloWorldApp` class definition, and the `main` method. The following explanation will provide you with a basic understanding of the code, but the deeper implications will only become apparent after you've finished reading the rest of the tutorial.

Source Code Comments

The following bold text defines the *comments* of the "Hello World!" application:

```
/**
 * 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.
    }
}
```

Comments are ignored by the compiler but are useful to other programmers. The Java programming language supports three kinds of comments:

`/* text */`
: The compiler ignores everything from `/*` to `*/`.

`/** documentation */`
: This indicates a documentation comment (*doc comment*, for short). The compiler ignores this kind of comment, just like it ignores comments that use `/*` and `*/`. The `javadoc` tool uses doc comments when preparing automatically generated documentation. For more information on `javadoc`, see the Javadoc tool documentation .

`// text`
: The compiler ignores everything from `//` to the end of the line.

The `HelloWorldApp` Class Definition

The following bold text begins the class definition block for the "Hello World!" application:

```
/**
```

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

As shown above, the most basic form of a class definition is:

```
class name {
    . . .
}
```

The keyword `class` begins the class definition for a class named `name`, and the code for each class appears between the opening and closing curly braces marked in bold above. Chapter 2 provides an overview of classes in general, and Chapter 4 discusses classes in detail. For now it is enough to know that every application begins with a class definition.

The `main` Method

The following bold text begins the definition of the `main` method:

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

In the Java programming language, every application must contain a `main` method whose signature is:

```
public static void main(String[] args)
```

The modifiers `public` and `static` can be written in either order (`public static` or `static public`), but the convention is to use `public static` as shown above. You can name the argument anything you want, but most programmers choose "args" or "argv".

The `main` method is similar to the `main` function in C and C++; it's the entry point for your application and will subsequently invoke all the other methods required by your program.

The `main` method accepts a single argument: an array of elements of type `String`.

```
public static void main(String[] args)
```

This array is the mechanism through which the runtime system passes information to your application. For example:

```
java MyApp arg1 arg2
```

Each string in the array is called a *command-line argument*. Command-line arguments let users affect the operation of the application without recompiling it. For example, a sorting program might allow the user to specify that the data be sorted in descending order with this command-line argument:

```
-descending
```

The "Hello World!" application ignores its command-line arguments, but you should be aware of the fact that such arguments do exist.

Finally, the line:

```
System.out.println("Hello World!");
```

uses the `System` class from the core library to print the "Hello World!" message to standard output. Portions of this library (also known as the "Application Programming Interface", or "API") will be discussed throughout the remainder of the tutorial.

Note: See online version of topics in this ebook to download complete source code.

Questions and Exercises: Getting Started

Questions

Question 1: When you compile a program written in the Java programming language, the compiler converts the human-readable source file into platform-independent code that a Java Virtual Machine can understand. What is this platform-independent code called?

Question 2: Which of the following is *not* a valid comment:

a. `/** comment */`
b. `/* comment */`
c. `/* comment`
d. `// comment`

Question 3: What is the first thing you should check if you see the following error at runtime:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
HelloWorldApp.java.
```

Question 4: What is the correct signature of the `main` method?

Question 5: When declaring the `main` method, which modifier must come first, `public` or `static`?

Question 6: What parameters does the `main` method define?

Exercises

Exercise 1: Change the `HelloWorldApp.java` program so that it displays `Hola Mundo!` instead of `Hello World!.`

Exercise 2: You can find a slightly modified version of `HelloWorldApp` here:
`HelloWorldApp2.java`

The program has an error. Fix the error so that the program successfully compiles and runs. What was the error?

Check your answers.

Questions

Question 1: When you compile a program written in the Java programming language, the compiler converts the human-readable source file into platform-independent code that a Java Virtual Machine can understand. What is this platform-independent code called?

Answer 1: Bytecode.

Question 2: Which of the following is *not* a valid comment:

a. `/** comment */`
b. `/* comment */`
c. `/* comment`
d. `// comment`

Answer 2: c is an invalid comment.

Question 3: What is the first thing you should check if you see the following error at runtime:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
HelloWorldApp.java.
```

Answer 3: Check your classpath. Your class cannot be found.

Question 4: What is the correct signature of the `main` method?

Answer 4: The correct signature is `public static void main(String[] args)` or `public static void main(String... args)`

Question 5: When declaring the `main` method, which modifier must come first, `public` or `static`?

Answer 5: They can be in either order, but the convention is `public static`.

Question 6: What parameters does the `main` method define?

Answer 6: The `main` method defines a single parameter, usually named `args`, whose type is an array of `String` objects.

Exercises

Exercise 1: Change the `HelloWorldApp.java` program so that it displays `Hola Mundo!` instead of `Hello World!`.

Answer 1: This is the only line of code that must change:

```
System.out.println("Hola Mundo!"); //Display the string.
```

Exercise 2: You can find a slightly modified version of `HelloWorldApp` here:
`HelloWorldApp2.java`

The program has an error. Fix the error so that the program successfully compiles and runs. What was the error?

Answer 2: Here's the error you get when you try to compile the program:

```
HelloWorldApp2.java:7: unclosed string literal
        System.out.println("Hello World!); //Display the string.
                           ^
HelloWorldApp2.java:7: ')' expected
        System.out.println("Hello World!); //Display the string.
                                                              ^
2 errors
```

To fix this mistake, you need to close the quotation marks around the string. Here is the correct line of code:

```
System.out.println("Hello World!"); //Display the string.
```

Lesson: Common Problems (and Their Solutions)

Compiler Problems

Common Error Messages on Microsoft Windows Systems

`'javac' is not recognized as an internal or external command, operable program or batch file`

If you receive this error, Windows cannot find the compiler (`javac`).

Here's one way to tell Windows where to find `javac`. Suppose you installed the JDK in `C:\jdk1.8.0`. At the prompt you would type the following command and press Enter:

```
C:\jdk1.8.0\bin\javac HelloWorldApp.java
```

If you choose this option, you'll have to precede your `javac` and `java` commands with `C:\jdk1.8.0\bin\` each time you compile or run a program. To avoid this extra typing, consult the section Updating the PATH variable in the JDK 8 installation instructions.

`Class names, 'HelloWorldApp', are only accepted if annotation processing is explicitly requested`

If you receive this error, you forgot to include the `.java` suffix when compiling the program. Remember, the command is `javac HelloWorldApp.java` not `javac HelloWorldApp`.

Common Error Messages on UNIX Systems

`javac: Command not found`

If you receive this error, UNIX cannot find the compiler, `javac`.

Here's one way to tell UNIX where to find `javac`. Suppose you installed the JDK in `/usr/local/jdk1.8.0`. At the prompt you would type the following command and press Return:

```
/usr/local/jdk1.8.0/javac HelloWorldApp.java
```

Note: If you choose this option, each time you compile or run a program, you'll have to precede your `javac` and `java` commands with `/usr/local/jdk1.8.0/`. To avoid this extra typing, you could add this information to your PATH variable. The steps for doing so will vary depending on which shell you are currently running.

`Class names, 'HelloWorldApp', are only accepted if annotation processing is explicitly requested`

If you receive this error, you forgot to include the `.java` suffix when compiling the program. Remember, the command is `javac HelloWorldApp.java` not `javac HelloWorldApp`.

Syntax Errors (All Platforms)

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 semicolon (`;`) 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, the following code snippet omits a semicolon (`;`) 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 issues two error messages because after it processes `count++`, the compiler's state indicates that it's in the middle of an expression. Without the semicolon, 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.

Runtime Problems

Error Messages on Microsoft Windows Systems

`Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorldApp`

If you receive this error, `java` cannot find your bytecode file, `HelloWorldApp.class`.

One of the places `java` tries to find your `.class` file is your current directory. So if your `.class` file is in `C:\java`, you should change your current directory to that. To change your directory, type the following command at the prompt and press Enter:

```
cd c:\java
```

The prompt should change to `C:\java>`. If you enter `dir` at the prompt, you should see your `.java` and `.class` files. Now enter `java HelloWorldApp` again.

If you still have problems, you might have to change your CLASSPATH variable. To see if this is necessary, try clobbering the classpath with the following command.

```
set CLASSPATH=
```

Now enter `java HelloWorldApp` again. If the program works now, you'll have to change your CLASSPATH variable. To set this variable, consult the Updating the PATH variable section in the JDK 8 installation instructions. The CLASSPATH variable is set in the same manner.

`Could not find or load main class HelloWorldApp.class`

A common mistake made by beginner programmers is to try and run the `java` launcher on the `.class` file that was created by the compiler. For example, you'll get this error if you try to run your program with `java HelloWorldApp.class` instead of `java HelloWorldApp`. Remember, the argument is the *name of the class* that you want to use, *not* the filename.

`Exception in thread "main" java.lang.NoSuchMethodError: main`

The Java VM requires that the class you execute with it have a `main` method at which to begin execution of your application. A Closer Look at the "Hello World!" Application discusses the `main` method in detail.

Error Messages on UNIX Systems

`Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorldApp`

If you receive this error, `java` cannot find your bytecode file, `HelloWorldApp.class`.

One of the places `java` tries to find your bytecode file is your current directory. So, for example, if your bytecode file is in `/home/jdoe/java`, you should change your current directory to that. To change your directory, type the following command at the prompt and press Return:

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

If you enter `pwd` at the prompt, you should see `/home/jdoe/java`. If you enter `ls` at the prompt, you should see your `.java` and `.class` files. Now enter `java HelloWorldApp` again.

If you still have problems, you might have to change your CLASSPATH environment variable. To see if this is necessary, try clobbering the classpath with the following command.

```
unset CLASSPATH
```

Now enter `java HelloWorldApp` again. If the program works now, you'll have to change your CLASSPATH variable in the same manner as the PATH variable above.

`Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorldApp/class`

A common mistake made by beginner programmers is to try and run the `java` launcher on the `.class` file that was created by the compiler. For example, you'll get this error if you try to run your program with `java HelloWorldApp.class` instead of `java HelloWorldApp`. Remember, the argument is the *name of the class* that you want to use, *not* the filename.

`Exception in thread "main" java.lang.NoSuchMethodError: main`

The Java VM requires that the class you execute with it have a `main` method at which to begin execution of your application. A Closer Look at the "Hello World!" Application discusses the `main` method in detail.

Applet or Java Web Start Application Is Blocked

If you are running an application through a browser and get security warnings that say the application is blocked, check the following items:

- Verify that the attributes in the JAR file manifest are set correctly for the environment in which the application is running. In a NetBeans project, you can open the manifest file from the Files tab of the NetBeans IDE by expanding the project folder and double-clicking `manifest.mf`.
- If you are running a local applet, set up a web server to use for testing. The Security Level slider in the Java Control Panel must be set to Medium to enable local applets to run, however, this is not recommended for security reasons.

Note: See online version of topics in this ebook to download complete source code.

Getting Started: End of Trail

You have reached the end of the "Getting Started" trail.

If you have comments or suggestions about this trail, use our feedback page to tell us about it.

 Learning the Java Language: This trail is a gentle introduction to object-oriented concepts and how the Java language implements them.

 Essential Classes: By taking this trail, you can find out about strings, exceptions, threads, and other Java features that are used in all kinds of Java programs.