**Chapter II**

**Introduction to Java**

**Chapter II Topics**

2.1 Introduction

2.2 Getting Started with Java

2.3 A Visit to the United Nations

2.4 Java Bytecode

2.5 Applications and Applets

2.6 Using Java Software

2.7 Text Output With print & println

2.8 Program Compile Errors

2.9 Program Comments

2.10 Escape Sequences

2.11 The AP Java Subset

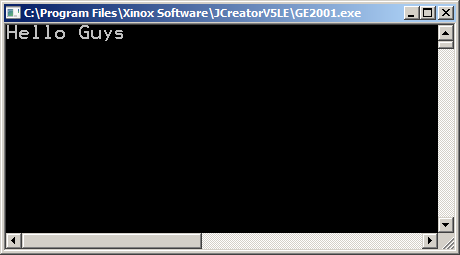
2.12 The Responsible Use of Computers

2.13 Summary

**2.1 Introduction**

Some students are impatient in general, but when it comes to technology, many students are very impatient. Lots of students march in a computer science class with prior knowledge of using a computer. Such students have played fancy video games, emailed all over town, researched on the Internet, and used computers for word processing and many other applications. These students are not impressed with a dull - black & white - text output, shown in figure 2.1 that displays: Hello Guys.

**Figure 2.1**



Computer science is not very different from many other professions in the introductory stages. Few medical students perform open-heart surgery during the first week of medical school. The average law student is not asked to argue a case before the Supreme Court after studying one court case. Navy pilot candidates are not asked to land an airplane on an aircraft carrier at night, during a storm, with one engine shut down, during week-one of flight training. In other words, if you expect to start creating programs that will guarantee you ten-thousand dollars a week part-time job, then you will be disappointed.

This chapter will start by showing you the necessary tools to write a simple Java program. Keep in mind that you will be creating your program in a high-level language, Java. This means you need some type of editing environment to write your program. After the program is written you need to go through several steps to create machine code that can be executed by the computer. Learning those fundamental program-writing skills is right now your primary concern.

There are two primary output modes that are available with Java: *text* and *graphics*. Text is both pretty easy and pretty boring. Graphics is exciting, interesting and quite a bit more challenging. The focus in this chapter is to understand first how to execute small, simple text programs. Exciting video games, snazzy interactive web pages are all possible with Java. However, that is not a practical starting point.

It is possible that you already know some other programming language, such as C++ or Visual BASIC. If such is the case, you will learn Java much faster and you may find the topic descriptions in Exposure Java rather slow moving. That is fine, but keep in mind that this text book is written for the student who has no prior knowledge of any programming language or any exposure to computer science concepts. If you have such prior knowledge, you are lucky, but do be careful. Java is similar to some other program languages, but there are also major differences, which can cause difficulties.

**2.2 Getting Started with Java**

There are many abstract concepts in computer science. These are concepts independent of any programming language. *Object Oriented Design*, *Algorithmic Analysis* and various other topics can be discussed without reference to a specific programming language. In high school and in college the title of the course will probably say *Computer Science*or *AP Computer Science*. The course title will not state *Java Programming*. Yet here you are in Chapter II with a section titled: *Introduction to Java*. You will learn many common generic computer science concepts. My experience has shown that students are more comfortable when abstractness and generic concepts come at a later date.

Consider the following. You love to travel. Traveling everywhere is your goal, your passion and your dreams. So what do you do? Just travel? That is not so practical. You want to visit many foreign destinations, but getting there requires familiarity with air travel or train travel or car travel or canoeing or hiking or biking. At this stage let us consider that our goal and destination is to unlock the mysteries of computer science and right now we are using the *Java Train* to get us there. So come aboard and see where we are heading.

**2.3 A Visit to the United Nations**

Some students are getting concerned. Perhaps you have heard that Mr. Schram is actually also certified as a Social Studies teacher and the poor man is just getting confused. How does the United Nations help us get started with Java? Good question and you will soon see how the United Nations provides us with an excellent analogy to explain how Java accomplishes some of its main goals. Have you ever observed a United Nations meeting with all the delegates present? It is fascinating to hear around 200 delegates from around the world communicate and yet each delegate speaks in his or her own language. How does everybody else understand what is discussed in roughly 200 different languages?

The delegates could consider that everybody speaks in one universal language. What might that language be? Oh, of course, it is English. That is easy for you to say, because you already speak English. How about Spanish? Spanish is spoken all around the world. It seems like a good candidate for a universal language. Now hold on and be not so quick. In reality more people speak Chinese than English or Spanish, so the logical choice should be Chinese. I trust you see the problem. Convincing 200 delegates to adopt a single language would probably mark the end of the United Nations, as we know it today.

Now the UN could hire translators for every possible language combination. That adds up to about 20,000 different translators and an enormous salary budget for the UN. We have not even considered where the 20,000 translators would be staying in the building. Perhaps you think that 20,000 is an outrageous number, but think about it. The delegate speaking *Punjabi* needs 199 translators from *Punjabi* to each of the other 199 languages. The next delegate needs 198 translators, and then 197 and so forth until the last delegate needs one translator. This means we need to add up all the numbers from 1 to 199. There is a mathematical formula that will compute this:

Sum of all #s from 1 to N = **N(N+1) = 199(200) = 19,900 (**close to 20,000)

**2 2**

I hope you do not think that the UN uses such an approach. A clever two-step translation process is used. English is selected as an intermediate language. Do not get confused, because English is not the selected universal language for all the delegates. It is an *intermediate* language. This requires 199 translators who can each speak one of the 199 languages and English.

Assume that delegate from India speaks *Punjabi*. His or her message is first translated by one translator from *Punjabi* to *English*. Then 198 translators go to work and translate *English* to each one of the other 198 languages. Suppose that the delegate from Iran responds in *Farsi*. The process repeats itself in the opposite direction. *Farsi* is first translated into *English*. Now *English* is translated into 198 different languages, including *Punjabi*, and the delegate from India has a first response. Yes it does result in a time delay, but with a minimal number of translators everybody is able to communicate. The secret of the United Nations communication success is the two-step translation process with the intermediate English step. Java uses a similar two-step translation process to achieve one of its important goals of being platform independent.

**2.4 Java Bytecode**

Computers have improved tremendously over the years and today most computers communicate rather well with each other. Still, it is common to create a program on one computer that will simply not work on another computer. As a summer workshop presenter for many years I would be careful to bring along my own Pascal and C++ software. Early on, I learned that my programs, carefully written and checked in my own lab, would frequently hiccup in another lab with different computers and software.

One of Java's goals is to be platform independent. The term *platform* is used abstractly here, but you can use it literally. If you walk from one platform onto another platform that are both of identical size and height it is easy to transfer. Different platforms cause difficulties. In the early days of train travel, Europeans experienced the inconvenience of changing trains at some country borders, because the railroad tracks were different sizes and trains could not continue.

|  |
| --- |
| **Platform Independence** |
| **A programming language is considered *platform independent* or *portable* if program source code created on one type of computer platform can execute on another computer platform without any difficulty.** |

So how does Java accomplish this feat? Programs start out as source code written by a programmer in a somewhat human language, which in this case is Java. Next this source code needs to be translated into machine code that the computer understands. A machine code file on one computer does not necessarily make sense to another computer. The code file may contain lovely ones and zeroes of the type that computers use, but not all computers follow the same rules of how to handle this low-level code. Look around when you travel. Not everybody drives on the right side of the road.

Java starts by compiling the high-level source code into *bytecode*. Bytecode is almost machine code, but not quite. Computers cannot execute bytecode. The bytecode file that is created will be identical regardless of where it was created. Think about the UN. Bytecode is the intermediate English stage. This means that Java compilers need to be available for all the different platforms to create the exact same bytecode for the same source code.

Bytecode is not understood by any computer. It is not the machine code that is normally required for execution. However, bytecode is understood by a special program that is a Java interpreter. That is right, Java uses a compiler to translate programmer's source code into bytecode and then continues with an interpreter to translate and execute the bytecode file line by line.

|  |
| --- |
| **Bytecode** |
| **Bytecode is a low level-level code file that cannot execute as a regular machine code file.**  **Bytecode is understood, and executed, by a Java interpreter, called a Java Virtual Machine (JVM)** |

Do you understand the United Nations analogy better now? Platform-A uses the necessary compiler software to translate a source code program into a bytecode file. This bytecode file arrives at Platform-B. Platform-B has acquired a Java interpreter that will translate, and execute the bytecode according to Platform-B's requirements.

So does the two-step translation process makes sense to you, even though *platform-independence* is not a priority in your life? But why use both a compiler and interpreter? Aren't compilers more efficient translators? Good question and compilers do execute much faster than interpreters. However, the efficiency comes with a price. Compilers are much larger programs than interpreters. It takes sophisticated processing to determine if an entire program has correct syntax before the translation process starts. Interpreters are only concerned if one line has correct syntax and then proceeds to translate that one line.

The answer to this question is connected with Sun Microsystems's (now Oracle) second goal. Remember that their first goal was to create a *platform-independent* language, and that goal was achieved with the two-step translation process. Their second goal was to create a language that would be suitable with web pages on the Internet. Achieving this goal requires that a web browser can translate the provided bytecode. Using a compiler does mean efficient execution speed, but it also means a large program that needs to be added to the web browser.

There is also a secondary problem. Compilers first translate the entire file before execution. Now this is a big plus and customers of application programs do not have to wait around to translate any programs. When you word process you do not start by translating some source code program. This was already done for you by the software company. You get the executable program and the only waiting time required is the time that it takes to load the program in memory and execute the instructions of the program. With a web browser there is another story. You click on some Internet link and now your web browser downloads some Java bytecode. Many people will get impatient waiting around for an initial translation process, even if the end result is quicker. If the web browser uses an interpreter that starts spitting out execution after each line of translation that means something is happening, and this approach is more desirable for a web browser.

|  |
| --- |
| **Java Uses a Compiler and an Interpreter** |
| **Java uses a compiler to translate the program source code created by the programmer into bytecode.**  **Java then continues and uses an interpreter to translate the bytecode into executable machine code line by line.** |

It must be clearly understood that it is not possible to take some Java bytecode and start executing this on any computer. Yes the code is meant to be portable to any computer with any operating system. However, this only works with a computer that has a *Java Virtual Machine*(JVM). **JVM** is the name for the interpreter that takes the bytecode and then executes the program. The name *virtual*is selected because with this interpreter in place, the computer now acts like any other computer that has the JVM. The key here is that there are many JVMs floating around. Everyone starts with the same bytecode, but these different JVMs all take the bytecode and create the appropriate machine code for its assigned computer.

You may also understand some delay with web pages. First, there is the time required to actually download the web page file. Second, if the file contains Java bytecode then there will be a second delay translating the Java file. Why then use Java? Java has several nice benefits for the Internet. With Java it is possible to make web pages dynamic and interact with the user. Regular web pages can look pretty, but it requires some special language like JavaScript, VBScript or Java to provide information to web pages that can be processed. A second benefit is security. Regular HTML web pages do not hide the code. The code used to create the web page can be viewed in the browser. Frequently, this is not very desirable like in the case where you take an on-line test. It is not possible to see the original Java source code and even if you could view the Java bytecode in the browser, it would not make any sense at all.

**2.5 Applications and Applets**

The term *Java Applet*is now so widespread that most computer and Internet users have seen or heard this term rather frequently. Do you know what an *applet* is? The name implies *little application* and that definition can cause confusion. A Word Processing program is a very large application. Another program that averages three numbers is a very small application, but the average program can either be an application or an applet in Java.

When a program is written to execute in its own stand-alone environment, it is a regular application. This is different from a program that operates inside a web page. The web page is controlled by HTML, which is *Hyper Text Markup Language*, a special program language for web pages. When a Java bytecode file operates inside a web page, it must conform to certain web page standards that do not exist in a stand-alone environment. The web page controls the Java bytecode and not the other way around. Now if the bytecode is designed to operate inside a web page it is called an *applet*, otherwise it is called an *application*.

|  |
| --- |
| Applet or Application? |
| **A Java program designed to operate inside a web page is called an *applet*.**  **A Java program designed to operate in a stand-alone environment is called an *application*.** |

If the difference between application and applet is clear then you should realize that an applet could be much larger than an application. Now in reality it is true that most applications are larger than applets; it is just important to know that this is not a requirement. You must also understand that a web page can have multiple applets inserted throughout the entire website. Remember that it is not possible to create an entire website with Java. The Java applet or applets must operate inside an HTML environment that controls the website.

**2.6 Using Java Software**

Oracle provides the necessary software to compile and execute Java programs. This is called the JDK (Java Development Kit). In 2009 Oracle bought Sun Microsystems so the software is actually downloaded from Oracle’s website. Sun/Oracle allows you to download this software free of charge, subject to a variety of legal conditions. Basically, to write Java programs you need an editor, a compiler and an interpreter.

|  |
| --- |
| The Basic Java Tools |
| 1. **A text editor to write Java program source code** 2. **A compiler to translate source code into bytecode** 3. **An interpreter to translate and execute bytecode** |

The Java software provides you with the necessary compiler and interpreter software for free. The text editor is where there are many choices. There is a choice between a text editor that is part of an integrated environment and a stand-alone text editor. There are various software packages on the market that combine the editor to write the program with the Java translators and various other tools to develop Java programs. Such software has a complete *Integrated Development Environment* or **IDE** for Java. Many of the Java IDEs need to be purchased, but there are also some excellent free IDE products on the market.

To truly understand the convenience of an IDE you will first need to understand what is involved with Java program development using a stand-alone text editor. You need to start with a fundamental text editor. It is not necessary that this is a full-fledged word processor. There is no need to underline program statements, display words in different font sizes or spell-check the program. Underlining Java code will confuse the compiler and spell-checkers are only familiar with a human language like English, not a programming language like Java. A plain-old editor like **Notepad** is sufficient, and such a text editor comes free with any type of windows environment. The text editor is used to create, load and save Java programs. All Java source programs end with the **.java** suffix.

Now it is necessary to use the Java compiler to create bytecode. The Java compiler can be accessed at the *Command Prompt*. If the Java compiler is happy with the syntax of your program, the source code will be translated into bytecode. All bytecode files will end with a **.class** suffix. If some mistake is made in the sentence structure or syntax of your program statements, the compiler will respond with some error messages. At this stage you need to return to the text editor to identify and fix your errors.

It is this process of going back forth between text editor and compiler that is avoided by the IDEs of a variety of software products. The IDE creates the program and also compiles and executes the program. If a mistake is made in the creation of the program, the IDE automatically returns you to the editor to find your mistake. You will not need to be too concerned about using the rather tedious *text editor and command prompt* since you are going to be using an IDE.

|  |
| --- |
| **The Three Steps of a Java Program** |
| **1. Write the Java source in some text editor.**  **The source code file must end with .java.**  **2. Translate the source code file with a Java compiler into**  **an intermediate bytecode file that will end with .class.**  **3. Execute the bytecode file with a Java Virtual Machine**  **(JVM) program, which is an interpreter.**  **This is also known at the Java Runtime Environment or (JRE) program.** |
| **Note: All three of these steps can be done with a text editor and the command prompt or an Integrated Development Environment (IDE), like JCreator.** |

Instructions on how to install this software is provided in separate PowerPoint presentations. These files are located in the **InstallingJava** folder. The first file in this folder is **InstallingJava-DownloadFiles.ppt**. This is for people who are starting with nothing. It takes you through the steps of downloading and installing the JDK, downloading and installing JCreator, and making all the appropriate adjustments to the JCreator settings. This can be used to install Java at home or pretty much anywhere. The second file is **InstallingJava-ProvidedFiles.ppt**. In many schools, like John Paul II High School, the teacher has already downloaded the necessary files, which are **jdk-7u3-windows-i586.exe** for the JDK and **jcrea500\_setup.exe** for JCreator. The files still need to be executed to install the software and the JCreator settings still need to be set.

There is a very common mistake made at this point. Students know we are using JCreator software to write the Java programs, so they install JCreator and completely forget about the JDK. Since JCreator is an IDE, it gives the impression that everything happens in JCreator. In reality JCreator does not function without the JDK. When you click the *Build* icon in JCreator it is the JDK that compiles your program. When you click the *Run* icon in JCreator it is the JDK that executes your program. The point is the JDK must be installed first before JCreator is installed.

Some people are in labs where the JDK and JCreator are already installed… or they have a laptop and their school has already installed the software. OK fine, but in many cases the JCreator settings still need to be done. For this situation there is one more PowerPoint available called **InstallingJava-SettingsOnly.ppt**.

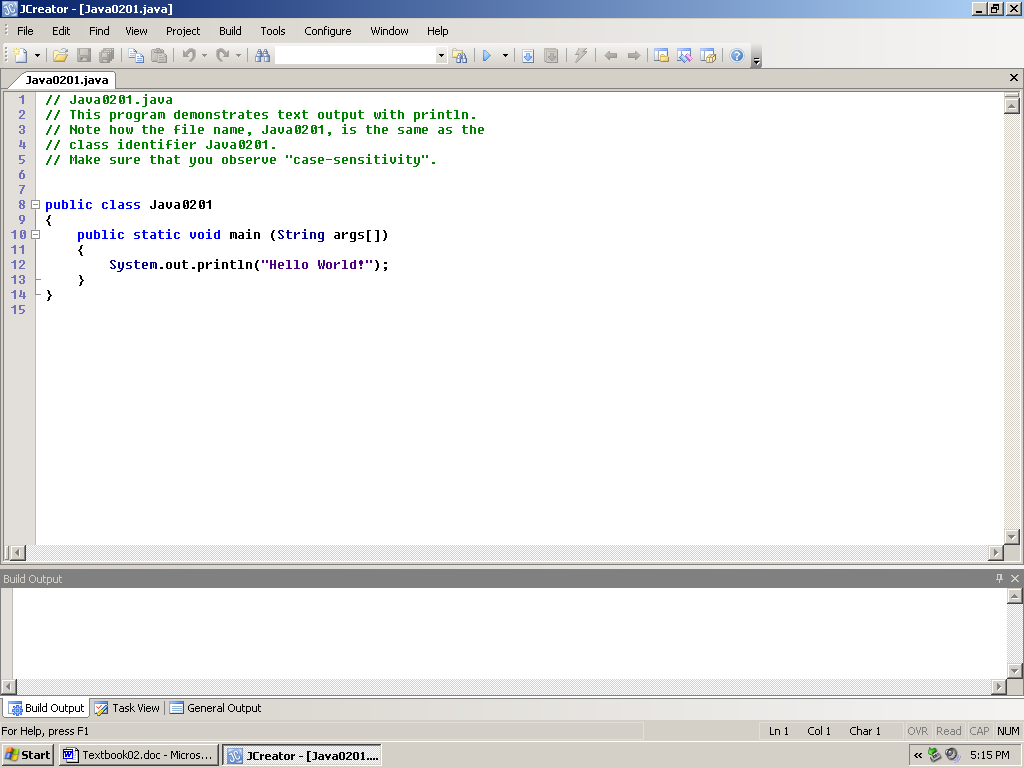
Please keep the following in mind. This chapter was updated in Jan 08, 2012. At that time the most recent version of the JDK is Java Version 7 Update 2, which is why the file is named **jdk-7u2-windows-i586.exe**. It is entirely possible that by the time you are reading this, there might be another update and the file name may be slightly different.

|  |
| --- |
| **Important Java IDEs Note** |
| **This version of Exposure Java is meant for AP Computer Science students. The focus of this course is on computer science with the programming language Java.**  **Separate files are provided for installing Java software. In this chapter JCreator is mentioned since it is the IDE used by the author in his classroom. Other popular IDEs are:**  **BlueJ**  **DrJava**  **Eclipse**  **JBuilder**  **NetBeans** |
| **The program examples in this textbook and with the lab assignments are not IDE dependent.**  **This means that any Java files can be used by any IDE.**  **The “screenshot” appearance of some program examples, like shown in figure 2,2, is the “JCreator” view, but the actual Java program is functional in any other IDE.** |

**2.7 Text Output With print & println**

If JCreator has been set up properly, you should see the display in figure 2.2 when you load **Java0201.java**. The **0201** in the file name comes from the fact that this is the first program example in chapter 02.

**Figure 2.2**

****

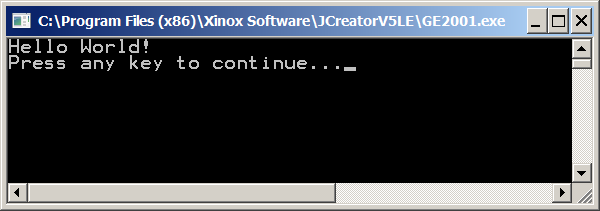
While figure 2.2 shows you exactly what is on the screen, it has a couple problems. The program itself is what we want to focus on and that is hard to see. Also, complete *Graphics User Interface* (**GUI**) displays consume a lot of computer memory and also take up a lot of space on the pages for printed versions of this book. Program examples will be shown in a table box followed by an output textbox. **Java0201.java** is shown again in this format in figure 2.3. The program source code and the program execution output will be shown in an **Arial** font style.

We will now continue and focus on the elements of text output. As a matter of fact, we are only looking at output only. Program input is actually more complex than program output and will be introduced in a later chapter.

**Figure 2.3**

|  |
| --- |
| // Java0201.java  // This program demonstrates text output with println.  // Note how the file name, Java0201, is the same as the  // class identifier Java0201.  // Make sure that you observe "case-sensitivity".  public class Java0201  {  public static void main (String args[])  {  System.out.println("Hello World!");  }  } |

**Figure 2.4**



First I wish to clear up one possible point of confusion. When text programs are executed in JCreator, the words *Press any key to continue...* are automatically added to the end of the output. This is shown in figure 2.4. Once you press a key the output window will close.

The time has come to take a closer look at a Java program and start to digest what is happening. Program **Java0201.java** is displayed yet again in figure 2.5, but this time, numbers are supplied for each line in the program to make an easy reference. Please realize that the numbers are **not** part of the program.

**Figure 2.5**

|  |
| --- |
| **#1** // Java0201.java  **#2** // This program demonstrates text output with println.  **#3** // Note how the file name, Java0201, is the same as the  **#4** // class identifier Java0201.  **#5** // Make sure that you observe "case-sensitivity".  **#6** public class Java0201  **#7** {  **#8**  public static void main (String args[])  **#9**  {  **#10**  System.out.println("Hello World!");  **#11** }  **#12** } |

|  |  |
| --- | --- |
| **Line Numbers** | **Brief explanation of the program statement** |
| Lines #1 - #5 | These five statements are comments for the program. The comments are not compiled into bytecode, nor will they ever be executed. The two **//** indicate what follows is a comment. |
| Line #6 | This statement starts your program. Use the Java keywords **public class** and follow them with the program name. |
| Line #7 | This is the opening brace of the program. Braces contain program blocks. |
| Line #8 | Every application program has a main segment.  You will use the **public static void main** **(String[] args**)heading with every application program. |
| Line #9 | This is the opening brace of the main module block. |
| Line #10 | Here is the actual program statement that executes something. The keywords **System.out.println** display the characters placed between the double quotes that follow. |
| Line #11 | This is the closing brace of the main module block. |
| Line #12 | This is the closing brace of the program block. |

A program language cannot use human language. Human language is idiomatic, ambiguous, and the meanings of words change. When I first learned English, *grass* was something you mowed, *bad* was coming home late after curfew and *gay* was fun loving and spirited. A program written with those words forty-five years ago would not do very well today.

Writing programs has close similarities to writing an essay. There is an important creative part that revolves around telling a story, debating a point or explaining a concept. An essay is made up of individual sentences that follow proper sentence structure, and each sentence is made up of individual words. It is not possible to create an effective sentence without knowing the meaning of the words that make up the sentence. Likewise in a program language you need to know the meaning of the special words that perform some specific purpose. Right now we shall call all the special words used by Java, *keywords*.

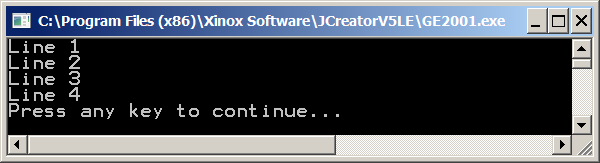
|  |
| --- |
| **Java Keywords and Program Statements** |
| A Java **keyword** is a word that has a special meaning in  a program or performs a special function.  *Two* or more keywords combine to make a program  statement.  Keywords in Java are case-sensitive. This means that  **print** is a Java keyword, which is not the same as **Print**. |

Have you seen many keywords so far? You certainly have and these keywords include **public**, **class**, **static**, **void**, **main**, **String** and **System.out.println**. This is already a pretty good size list for starters. At this stage assume that all these keywords, except for **System.out.println** are necessary in every program and an explanation of their purpose will come at a later time.

In this section you need to learn about **println**. You did notice that **println** is attached to some other keywords with a period separation. **System.out** precedes **println** and you probably suspect that the periods indicate some type of relationship. You are quite right and **System.out** means *the output of the system*. The third keyword, **println**,displays output and a carriage-return/line-feed (**crlf**), as is shown by program **Java0202.java**, in figure 2.6.

**Figure 2.6**

|  |
| --- |
| // Java0202.java  // This program demonstrates how to display four lines of text  // using the <println> keyword.  public class Java0202  {  public static void main(String args[])  {  System.out.println("Line 1");  System.out.println("Line 2");  System.out.println("Line 3");  System.out.println("Line 4");  }  } |

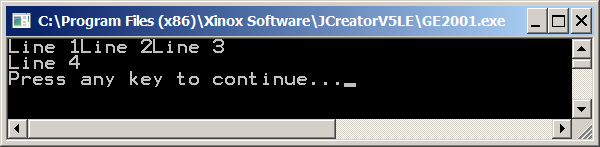


The program output shows that each println statement generates output on its own line. You might mistakenly think that this is caused by the fact that there are four println statements on four separate lines. Such an assumption is logical, but incorrect. Check out the next program example in figure 2.7. This program is almost identical to the previous program. There is one small, significant difference. Two **println** keywords are now changed to keyword **print**.

**Figure 2.7**

|  |
| --- |
| // Java0203.java  // This program demonstrates the difference between <print> and <println>.  // The <println> command adds a "line feed" after the output display.  public class Java0203  {  public static void main(String args[])  {  System.out.print("Line 1");  System.out.print("Line 2");  System.out.println("Line 3");  System.out.println("Line 4");  }  } |

**Figure 2.7 Continued**

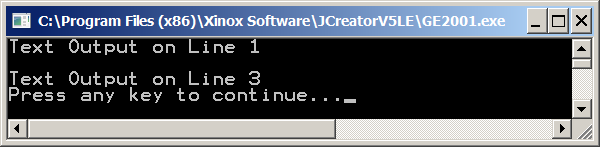


In program **Java0203.java** there are four separate program statements for each **print** and **println** keyword, but the output is displayed on two lines. The **print** keywordgenerates output without a *carriage return* and the **println** keyword does include a *carriage return*.

Do you also notice in the output that the actual characters that are displayed are the same characters that you see in the program statements between double quotes after the **println** or **print** keywords? You may decide that it is necessary for **println** or **print** to finish the statement with parentheses that contain some string of characters to be displayed. This is not always a requirement. Program example **Java0204.java**, in figure 2.8, includes one program statement where println is followed by only a set of empty parentheses. It appears that such an "empty" println statement still generates a carriage return only.

**Figure 2.8**

|  |
| --- |
| // Java0204.java  // This program shows how to skip a line between statements.  // Using <println> with empty parentheses will generate  // a carriage-return/line-feed.  public class Java0204  {  public static void main (String args[])  {  System.out.println("Text Output on Line 1");  System.out.println();  System.out.println("Text Output on Line 3");  }  } |



|  |
| --- |
| **System.out.println and System.out.print** |
| Both keywords **println** and **print** generate an output display of characters contained between double quotes.  Both **println** and **print** follow keywords **System.out**.  **println**("Java is an island in Indonesia.") will display:  Java is an island in Indonesia.  **print**("Java is an island in Indonesia.") will also display:  Java is an island in Indonesia.  The keyword **println** generates an output display followed by a *carriage-return/linefeed* (crlf).  The keyword **print** generates an output display without a **crlf**.  The statement **System.out.println();** generates a  **crlf**, meaning skip a line, without any other output display. |

**2.8 Program Compile Errors**

One of the most difficult aspects of computer programming for beginning students is handling error messages. Chapter 2 is not the place to explain program errors in detail, but you will see two program examples to give you an idea how the Java compiler responds when a program does not compile. The first example shows an error that occurs when a program's file does not match its class name.

You have learned or observed that every Java program appears to have a statement like **public class Java0205**. You may also have observed that the name, which follows **class**,called an identifier, is the same name as the source code file name. Basically, you accept such a directive and do not get too excited.

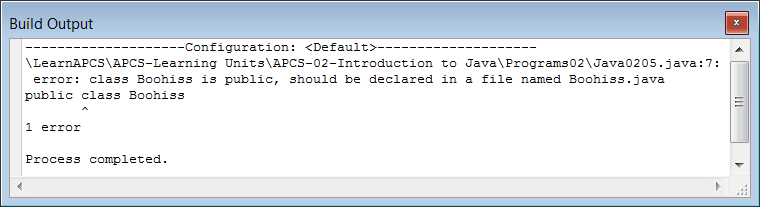
The next program example is stored as program **Java0205.java**. This source code is compiled into bytecode using the exact same procedure as any of the previous programs, and you probably do not expect any indication of errors. Figure 2.9 shows that the name following class is now **Boohiss**, which is noticeably different from the convention of using **Java0205**. Any ideas what the output will be? Are you expecting a few lines of text following the logic of the **println** keyword? Or do you expect maybe something else? Most observant students will know something is happening. Why does Mr. Schram make such a big deal over this class name and file name if it would not make any difference?

**Figure 2.9**

|  |
| --- |
| // Java0205.java  // This program demonstrates that the file name of a program  // and the public class name must be identical.  // This program will not compile.    public class **Boohiss**  {  public static void main (String args[])  {  System.out.println("The bytecode file name");  System.out.println("will be the same as the");  System.out.println("public class identifier.");  }  } |

Figure 2.10 shows the output of program **Java0205.java**. Output may be an incorrect term. You are treated to an error message, which tells you in no uncertain terms that the file name and the class name should be identical.

**Figure 2.10**

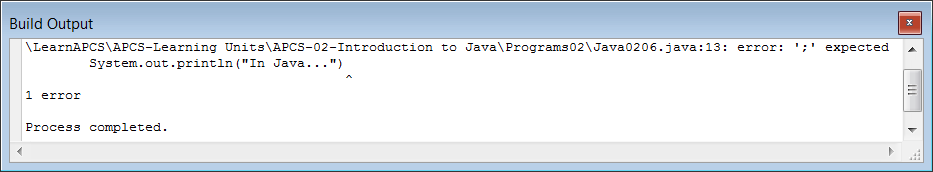
****

|  |
| --- |
| **File Names and Class Names** |
| The external file name of your program must be identical to the **public class** name inside your program, minus the **java** extension. For example:  If you use **public class Howdy** in your program then you must save the program with file name ***Howdy.java***. |

The next program example is a more typical example of a program with some small error that prevents the creation of a bytecode file. You do need to realize that the compiler first checks to see if the program is written with correct program statements and then proceeds to the second step by creating a bytecode file. Program **Java0206.java**, in figure 2.11 does not compile. Can you see what is wrong?

**Figure 2.11**

|  |
| --- |
| // Java0206.java  // This program has an intentional mistake.  // The output window indicates an error and the program does not execute.  // Many error messages provide important clues to help fix the problem.  public class Java0206  {  public static void main (String args[])  {  System.out.println("In English...");  System.out.println("Every sentence ends with a period (.)");  System.out.println("In Java...")  System.out.println("Every statement ends with a semicolon (;)");  }  } |

****

Your main mission in this chapter is to learn how to compile and execute programs. The previous two program examples are provided to let you know that executing a program will not be possible unless the compile stage completed successfully. If you see error messages then execution will not be possible.

Program **Java0206.java** contained a small error. At the end of a statement some required punctuation, a semi-colon, was left out. Java program statements all need to end with semi-colons. If you look at the error message, you will note that the Java compiler did a good job identifying the problem.

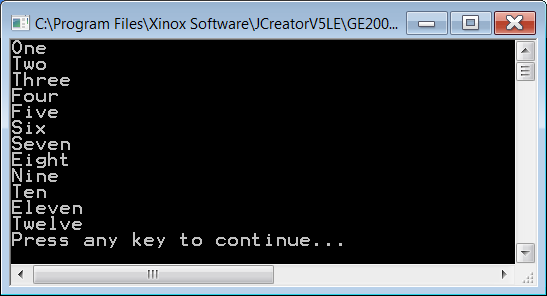
**2.9 Program Comments**

All of the programs that we have looked at have begun in the same way. They all begin with a few lines of text that briefly explain the program. In the first program these were called *comments*. What exactly are *comments*? What purpose do they serve? Well, they are totally ignored by the compiler. They are not compiled. They are not executed. So what good are they? Comments are very useful because they aid in *Program Documentation*. A well-documented program is easier to read, debug, update and enhance. The proper use of comments is only the first part program documentation. You will learn more about this in later chapters.

In Java, we actually have 2 kinds of comments. The first is what we have been using all along. It is the *Single-Line Comment*. When you begin a line with a *double slash* ( // ) the entire line is ignored by the compiler. Just that line and no other. This is demonstrated in program **Java0207.java** in figure 2.12. As with the earlier programs, there are several single-line comments at the beginning of the program. This is not the only place comments can be used. They can be used anywhere in the program. You should see that a comment can be placed right after a program statement. At the end of the program you see that an entire program statement can be *commented-out*. This is useful if a program statement has an error that you cannot fix right now. The statement can be commented-out and you can check the rest of the program.

**Figure 2.12**

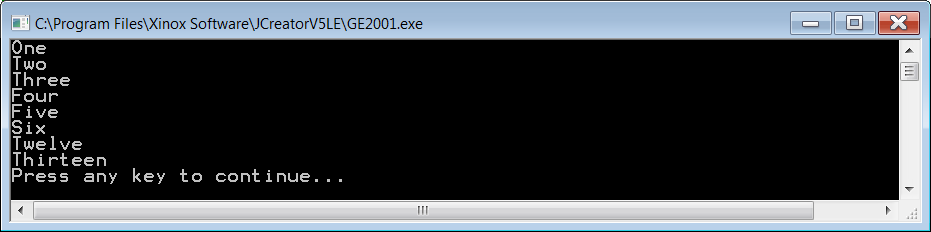
|  |
| --- |
| // Java0207.java  // This program displays several number words.  // The focus now is on program comments.  // Program comments aid in "program documentation" and make your program more readable.  // Every line that begins with a "double slash" is considered a "comment" by the Java compiler.  // The Java compiler ignores all comments. They are not compiled. They are not executed.  // If a line begins with a double slash, it is simply ignored by the compiler.  // That is precisely what is happening with the first 12 lines of this program.  // Note below that a comment can also be placed in the middle of the program.  // They can even be placed right after a program statement on the same line.  // Note: The word "Thirteen" is not displayed because it has been "commented-out"  // possibly by someone suffering from Triskaidekaphobia (the fear of the number 13).  public class Java0207  {  public static void main (String args[])  {  System.out.println("One");  System.out.println("Two");  System.out.println("Three");  System.out.println("Four");  System.out.println("Five");  System.out.println("Six"); // half a dozen  System.out.println("Seven");  System.out.println("Eight");  System.out.println("Nine");  System.out.println("Ten");  System.out.println("Eleven");  System.out.println("Twelve"); // one dozen  // System.out.println("Thirteen"); // one baker's dozen  }  } |



It was mentioned that there are 2 kinds of comments. The first is the *single-line* variety. The second is the *Multi-Line Comment*. This type of comment is shown in program **Java0208.java** in figure 2.13. A *slash* followed by an *asterisk* ( /\* ) is used to mark the beginning of the multi-line comment. An *asterisk* followed by a *slash* ( \*/ ) is used to mark the end. Be careful not to forget to *end* your multi-line comment. If you do, the entire remainder of your program becomes one big comment. Multi-line comments are useful when entire sections of your program need to be *commented-out*.

**Figure 2.13**

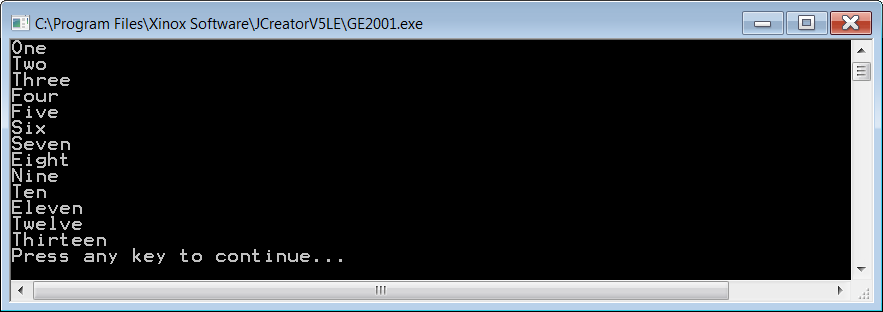
|  |
| --- |
| // Java0208.java  /\* This program is very similar to the previous program, but the output will be different.  Instead of displaying all number words from "One" through "Thirteen",  it will only display some of them.  There are actually 2 kinds of comments in Java.  "Single-Line Comments" are the type you saw in the last program.  They begin with a double slash.  Java also has "Multi-Line Comments".  To create a multi-line comment, you begin with a slash and an asterisk.  You conclude a multi-line comment with an asterisk followed by a slash.  This can be very useful in program development.  This time the output will not show the words "Seven" through "Eleven" because they  have been "commented-out" with a multi-line comment. \*/  public class Java0208  {  public static void main (String args[])  {  System.out.println("One");  System.out.println("Two");  System.out.println("Three");  System.out.println("Four");  System.out.println("Five");  System.out.println("Six"); // half a dozen  /\* System.out.println("Seven");  System.out.println("Eight");  System.out.println("Nine");  System.out.println("Ten");  System.out.println("Eleven");  \*/ System.out.println("Twelve"); // one dozen  System.out.println("Thirteen"); // one baker's dozen  }  } |



One more program example will be shown. Program **Java0209.java** in figure 2.14 shows another use for a multi-line comment. It can also be used to create a *heading* in your program. Right now we tend to grade student program on the monitor, but in the past when we required students to print their program, the students would need to include a heading using this technique.

**Figure 2.14**

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* \*  \* Java Program 0209 \*  \* \*  \* Numbers from 1-13 \*  \* \*  \* By: John Schram \*  \* \*  \* 3/15/12 \*  \* \*  \* This program is similar to \*  \* the previous two and shows \*  \* that a comment can be used \*  \* to create a heading. \*  \* \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  public class Java0209  {  public static void main (String args[])  {  System.out.println("One");  System.out.println("Two");  System.out.println("Three");  System.out.println("Four");  System.out.println("Five");  System.out.println("Six"); // half a dozen  System.out.println("Seven");  System.out.println("Eight");  System.out.println("Nine");  System.out.println("Ten");  System.out.println("Eleven");  System.out.println("Twelve"); // one dozen  System.out.println("Thirteen"); // one baker's dozen  }  } |



**2.10 Escape Sequences**

There are many different ways to generate output in Java. The variety of graphics output displays are especially impressive. Right now in your early text-output stage variety is limited. It is possible to control output to a small degree with *escape sequences*. An *escape sequence* is a single slash followed by some character placed between double quotes.

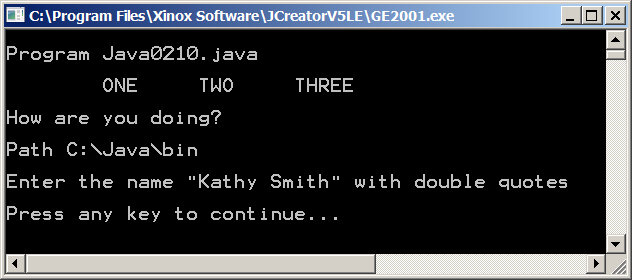
Java recognizes the single slash and does not display it, as you would expect. After all, characters between double quotes are literally displayed. That is the whole point of a literal string. The character following the single slash determines the nature of the special output. For instance the sequence **\n** will generate a carriage-return, line-feed. Program **Java0210.java**, in figure 2.15 shows five of the more common escape sequences. You will especially find that the **\n** and **\t** sequences are particularly useful.

Look carefully at the output. You will not see the string **Hello John** anywhere. It is in the program on line 17. However, the **/r** escape sequence does not provide a line-feed. This results in **Kathy Smith** covering up **Hello John**.

**Figure 2.15**

|  |
| --- |
| // Java0210.java  // This program demonstrates escape sequences in Java  public class Java0210  {  public static void main (String args[])  {  // \n performs a carriage-return and a line-feed  System.out.println("\nProgram Java0210.java\n");  // \t performs a horizontal tab  System.out.println("\tONE\tTWO\tTHREE\n");  // \r performs only a carriage-return without a line-feed  System.out.print("Hello John\r");  System.out.println("How are you doing?\n");  // \\ displays a single backslash  System.out.println("Path C:\\Java\\bin\n");  // \" displays double quotes  System.out.println("Enter the name \"Kathy Smith\" with double quotes\n");  }  } |

**Figure 2.15 Continued**



**2.11 The AP Java Subset**

This textbook can be viewed as an introductory computer science textbook, which uses the Java programming language. This is certainly true, but a deliberate effort is made to prepare students for the AP Computer Science Examination.

|  |
| --- |
| Please be advised that both **AP** and **Advanced Placement Program** are trademarks or registered trademarks of the **College Entrance Examination Board**, which was not involved with the production of and does not endorse this book. |

The College Board works together with **Educational Testing Service** to create the **AP Computer Science Examination**. In an effort to create a standardized test for all students, the College Board and ETS have created an **AP** **Java Subset**. The subset is very necessary. The number of available topics in Java is absolutely enormous and decisions need to be made which topics are more significant and should be tested. There is also a secondary consideration. Some topics are very difficult to test in a manner that is fair to all students. The AP Java Subset created by the *Test Development Committee* of the College Board does not dictate what should be taught in an AP Computer Science course. The AP Java Subset indicates what will be tested.

This textbook, like practically every other textbook on this subject, will include topics that are very worthwhile to know, but will not be explicitly tested. In an effort to clarify to teachers and students you will see special warning boxes in appropriate chapters, which indicate on the current chapter those topics that will be tested and which topics will not be tested. These alert tables will be used for various other reasons as well to help prepare students for the AP Computer Science Examination. You will see these alert tables starting in Chapter III.

**2.12 The Responsible Use of Computers**

Computers have been made simpler in so many ways. Tedious computations are now performed by computers. Creating any type of writing is so much simpler with a word processor. International travel reservations can be made in minutes, rather than in days. Communication with anybody in the world is now instantly possible and it is very cheap. Information about any conceivable topic can be found on the Internet. And yes, let us not forget that there is now an explosion of electronic games that can be played on computers.

Unfortunately, computers can be used incorrectly as well. Electronic software is easy to copy and many people are tempted to help themselves to a free copy of software. Electronic transfer of money can be abused and computers have created a level of white-collar crimes that were unimaginable before. The Internet provides connections for millions of people, but it also makes it possible to collect private information from people. A whole new criminal area of identity theft has evolved. Computers are marvelous tools, but at the start of this course when you are about to learn some serious computer information, you need to stop and think about the responsibility and consequences of using computers.

This chapter section is designed to make you think and discuss some important computer issues. **Do not take this topic lightly**. I have experienced many students who have lost hundreds of hours of computer work, because they neglected to back up their efforts. I know several students, who have felony records because they made some seriously bad decisions with their computer knowledge. The evening news also reports cases regularly of naive young teenagers who put their private information on the Internet and find themselves getting involved with adults who take advantage of children who are clueless about the consequences of providing private information to an unknown audience.

Using computers responsibly involves two major categories. What can happen to your computer and you personally and what can you do to other computers and other people. Hopefully, this section will make you realize that you have a tremendous amount of control over using computers properly, and you will neither be a victim nor cause somebody else to become a victim.

**Protect Your Computer From the Environment**

Computers and computer information are vulnerable. Get used to it. Any electronic device, computers included, does poorly when it is mishandled. Computers should not get too hot or too cold. They should not be dropped down a stair case. You think I am kidding? I have observed three occasions where a student dropped a laptop computer down a staircase. I have seen a student lift up a laptop by the power cord, only to watch it become unplugged and obey the laws of gravity. I have seen a student walk on top of a computer. I have seen a student walk off with a computer connected to a power cord and watch the computer slip out off the holding hands and fall down. I have observed multiple cases of drinks that spilled on computer keyboards, making the keyboard very sticky at best and totally non-functional at worst.

Be aware. For starters do not place a computer where it can fall from a desk or table. Place cords in such a way that people will neither trip nor pull computers down. Do not keep drinks of any kind anywhere near a computer.

Physical damage to a computer happens, but it is not the biggest problem. The number one source of grief for many people, especially students, is loss of data. All computer data is stored in some electronic fashion. For starters the essay you are writing is only stored in the Random Access Memory (RAM) of the computer. The RAM holds information while the computer is powered up. If there is a power outage, no matter how brief, you information will be gone. If you cleverly forget to save your work then it will also be gone. Everything you create needs some type of permanent storage. Your data can be stored on a hard drive, CD, jump drive, network or a second computer. All these devices store the information even when the computer is turned off, but information is still vulnerable. Hard drives do "crash" and CDs can become dirty or become lost. It is wise to have a second backup for anything important.

Please realize the following: it is very human to think that problems only happen to other people. Other people have accidents. Other people drop their computers. Other people have bad hard drives and lose data. I have actually seen a person get ready to pump gas with a burning cigarette in the same hand as the gas hose. He told me it was fine, because he did it before and nothing happened. The poor kid was not familiar with the "fat chance" theory.

Aside from saving programs regularly and then backing up all your data to some secondary backup device, is there something else you can do? Yes, there is. You can use a "battery backup" system. Such a system is much more than an extension cord with some power surge protection. The battery backup system provides power surge protection and keeps the computer turned on when there is power failure. This allows you to safely store your data. Many of these systems also have *Automatic Voltage Regulation* (**AVR**), which protects your computer from "dirty" electricity that has over-voltage and under-voltage periods.

**Protect Your Computer From Viruses**

Computer viruses are a fact of life. Do you actually know what a computer virus is? It is a special program that has two qualities. First, there is the duplication part. A computer virus can attach itself to your computer, search for email addresses and happily spread to computers on your email address book. The virus can also travel to other computers connected to your computer. So far the virus has only traveled and duplicated. The second part of a virus is the payload. The payload is the actual part of the virus that executes a program and does potential harm. There are mild viruses, which only display some silly message. There are also nasty viruses, which delete files on a hard drive, shut down the computer, alter information and cause major headaches to people who rely on computers.

Why do viruses exist? Well there are people with too much time on their hands who apparently have reasons - in their minds - to give major grief to other people. Sometimes computer programmers, who were fired, sabotage computers with a time-delay virus. This happened to Eastern Airlines in the Eighties. Six months after a major computer programmer was fired, the airline reservation computers lost all their data. Not a single passenger was confirmed on any flight. There are also military reasons. During the Cold War, there were hundreds of computer scientists employed by the USSR to create viruses that would handicap computer use in the West. During the first Iraq war, the United States sold computer equipment to France and France in turn sold this equipment to Iraq. The equipment was sold with a time-delay virus. The day that the U.S. attacked Iraq, the virus disabled the anti-aircraft computer programs in Baghdad. U.S. planes could fly over Baghdad with anti-aircraft missiles paralyzed by ineffective software.

Today almost all computers come equipped with some type of virus software. Frequently, people think that they have proper protection. Here is the problem. All anti-virus software comes with a *virus definitions* file. This file is pretty much guaranteed to be outdated by the time the software arrives in the store and is sold. All anti-virus software alert computer users to get virus updates, but many people ignore these warnings. Do keep in mind that bad things only happen to other people. The better anti-viruses software automatically updates your computer at regular intervals. New viruses are created at an alarming rate.

There are still other access problems invading your computer. Special software, called *spyware*, snoops around your computer and tries to record information. The spyware may be used for various purposes, but one immediate effect of this extra software is a computer slowdown. There also is your good friend SPAM. SPAM are annoying, unwanted email messages and other miscellaneous pop-up windows, that can slow your computer down tremendously. Most modern email programs include protection against SPAM and spyware.

**Protect Your Computer From Improper Access**

In the business world and at school, computers are frequently turned on, logged in and then computer users runs off to go to the bathroom, make come copies, get a drink, get a donut, meet with other people or any other reason that removes them from their computers. While they are gone, their computers can be accessed. In this day, it is easy to slip in a "jump drive", which is a miniature memory stick that plugs into a *Universal Serial Bus* (**USB**) port. These memory sticks are small, but they can hold up to 2 GBs of memory or more and that is plenty of space to store all kinds of data from your computer.

People with some sophisticated knowledge also know how to make your computer remotely accessible. Later, in their own environment, your computer is accessed and you will not even realize what is happening.

In this day-and-age of laptop computers, access becomes simpler. It is easy to steal a laptop computer. During the last school year, on the last day of school, in my own classroom, a computer was stolen. All laptops look pretty much the same. A few have some personal stickers on the laptop, but the reality is that the action of placing a laptop computer in a backpack is a normal action. It does not look suspicious. In this case, a student left the classroom to talk to a student in an adjacent class, to put together an end-of-year team project. When the student returned at the end of class, his laptop computer was gone.

In a school and business environment, technological devices look very similar. The laptop charger and power cords look all alike. There are different brands of jump drives, but the same brands look identical. Frequently, people use CDs without a label on them. What does this all mean? You can easily lose your stuff, because perfectly honest people confuse your property with their property. In other words put labels on your CD, mark your computers, mark your chargers, mark your jump drives.

**The Ethical Use of Computer Software**

It is easy to be outraged when somebody steals your property, copies your files, sends you annoying email and invades your computer with nasty viruses. Is proper behavior different in the other direction? It used to amaze me when my wife had child care that parents had no difficulty dropping off sick children. *He is only a little sick with a small fever. I do not think he will infect anybody,* was a common statement heard in the morning at drop off time. Yet if the child would become sick, there would follow a major *crime scene investigation* to determine which offending child has infected their child.

What happens if your friend has a new game, a new word processing program or any other new type of software? You sure like that new software, but you cannot afford it. At this point the brain kicks in with its brilliant rationalization skills and minutes later you find it completely justified to convince your friend into a copy. "What hurt is there? It is only one copy and the company won't miss that single copy." Another popular rationalization is to claim that the company charges too much for the software. You, the noble person you are, will fight this tyranny by acquiring the software for free. That will teach the company a lesson.

There is an irony here. Department store merchandise costs more because the cost of shoplifting is added the merchandise cost. Likewise, software would actually be quite a bit cheaper if everybody purchased an honest copy for themselves. Is it surprising that so many people think that this is just fine? Perhaps not, if you consider the "what is a conscience" story.

Some years ago I was reading a book of Native American Indian stories. In one story a young Indian boy, *Curious Beaver*, asks his grandfather, *Aging Bear*, what a conscience is. The grandfather replies that a conscience helps you to tell right from wrong. The grandson then asks how does a conscience manage to help a person to know what is right. The grandfather replies:

*My dear grandson, every person is born with a conscience. The conscience is a triangle inside your body, located close to your heart. The three edges of the triangle are very sharp. Anytime that you do something wrong, the triangle turns. As the triangle turns, the sharp edges cause pain. You feel this pain when you behave incorrectly. The worse your deed and the faster the triangle turns, causing greater pain.*

The son appears puzzled and tells his dad that he understands how the conscience works, but he is confused. The boy says that he has seen several people who have performed major misdeeds and they did not seem bothered by any pain. The father looks solemn and becomes very serious.

*You are very observant and very correct for one so young. Yes there are people who commit great sins frequently and yet, they seem unconcerned about the gravity of their acts. You see, the conscience turns a lot when people sin, but the frequent turning makes the sharp edges dull. Soon the triangle can turn freely inside the person and there is no longer pain. It is a great tragedy when somebody has done so much wrong that the conscience is no longer able to help people realize how wrong their actions are.*

Software is sold with simple information about licensing. I have purchased a copy of Microsoft Office Teacher Edition. It states directly on the front of the box, inside a conspicuous green star, that the *product is licensed for noncommercial use on up to 3 home PCs*. Pretty simple to understand.

**Hacking and Other Misdeeds**

There are people who think that visiting other networks and other computers is quite innocent. The innocence is based on the fact that no attempt is made to transfer money illegally. No attempt is made to acquire any private information to assume some else's identity to acquire credit cards and other such evils. No these folks are very nice people - so they claim - who happen to do a little bit of innocent hacking into other systems. Besides, if somebody is dumb enough to allow easy access to their computer then they deserve this hacking business.

You may or may not accept this type of explanation, but there is a fundamental problem. There are plenty of hackers who have sufficient knowledge to break into computer systems. Frequently, this knowledge is based on hacking software that is downloaded from the Internet. You now have somebody who is smart enough to figure out how to break into a system with some special program, but in reality these hackers have no clue how their tools work. You can assist somebody else to do considerable damage to computer systems by releasing viruses you do not even realize are attached to the free hacking programs.

There are other problems, which are not hacking. Consider the following. There was a student several years ago who decided to send an email message to all the teachers in the school. This was not hacking. The email addresses of teachers are advertised on the school's web site. His message was the following:

*I know where you are and I know what you are doing.*

Many teachers were relaxed and responded to come over and help cut the grass or grade papers. Other teachers were scared and went to the police feeling they were stalked. A day later the young man, who was having some fun, found himself arrested and he was in a legal nightmare for the next three months. As a minor everything was eventually dismissed in some fashion. He thought that his clever prank had gone away. A year later he was a senior student and suddenly found out that not a single teacher in his school was willing to write a positive college recommendation for him.

**Computer Vandalism**

I will finish this section with vandalism deeds that are quite common. Stealing computers is not such a common problem. It is much more common that keys are altered on a keyboard. It is also common that goofy, unwanted programs appear on computers. Sometimes students alter window titles and replace them with some clever message, usually with an improper choice of vocabulary. This type of misbehavior is minor in its effect, but it is annoying and time-consuming to clean up the mess.

**2.13 Summary**

Sun Microsystems provides free software to compile and execute Java programs. The Java Development Kit (JDK) can be found on Oracle’s website. The current version (as of March 2012) is Version 7.0 Update 3.

The Java program language is designed to create full-fledged program *applications* for industry, as well as *applets* that will operate inside a web page. Normally, an application is a larger program than an applet. However, an applet is not necessarily a small program.

The secret of Java's success is the use of both a compiler and an interpreter. First, the Java compiler is used to create an intermediate stage in program translation, called bytecode. Second an interpreter is used to translate and execute the bytecode. Sun created multiple interpreters for different platforms. The interpreter is called a Java Virtual Machine (JVM). There are three distinct steps in creating a Java program.

The Java compiler and Java interpreter translate and then execute existing programs. You still need some tools to write the Java source code. It is possible to write code with any text editor and then use the command prompt to provide commands for compiling and executing. It is easier to use an *Integrated Development Environment* (**IDE**) like JCreator to assist in program writing. JCreator can be downloaded for free if you use the light edition. JCreator can execute both application programs and applet programs. You need to create a special web page that imports the Java bytecode applet file.

A program is made up of program statements, and program statements are made up of keywords. Two such program statements are **System.out.print** and **System.out.println**. Both of these program statements send text output to the computer screen. The keyword **println** adds a carriage-return/line-feed (**crlf**) to the output statement.

The Java compiler checks to see that every program statement is written with correct syntax before a bytecode file is created. When mistakes are made, the compiler generates error messages about these mistakes.

Computer program are constantly altered. Look at the many versions you see of popular computer software. It is much easier to read, debug, update and enhance a program that is well-documented. Using comments aids program documentation.

Exposure Java will include all the topics of the required AP Computer Science curriculum. The College Board has created an *AP Java Subset*, which is part of the official AP Course Description. The course description can be downloaded from *AP Central*, the official web site of the College Board for AP courses information. The web site is located at:

http://apcentral.collegeboard.com/apc

Computers must be used in a responsible manner. First, care must be taken that a computer is not harmed physically or stolen. Computers are vulnerable and can be easily damaged. Computer data is stored in areas that can be easily corrupted and all important data needs to be backed up in one or more storage areas, like hard drives, CDs or memory sticks. Computers also need up-to-date protection programs against viruses, spyware and SPAM.

Using copyrighted software may be simple, but it is unethical. All software have license agreements that state if the software is free, licensed for one user, multiple users or an entire site, like a school or company. Many people hack into computers, supposedly for fun, unaware that they are inflicting damage with unknown computer hacking programs.

Literal strings will not always display *literally*. Certain special characters following a single slash like **\n**, **\t** will result in a carriage return, a tab or some other special output display. The slash-character combination is called an *escape sequence*.