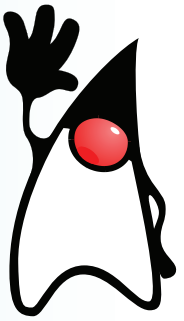
1 Getting started

Welcome to the exciting world of Java programming. This chapter shows how to create and execute simple Java programs and demonstrates how to store data within programs.

Introduction

The Java™ programming language was first developed in 1990 by an engineer at Sun Microsystems named James Gosling. He was unhappy using the C++ programming language so he created a new language that he named "Oak", after the oak tree that he could see from his office window.



As the popularity of the World Wide Web grew, Sun recognized that Gosling's language could be developed for the internet. Consequently Sun renamed the language "Java" (simply because that name sounded cool) and made it freely available in 1995. Developers around the world quickly adopted this exciting new language and, because of its modular design, were able to create new features that could be added to the core language. The most endearing additional features were retained in subsequent releases of Java as it developed into the comprehensive version of today.

The essence of Java is a library of files called "classes", which each contain small pieces of ready-made proven code. Any of these classes can be incorporated into a new program, like bricks in a wall, so that only a relatively small amount of new code ever needs to be written to complete the program. This saves the programmer a vast amount of time and largely explains the huge popularity of Java programming. Additionally, this modular arrangement makes it easier to identify any errors than in a single large program.

Java technology is both a programming language and a platform. In Java programming the source code is first written as human-readable plain text files ending with the **.java** extension. These are compiled into machine-readable **.class** files by the **javac** compiler. The **java** interpreter can then execute the program with an instance of the Java Virtual Machine (Java VM):

MyProgram.java MyProgram.class MyProgram

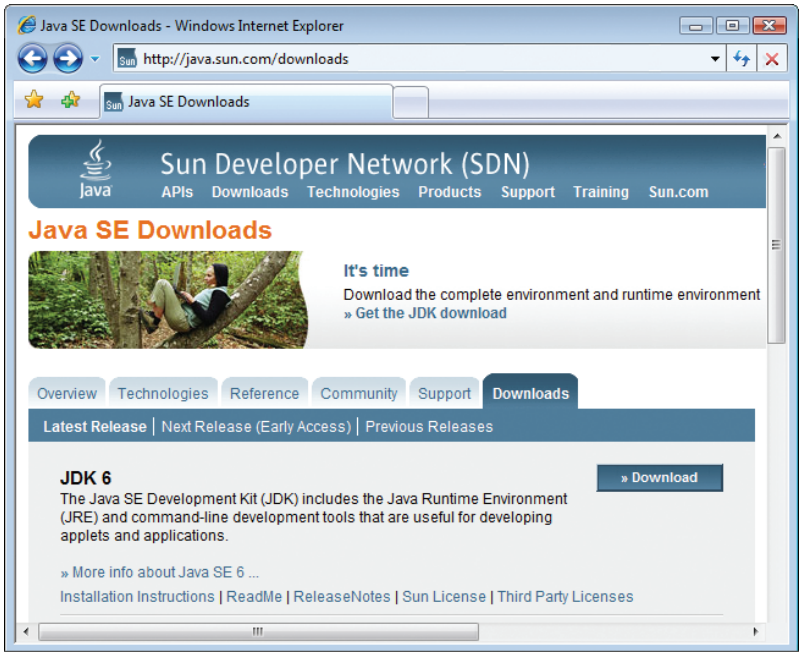
As the Java VM is available on many different operating systems the same **.class** files are capable of running on Windows, Linux and Mac operating systems – so Java programmers theoretically enjoy the cross-platform ability to "write once, run anywhere".

Hot tip

There is no truth in the rumor that JAVA stands for Just Another Vague Acronym.

In order to create Java programs **the Java class libraries and the javac** compiler need to be installed on your computer. In order to run Java programs the Java™ Runtime Environment (JRE) needs to be installed to supply the **java** interpreter. All of these components are contained in a freely available package called the Java™ Development Kit (JDK).

The Java programs in this book use version JDK 6, which incorporates the Java Standard Edition Development Kit and the Java Runtime Environment. JDK 6 can be downloaded from the Sun Microsystems website at **http://java.sun.com**.



The JDK 6 package is available in separate versions for both 32-bit and 64-bit variants of the Windows, Solaris and Linux operating systems – select the appropriate version for your computer to download the Java Development Kit.

Don't forget

The Sun download page also features other packages, but only the JDK 6 package is required to get started with Java programming.

Installing the JDK

Select the appropriate Java Development Kit package for your system from the Sun downloads page and then follow these steps to install Java on your computer:

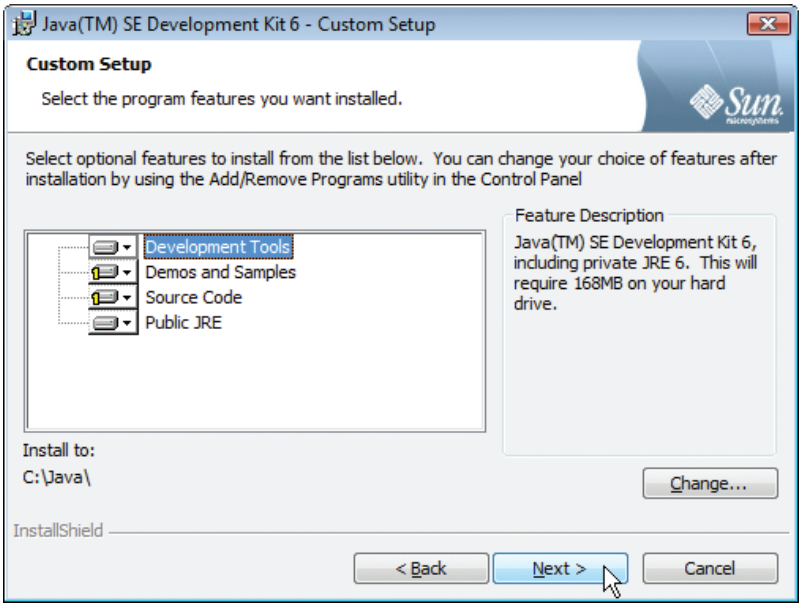
Beware

A previous version of the JRE may be installed so your web browser can run Java applets. It is best to uninstall this to avoid confusion with the newer version in JDK6.

1. Uninstall any previous versions of the JDK or Java Runtime Environment from your system

2. Start the installation and accept the License Agreement

3. When the Custom Setup dialog appears either accept the suggested installation location or click the Change button to choose your preferred location, such as **C:\Java** for Windows systems or **/usr/Java** for Linux systems



Hot tip

You can start out by installing just the minimum features to avoid confusion.

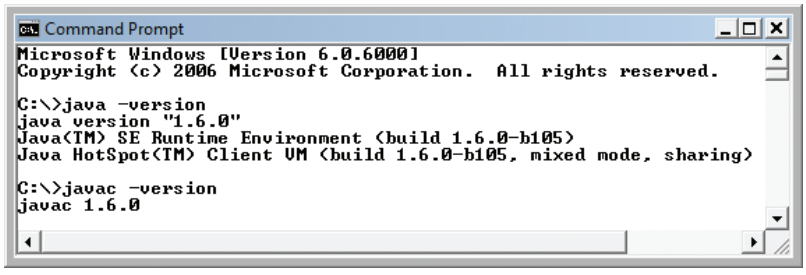
4. Ensure that the Development Tools and Public JRE features are selected from the list. Optionally you may deselect the other features as they are not required to start programming with this book

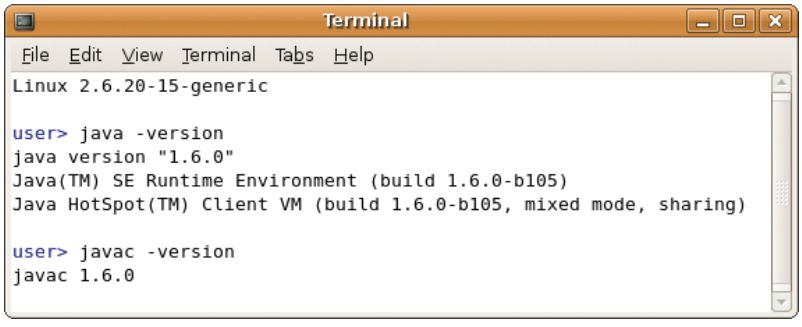
5. Click the Next button to install all the necessary Java class libraries and tools at the chosen location

The tools to compile and run Java programs are normally operated from a Command Prompt and are located in the **bin** sub-directory of the Java directory. These can be made available from anywhere on your computer by adding their location to the system path:

- On Windows Vista or Windows XP, navigate through Start, Control Panel, System, Advanced (System Settings), Environment Variables. Select the System Variable named "Path" then click the Edit button. Add the address of Java's **bin** sub-directory at the end of the list in the Variable Value field. For instance, add **C:\Java\bin;** then click the OK button.
- On Linux, add the location of Java's **bin** sub-directory to the system path by editing the **.bashrc** file in your home directory. For instance, add **PATH=$PATH:/usr/Java/bin** then save the file.

To test the environment open a prompt window then enter **java -version** and hit Return to see the interpreter's version number. Now enter **javac -version** and hit Return to see the compiler's version number. Both numbers should match – in this case each is version number 1.6.0, and you're ready to start Java programming.





Don't forget

If the .bashrc file is not immediately visible in your Linux home directory choose View, Show Hidden Files to reveal it.

Hot tip

On older versions of Windows the JDK tools can be made globally available by editing the autoexec.bat file to add the location of Java's bin sub-directory at the end of the SET PATH line.

Writing a first Java program

All Java programs start as text files that are later used to create "class" files, which are the actual runnable programs. This means that Java programs can be written in any plain text editor such as the Windows Notepad application.

Follow these steps to create a simple Java program that will output the traditional first program greeting:

Hello.java

1. Open a plain text editor, like Notepad, and type this code exactly as it is listed – to create a class named "Hello"

```
class Hello
{

}
```

2. Between the curly brackets of the **Hello** class, insert this code – to create a "main" method for the **Hello** class

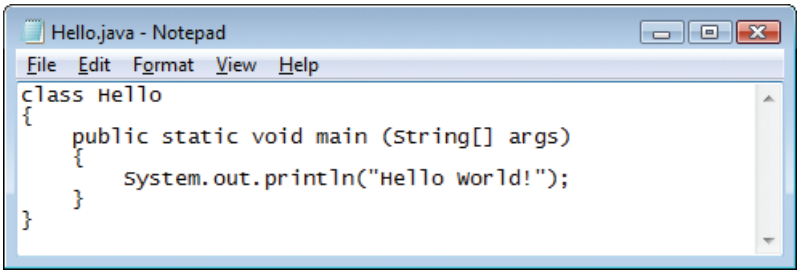
```
public static void main( String[] args )
{

}
```

3. Between the curly brackets of the **main** method, insert this line of code – stating what the program will do

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

4. Save the file at any convenient location, but be sure to name it precisely as **Hello.java** – the complete program should now look like this:



Beware

Java is a case-sensitive language where "Hello" and "hello" are distinctly different – traditionally Java program names should always begin with an uppercase letter.

Don't forget

Java programs are always saved as their exact program name followed by the ".java" extension.

The separate parts of the program code on the opposite page can be examined individually to understand each part more clearly:

The Program Container

```
class Hello {      }
```

The program name is declared following the **class** keyword and followed by a pair of curly brackets. All of the program code that defines the **Hello** class will be contained within these curly brackets.

The Main Method

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

This fearsome-looking line is the standard code that is used to define the starting point of nearly all Java programs. It will be used in most examples throughout this book exactly as it appears above – so it may be useful to memorize it.

The code declares a method named "main" that will contain the actual program instructions within its curly brackets.

Keywords **public static void** precede the method name to define how the method may be used and are explained in detail later.

The code (**String[] args)** is useful when passing values to the method and is also fully explained later in this book.

The Statement

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

Statements are actual instructions to perform program tasks and must always end with a semi-colon. A method may contain many statements inside its curly brackets to form a "statement block" defining a series of tasks to perform, but here a single statement instructs the program to output a line of text.

Turn to the next page to discover how to compile and run this program.

All stand-alone Java programs must have a main method. Java applets are different and their format is explained later.

Create a MyJava directory in which to save all your Java program files.

Compiling & running programs

Before a Java program can run it must first be compiled into a **class** file by the Java compiler. This is located in Java's **bin** sub-directory and is an application named **javac**. The instructions on page 11 described how to add the **bin** sub-directory to the system path so that **javac** can be invoked from any system location.

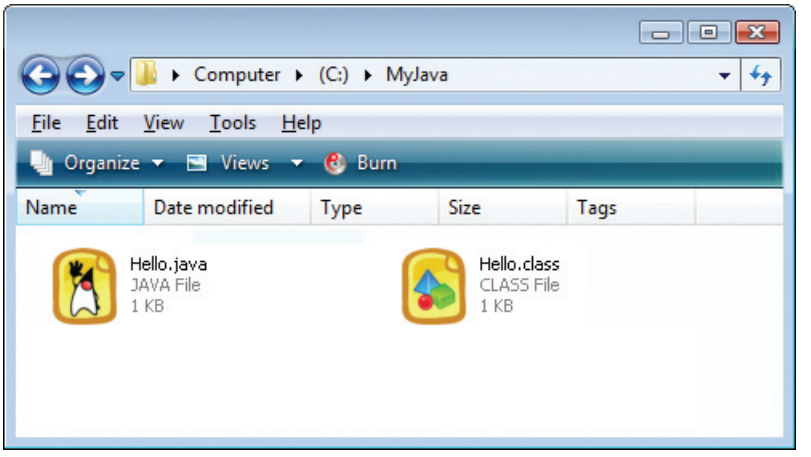
Follow these steps to compile the program on the previous page:

1. Open a Command Prompt/Terminal window then navigate to the directory where you saved the **Hello.java** source code file

2. At the prompt type **javac** followed by a space then the full name of the source code file **Hello.java** and hit Return

If the **javac** compiler discovers errors in the code it will halt and display a helpful report indicating the nature of the error – see page 22 for Troubleshooting Problems.

If the **javac** compiler does not find any errors it will create a new file with the program name and the **.class** file extension.



Hot tip

Open a Command Prompt/Terminal window then type javac and hit Return to reveal the Java compiler options.

Hot tip

You can also compile the source code from another location if you state the file's full path address to the javac compiler – in this case C:\MyJava\Hello.java.

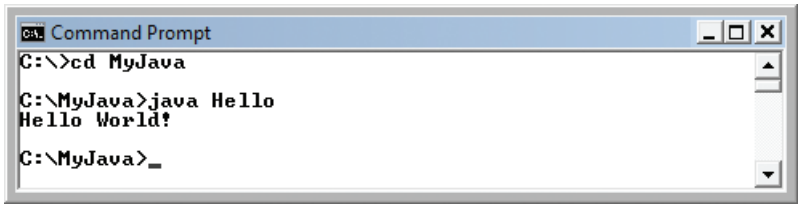
When the Java compiler completes compilation the Command Prompt/Terminal window focus returns to the prompt without any confirmation message – and the program is ready to run.

The Java program interpreter is an application named **java** that is located in Java's **bin** sub-directory – alongside the **javac** compiler. As this directory was previously added to the system path, on page 11, the **java** interpreter can be invoked from any location.

Follow these steps to run the program that was compiled using the procedure described on the page opposite:

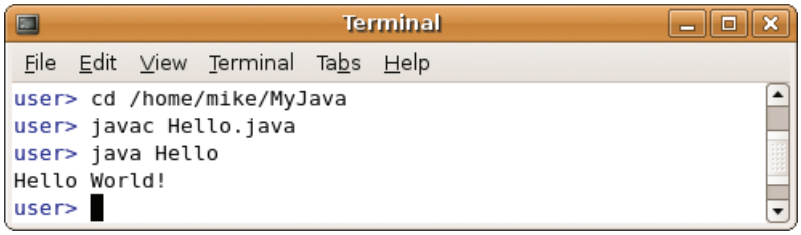
1. Open a Command Prompt/Terminal window then navigate to the directory where the **Hello.class** program file is located

2. At the prompt type **java** followed by a space then the program name **Hello** and hit Return



The **Hello** program runs and executes the task defined in the statement within its main method – to output "Hello World!". Upon completion focus returns to the prompt once more.

The process of compiling and running a Java program is typically combined in sequential steps, and is the same regardless of platform. The screenshot below illustrates the **Hello** program being compiled and run in combined steps on a Linux system:



Do not include the .class extension when running a program – only use the program name.

Creating a variable

In Java programming a "variable" is simply a useful container in which a value may be stored for subsequent use by the program. The stored value may be changed (vary) as the program executes its instructions – hence the term "variable".

A variable is created by writing a variable "declaration" in the program, specifying the type of data that variable may contain and a given name for that variable. For example, the **String** data type can be specified to allow a variable named "message" to contain regular text with this declaration:

```
String message ;
```

Variable names are chosen by the programmer but must adhere to certain naming conventions. The variable name may only begin with a letter, dollar sign $, or the underscore character _ , and may subsequently have only letters, digits, dollar signs, or underscore characters. Names are case-sensitive, so "var" and "Var" are distinctly different names, and spaces are not allowed in names.

Variable names should also avoid the Java keywords, listed in the table below, as these have special meaning in the Java language.

Beware

Each variable declaration must be terminated with a semi-colon character – like all other statements.

Hot tip

Strictly speaking, some words in this table are not actually keywords – true, false, and null are all literals, String is a special class name, const and goto are reserved words (currently unused). These are included in the table because they must also be avoided when naming variables.

abstract	default	goto	package	synchronized
assert	do	if	private	this
boolean	double	implements	protected	throw
break	else	import	public	throws
byte	enum	instanceof	return	transient
case	extends	int	short	true
catch	false	interface	static	try
char	final	long	strictfp	void
class	finally	native	String	volatile
const	float	new	super	while
continue	for	null	switch	

As good practice, variables should be named with words or easily recognizable abbreviations, describing that variable's purpose. For example, "button1" or "btn1" to describe button number one. Lowercase letters are preferred for single-word names, such as "gear", and names that consist of multiple words should capitalize the first letter of each subsequent word, such as "gearRatio" – the so-called "camelCase" naming convention.

Once a variable has been declared, it may be assigned an initial value of the appropriate data type using the equals sign = , either in the declaration or later on in the program, then its value can be referenced at any time using the variable's name.

Follow these steps to create a program that declares a variable, which gets initialized in its declaration then changed later:

1. Start a new program named "FirstVariable", containing the standard main method

```
class FirstVariable
{
        public static void main ( String[] args ) {        }
}
```

FirstVariable.java

2. Between the curly brackets of the main method, insert this code to create, intialize, and output a variable

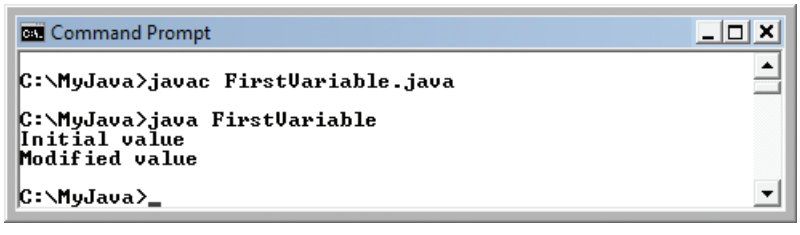
```
String message = "Initial value" ;
System.out.println( message ) ;
```

3. Add these lines to modify and output the variable value

```
message = "Modified value" ;
System.out.println( message ) ;
```

If you encounter problems compiling or running the program you can get help from Troubleshooting Problems on page 22.

4. Save the program as **FirstVariable.java** then compile and run the program



Recognizing data types

The most frequently used data types in Java variable declarations are listed in this table along with a brief description:

Due to the irregularities of floating-point arithmetic the float data type should never be used for precise values, such as currency – see page 130 for details.

Data type	Description	Example
char	A single Unicode character	'a'
String	Any number of Unicode characters	"my String"
int	An integer number, from -2.14 billion to +2.14 billion	1000
float	A floating-point number, with a decimal point	3.14159265f
boolean	A logical value of either true or false	true

Notice that **char** data values must always be surrounded by single quotes and **String** data values must always be surrounded by double quotes. Also remember that **float** data values must always have an "f" suffix to ensure they are treated as a **float** value.

In addition to the more common data types above, Java provides these specialized data types for use in exacting circumstances:

All data type keywords begin with a lowercase letter except String – which is a special class.

Data type	Description
byte	Integer number from -128 to +127
short	Integer number from -32,768 to +32,767
long	Positive or negative integer exceeding 2.14 billion
double	Extremely long floating-point number

Specialized data types are useful in advanced Java programs – the examples in this book mostly use the common data types described in the top table.

Follow these steps to create a Java program that creates, initializes, and outputs variables of all five common data types:

1. Start a new program named "DataTypes" containing the standard main method

```
class DataTypes
{
        public static void main ( String[] args ) {          }
}
```

DataTypes.java

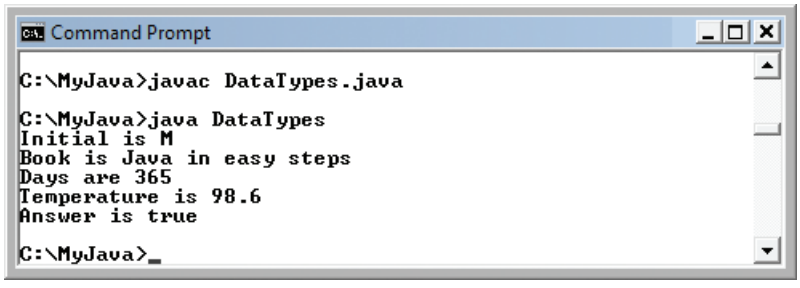
2. Between the curly brackets of the main method, insert these declarations to create and intialize five variables

```
char letter = 'M' ;
String title = "Java in easy steps" ;
int number = 365 ;
float decimal = 98.6f ;
boolean result = true ;
```

3. Add these lines to output an appropriate text **String** concatenated to the value of each variable

```
System.out.println( "Intial is " + letter ) ;
System.out.println( "Book is " + title ) ;
System.out.println( "Days are " + number ) ;
System.out.println( "Temperature is " + decimal ) ;
System.out.println( "Answer is " + result ) ;
```

4. Save the program as **DataTypes.java** then compile and run the program



Hot tip

Notice how the + character is used here to join (concatenate) text String values and variable values.

Hot tip

The Java compiler will report an error if the program attempts to assign a value of the wrong data type to a variable – try changing the values in this example, then attempt to recompile the program to see the effect.

Creating constants

The "final" keyword is a modifier that can be used when declaring variables to prevent any subsequent changes to the values that are initially assigned to them. This is useful when storing a fixed value in a program to avoid it becoming altered accidentally.

Variables created to store fixed values in this way are known as "constants" and it is convention to name constants with all uppercase characters – to distinguish them from regular variables. Programs that attempt to change a constant value will not compile and the **javac** compiler will generate an error message.

Follow these steps to create a Java program featuring constants:

Constants.java

1. Start a new program named "Constants" containing the standard main method

```
class Constants
{
        public static void main ( String[] args ) {          }
}
```

2. Between the curly brackets of the main method, insert this code to create and intialize three integer constants

```
final int TOUCHDOWN = 6 ;
final int CONVERSION = 1 ;
final int FIELDGOAL = 3 ;
```

3. Now declare four regular integer variables

```
int td, pat, fg, total;
```

4. Intialize the regular variables – using multiples of the constant values

```
td = 4 * TOUCHDOWN ;
pat = 3 * CONVERSION ;
fg = 2 * FIELDGOAL ;
total = ( td + pat + fg ) ;
```

5. Add this line to display the total score

```
System.out.println( "Score: " + total ) ;
```

6. Save the program as **Constants.java** then compile and run the program to see the output, Score: 33
(4 x 6 = 24, 3 x 1 = 3, 2 x 3 = 6, so 24 + 3 + 6 = 33).

Hot tip

The * asterisk character is used here to multiply the constant values, and parentheses surround their addition for clarity.

Adding comments

When programming, in any language, it is good practice to add comments to program code to explain each particular section. This makes the code more easily understood by others, and by yourself when revisiting a piece of code after a period of absence.

In Java programming, comments can be added across multiple lines between /* and */ comment identifiers, or on a single line after a // comment identifier. Anything appearing between /* and */, or on a line after //, is completely ignored by the **javac** compiler.

When comments have been added to the **Constants.java** program, described opposite, the source code might look like this:

```
/*
        A program to demonstrate constant variables.
*/

class Constants
{
        public static void main( String args[] )
        {
                // Constant score values.
                final int TOUCHDOWN = 6 ;
                final int CONVERSION = 1 ;
                final int FIELDGOAL = 3 ;

                // Calculate points scored.
                int td, pat, fg, total ;
                td  = 4 * TOUCHDOWN ;          // 4x6=24
                pat = 3 * CONVERSION ;         // 3x1= 3
                fg  = 2 * FIELDGOAL ;          // 2x3= 6
                total = ( td + pat + fg ) ;    // 24+3+6=33

                // Output calculated total.
                System.out.println( "Score: " + total ) ;
        }
}
```

Saved with comments, the program compiles and runs as normal:

```
Command Prompt

C:\MyJava>javac Constants.java

C:\MyJava>java Constants
Score: 33

C:\MyJava>_
```

Constants.java (commented)

You can add a statement that attempts to change the value of a constant, then try to recompile the program to see the resulting error message.

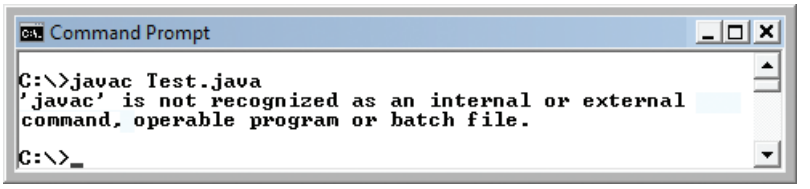
Troubleshooting problems

Sometimes the **javac** compiler or **java** interpreter will complain about errors so it's useful to understand their cause and how to quickly resolve the problem. In order to demonstrate some common error reports this code contains some deliberate errors:

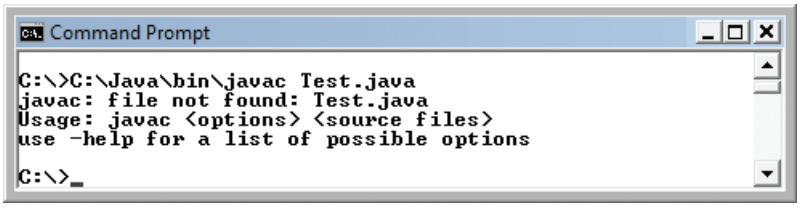
Test.java

```
class test
{
        public static void main ( String[] args )
        {
                String text ;
                System.out.println( "Test " + text )
        }
}
```

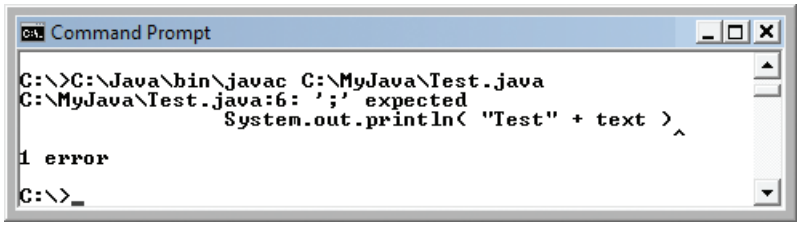
A first attempt to compile **Test.java** throws up this error report:



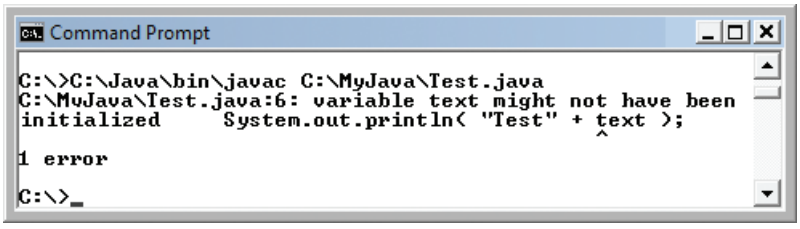
- Cause – the **javac** compiler cannot be found
- Solution – edit the system **PATH** variable, as described on page 11, or use its full path address to invoke the compiler



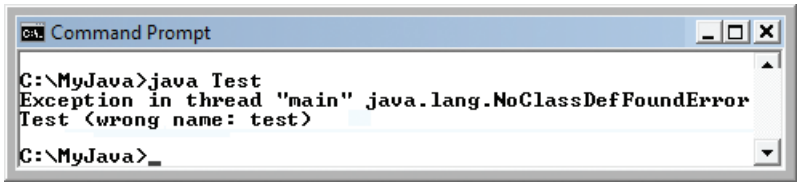
- Cause – the file **Test.java** cannot be found
- Solution – navigate to the directory where the file is located, or use the full path address to the file in the command



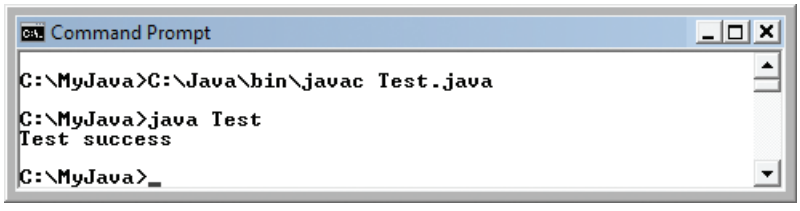
- Cause – the statement is not terminated correctly
- Solution – add a semi-colon at the end of the statement



- Cause – the variable **text** has no value
- Solution – in the variable declaration assign the variable a valid **String** value, for instance **= "success"**



- Cause – the program name and class name do not match
- Solution – change the class name from **test** to **Test** then save the source code file, recompile, and run the program:



The error report where the program name does not match the class name includes many additional lines that are not shown here, for brevity.

You must run the program from within its directory – you cannot use a path address as the java launcher requires a program name, not a file name.

Summary

- Java is both a programming language and a runtime platform
- Java programs are written as plain text files with a **.java** extension
- The Java compiler **javac** creates compiled **.class** program files from original **.java** source code files
- The Java interpreter **java** executes compiled programs using an instance of the Java Virtual Machine
- The Java VM is available on many operating system platforms
- Adding Java's **bin** sub-directory to the system **PATH** variable allows the **javac** compiler to be invoked from anywhere
- Java is a case-sensitive language
- The standard **main** method is the entry point for Java programs
- The **System.out.println()** statement outputs text
- A Java program file name must exactly match its class name
- Java variables can only be named in accordance with specified naming conventions and must avoid the Java keywords
- In Java programming each statement must be terminated by a semi-colon character
- The most common Java data types are **String**, **int**, **char**, **float** and **boolean**
- **String** values must be enclosed in double quotes, **char** values in single quotes, and **float** values must have an "f" suffix
- The **final** keyword can be used to create a constant variable
- Comments can be added to Java source code between **/*** and ***/**, on one or more lines, or after **//** on a single line
- Error reports identify compiler and runtime problems

Contents

Foreword

The examples in this book have been carefully prepared to demonstrate many features of Java. You are encouraged to try out the examples on your own computer to discover the exciting possibilities offered by the Java programming language. The straightforward descriptions should allow you to easily recreate the examples manually or, if you prefer, you can download an archive containing all the compiled example programs and their source code by following these simple steps:

1. Open your browser and visit our website at **http://www.ineasysteps.com**

2. Navigate to the **Resource Center** and choose the **Downloads** section

3. Find the **From Java in easy steps, 3rd edition** item in the **Source Code** list then click on the hyperlink entitled **All code examples** to download the compressed ZIP archive

4. Extract the contents of the ZIP archive to any convenient location on your computer – for easy reference these are arranged in sub-folders whose names match each chapter title of this book. The programs are named as described in the book and are located in the appropriate chapter folder of the archive. For example, the **Hello** program, described in the first chapter, is located in the chapter folder named **1-Getting started**.