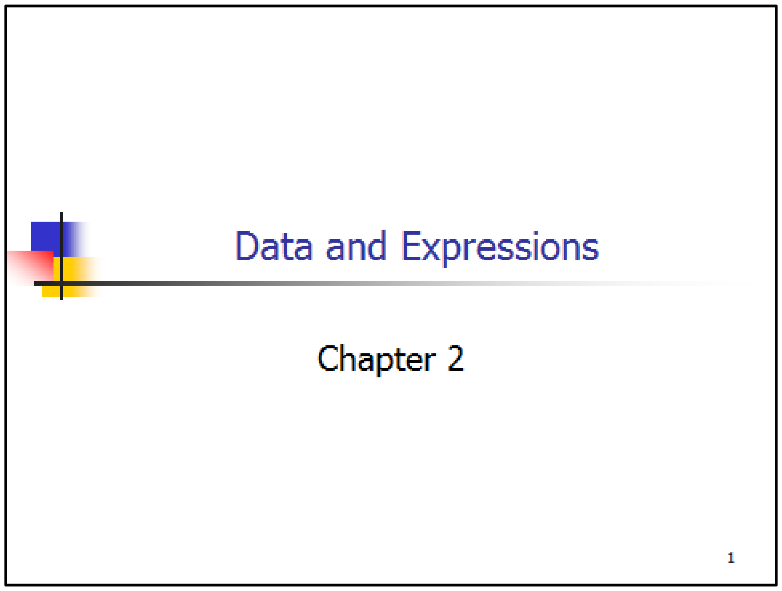


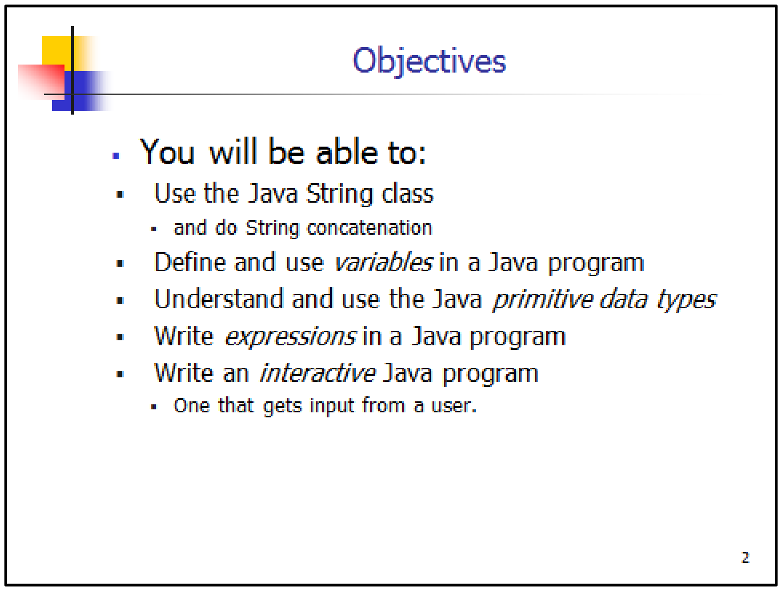
Data and Expressions

Chapter 2

Hello, and welcome to week 2 of Programming Concepts. This is Rollins Turner, and I will lead you through this PowerPoint presentation just as if we were in the classroom.

Last week we concentrated on learning some basic programming concepts and learning to use the tools that you will need for Java programming: a text editor, a Java compiler, and the Java interpreter, also known as the Java Virtual Machine. I demonstrated particularly simple versions of text editors and the Java compiler. We used the very simple “Hello, World” program to get started with the tools.

This week we will start to look at programming concepts in a bit more depth, and take our first steps beyond “Hello, World”.



Objectives

- You will be able to:
  - Use the Java String class
    - and do String concatenation
  - Define and use *variables* in a Java program
  - Understand and use the Java *primitive data types*
  - Write *expressions* in a Java program
  - Write an *interactive* Java program
    - One that gets input from a user.

Here’s what you should get from today’s class.

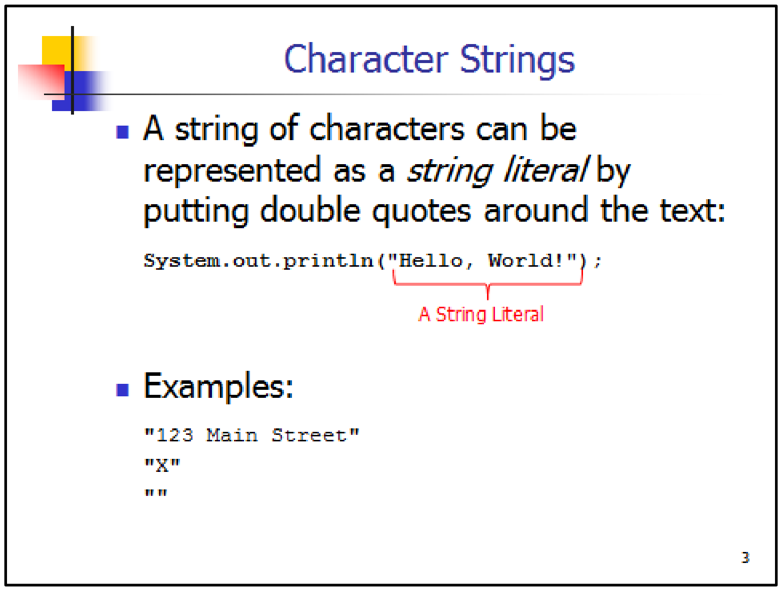
You will be able to use character strings in your Java programs, and will learn how String concatenation works in Java.

You will learn how to use *variables* to put data into your program and how to access and modify the data.

The Java language provides several different forms of variables, which are called *primitive data types*. The word primitive, as used here doesn’t mean prehistoric. It just means simple or basic. Types of data that are built into the language, as opposed to data types created by the programmer.

You will learn how to combine variables to create what we call *expressions*. Expressions consist of two or more variables combined by what we call *operators*. Operators tell the Java Virtual Machine to perform *operations*, such as addition, subtraction, etc. The variables that are to be inputs for an operation are called *operands*. Most operators take two operands, and produce a single result, such as the sum or difference.

And finally you will learn how to make a program interactive, by getting input from the user.



Character Strings

- A string of characters can be represented as a *string literal* by putting double quotes around the text:

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

$\underbrace{\texttt{"Hello, World!"}}_{\text{A String Literal}}$

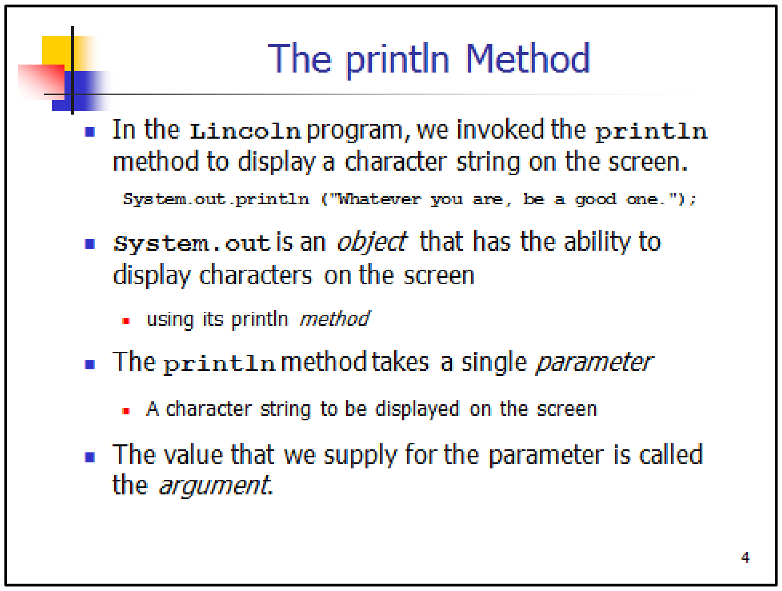
- Examples:

```
"123 Main Street"
"X"
""
```

In a Java program we often need to display strings of characters to the user, as we saw in “Hello, World”. The simplest way to do this is by using what we call a *string literal*. A string literal is just a string of characters surrounded by quotation marks, such “Hello, World”. Another example is “123 Main Street” as shown in this slide.

A string literal can consist of just a single character enclosed in quotation marks, such as “X”. Or it can consist of just the quotation marks with nothing inside, what we call an *empty string*. An empty string is still a string, even though it has no characters. This may seem like a rather useless thing, but as you continue through this course you will occasionally run into a need for an empty string.

By the way, when you are writing a string literal, be sure to use “straight” quotation marks, as shown on this slide. This is the normal quotation mark that is on your keyboard. The curved quotation marks produced automatically by word processing programs won’t work. So long as you are using a programmer’s editor, or any simple plain text editor, this won’t be a problem. But it you copy and paste from a document, you might get the wrong kind of quotation mark.


The println Method

- In the `Lincoln` program, we invoked the `println` method to display a character string on the screen.

```
System.out.println ("Whatever you are, be a good one.");
```

- `System.out` is an *object* that has the ability to display characters on the screen
  - using its println *method*
- The `println` method takes a single *parameter*
  - A character string to be displayed on the screen
- The value that we supply for the parameter is called the *argument*.

4

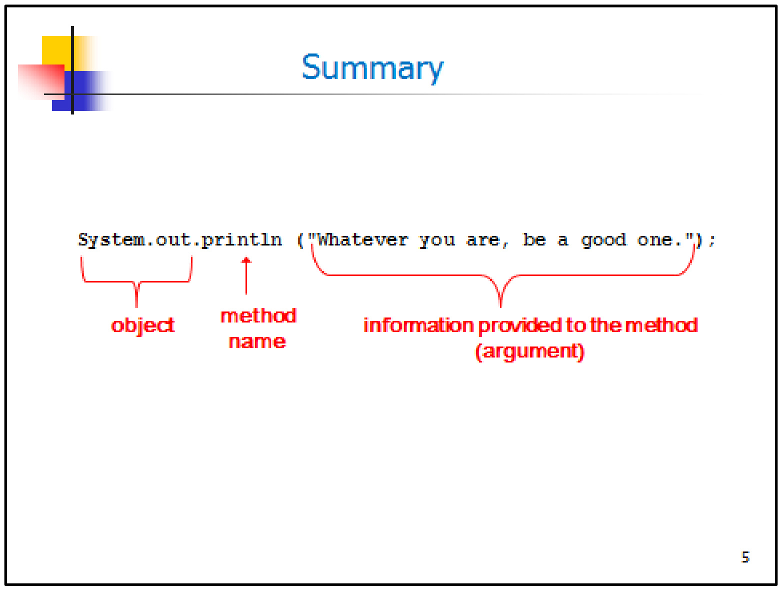
To display a string of characters on the screen, we use the println method.

println is a method of the System.out object, so we write the statement as shown in this slide. First we identify the object, System.out, then write the name of the method, println. There must be a period (or “dot”) between the object name and the name of the method.

When we invoke a method of any object, the name of the method is always followed by a pair of parentheses, which are used to provide inputs to the method. There may or may not be something inside the parentheses, depending on the method.

We refer to a method’s inputs as its *parameters*. A method may have one or more parameters, or it may have none. The println method has one parameter.

We refer to the values that we pass to methods as *arguments*. The distinction between parameters and arguments is fairly subtle, and many programmers use the terms interchangeably. The distinction will make more sense after you have learned more about methods.



Summary

```
System.out.println ("Whatever you are, be a good one.");
```

object
method name
information provided to the method (argument)

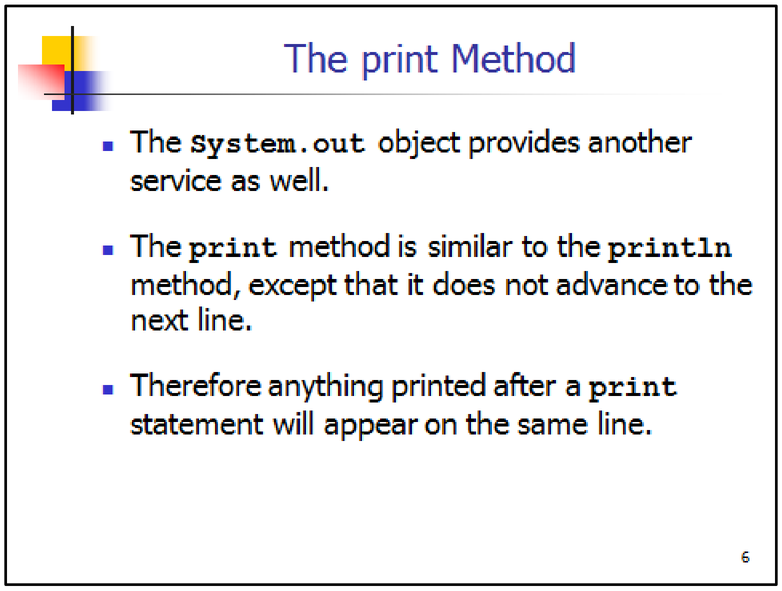
5

This slide summarizes what we have learned about the println method.

System.out is an object that is built in to the Java run time system.

println is the name of a method that outputs a line of text to the screen.

And println takes a string as its one parameter. In this example we provide a string literal as the argument to the println method.



The print Method

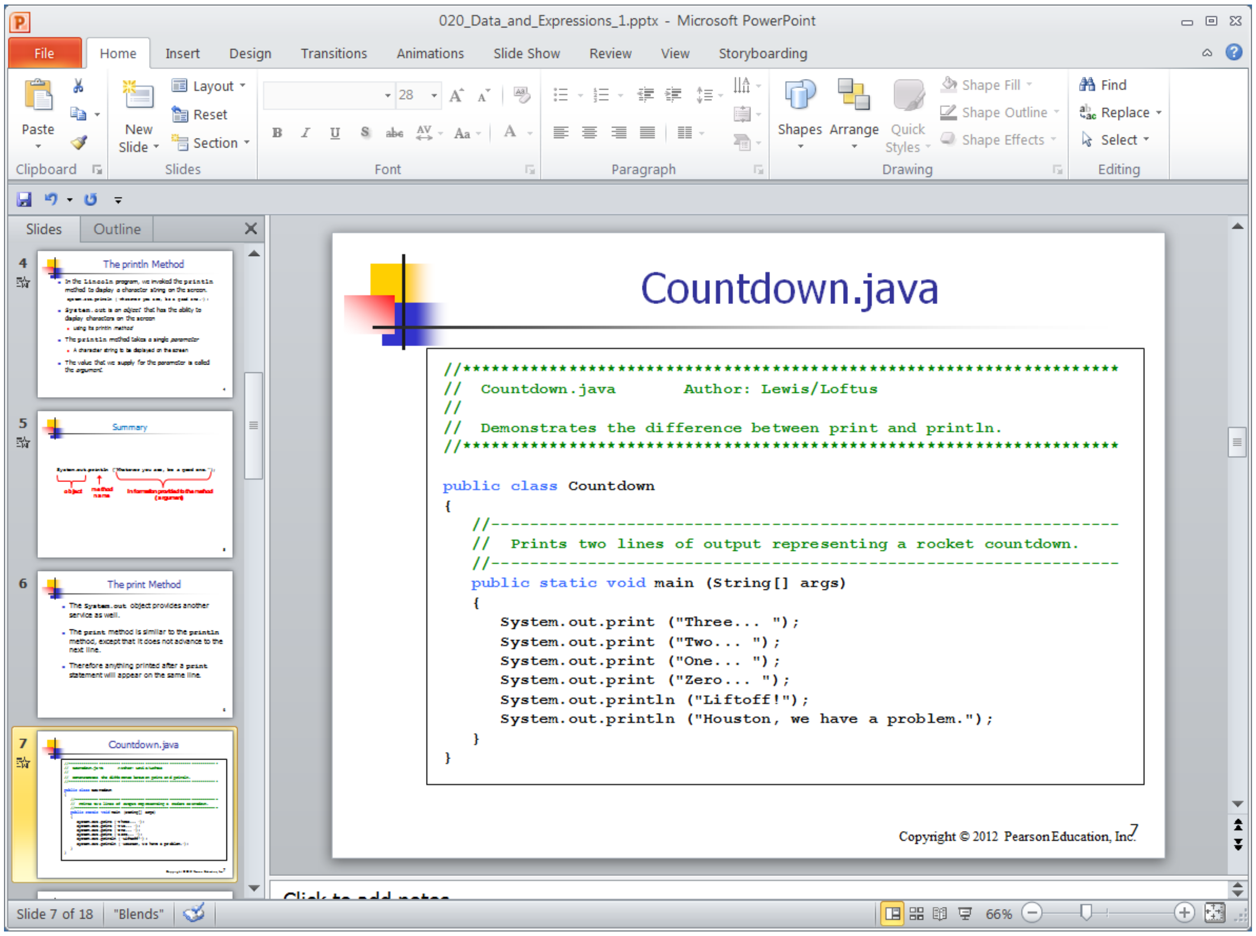
- The `System.out` object provides another service as well.
- The `print` method is similar to the `println` method, except that it does not advance to the next line.
- Therefore anything printed after a `print` statement will appear on the same line.

6

The println method goes to a new line on the screen after displaying the text that we provide. There are often times when we don’t want to automatically go to a new line after displaying some text. The System.out object has another method that works like println but does not go to a new line. This is the plain print method.

If we use the print method to display some text on the screen, the next thing that we send to the screen will immediately follow what we displayed with print.

An example using the plain print method is shown on the next slide.

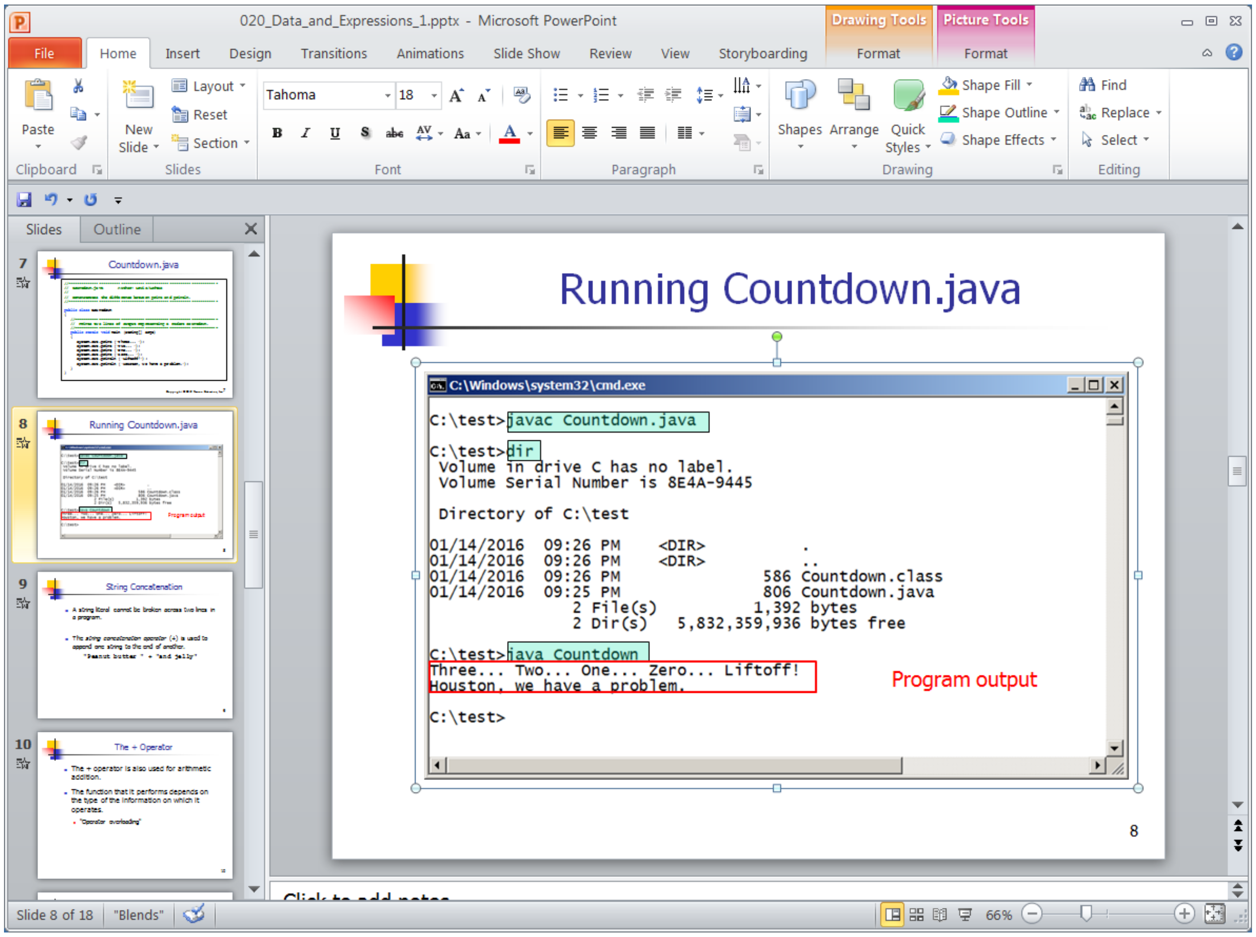


Countdown.java

```
//********************************************************************
//  Countdown.java       Author: Lewis/Loftus
//
//  Demonstrates the difference between print and println.
//********************************************************************

public class Countdown
{
   //-----------------------------------------------------------------
   //  Prints two lines of output representing a rocket countdown.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.print ("Three... ");
      System.out.print ("Two... ");
      System.out.print ("One... ");
      System.out.print ("Zero... ");
      System.out.println ("Liftoff!");
      System.out.println ("Houston, we have a problem.");
   }
}
```

7

(No narration)



Running Countdown.java

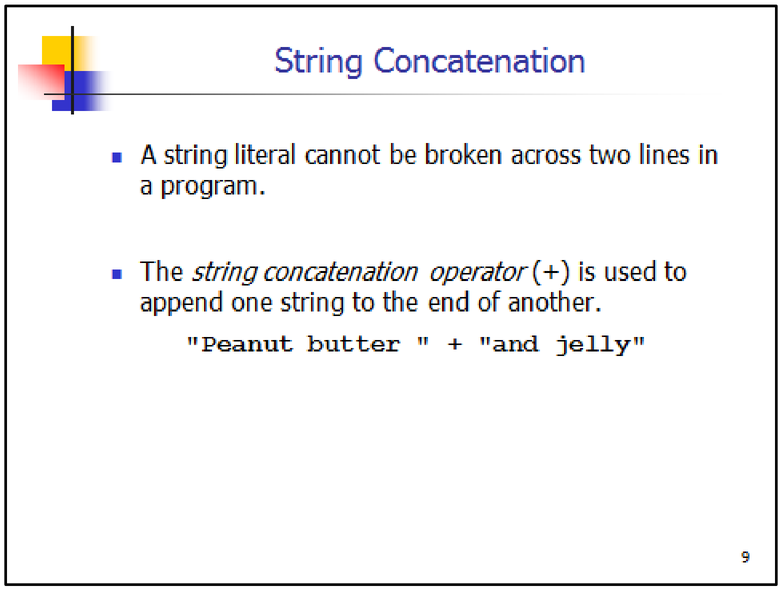
```
C:\test>javac Countdown.java

C:\test>dir
 Volume in drive C has no label.
 Volume Serial Number is 8E4A-9445

 Directory of C:\test

01/14/2016  09:26 PM    <DIR>          .
01/14/2016  09:26 PM    <DIR>          ..
01/14/2016  09:26 PM               586 Countdown.class
01/14/2016  09:25 PM               806 Countdown.java
               2 File(s)          1,392 bytes
               2 Dir(s)   5,832,359,936 bytes free

C:\test>java Countdown
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.

C:\test>
```

Program output

8

This slide shows a screenshot of a command window where I have compiled and run the Countdown program shown on the preceding slide.

Notice in the program output how the output following the output of each call to System.out.print continues on the same line.



String Concatenation

- A string literal cannot be broken across two lines in a program.
- The *string concatenation operator* (+) is used to append one string to the end of another.

```
"Peanut butter " + "and jelly"
```

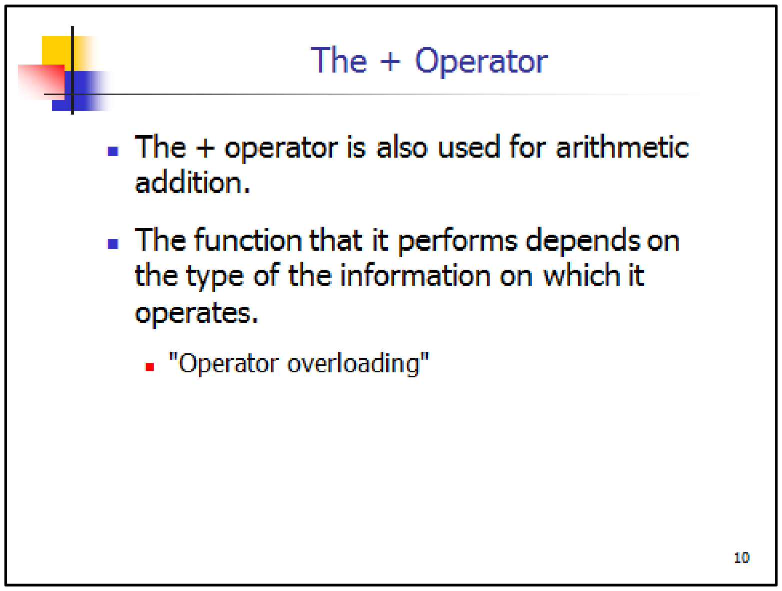
9

In a Java program, a string literal must be on a single line. If we attempt to continue a string literal on the next line, we will get a compiler error.

In order to create strings longer than we can conveniently write on a single line, we can use the *string concatenation operator*, which is written as a plus sign.

Concatenation is a fancy word for appending one string to another.

To concatenate two strings, we write the first string followed by a plus sign followed by the second string, as shown in this slide.



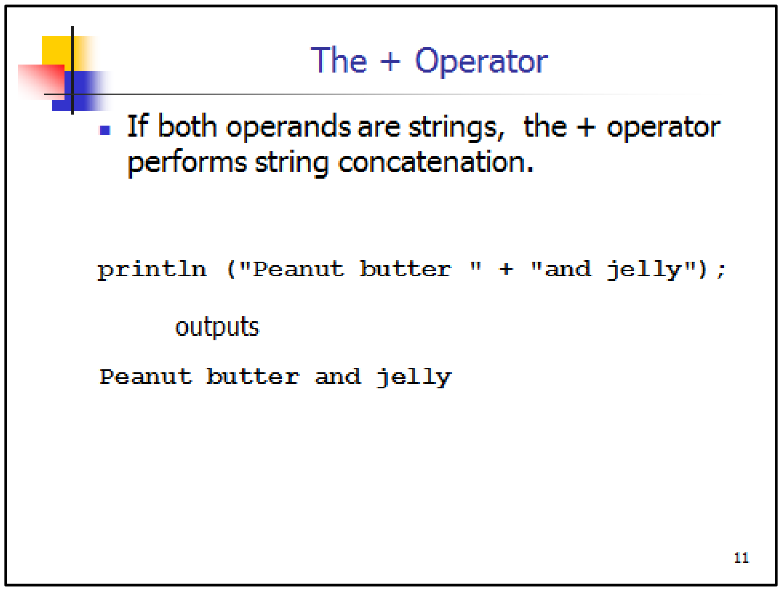
The + Operator

- The + operator is also used for arithmetic addition.
- The function that it performs depends on the type of the information on which it operates.
  - "Operator overloading"

10

Now it turns out that the plus sign is also used to add two numbers together, as we might expect. So how does the Java compiler know which operation we mean when we write a plus sign as an operator? It determines what we intend by looking at the operands.

This is an important concept in Java and many other programming languages, called “operator overloading.” Operator overloading means making a given operator, such as plus sign, do different things according to the kinds of inputs provided.



The + Operator

- If both operands are strings, the + operator performs string concatenation.

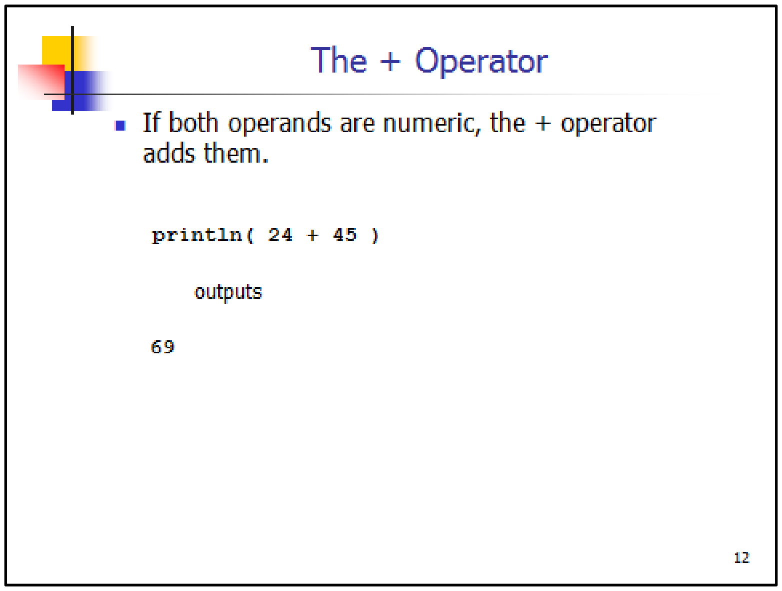
```
println ("Peanut butter " + "and jelly");
```

outputs

```
Peanut butter and jelly
```

11

(No narration)



The + Operator

- If both operands are numeric, the + operator adds them.

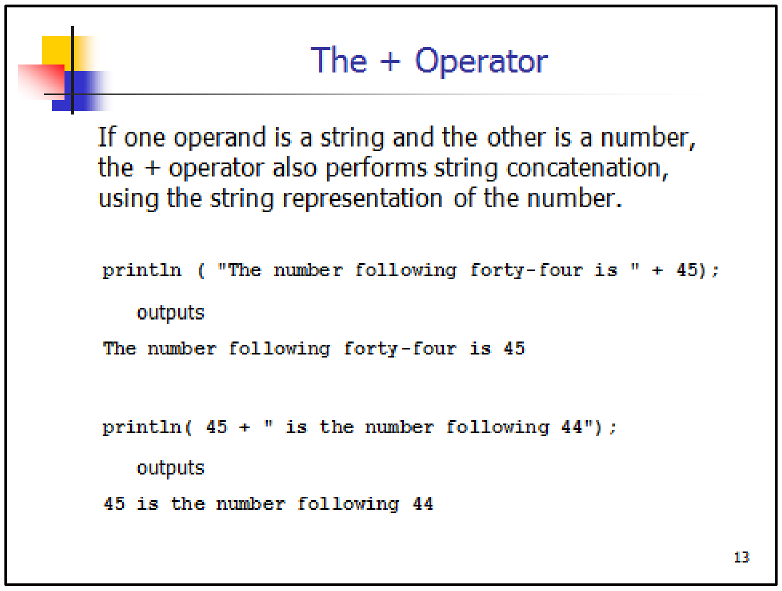
```
println( 24 + 45 )
```

outputs

```
69
```

12

(No narration)



The + Operator

If one operand is a string and the other is a number, the + operator also performs string concatenation, using the string representation of the number.

```
println ( "The number following forty-four is " + 45);
```

outputs

```
The number following forty-four is 45
```

```
println( 45 + " is the number following 44");
```

outputs

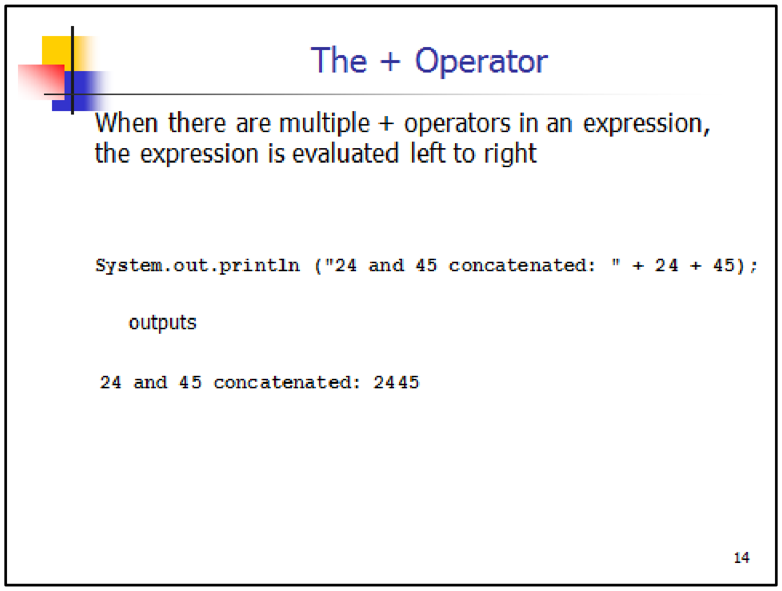
```
45 is the number following 44
```

13

If both operands are of the same type, the compiler does the obvious thing. If they are strings it concatenates them; if they are numbers it adds them.

The situation gets more interesting when one operator is a string and the other is a number. How does the compiler decide what to do?

Well, a number can always be represented as a string, but a string cannot necessarily be converted into a number. So the compiler does the thing than can't fail. It replaces the number by the string representing that number and does string concatenation



The + Operator

When there are multiple + operators in an expression, the expression is evaluated left to right

```
System.out.println ("24 and 45 concatenated: " + 24 + 45);
```

outputs

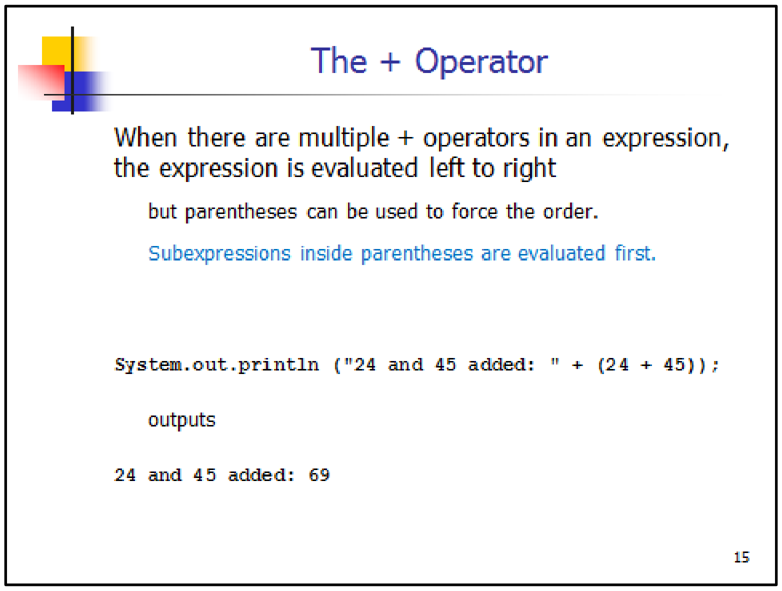
```
24 and 45 concatenated: 2445
```

The situation gets even more interesting when there are multiple operators in the same expression. The + operator takes two operands. If we include two + operators in an expression, as shown in this slide, the compiler generates code to perform one of the operations first, using the operands immediately surrounding the +. Then it uses the result of that operation as an operand for the other operator. It does this from left to right within the expression.

In the example shown in this slide, the compiler will first do the operation "24 and 45 concatenated: " + 24. Since this is a string and a number, it will convert the number 24 to the string "24" and then do string concatenation. This produces the string

"24 and 45 concatenated: 24"

Then it concatenates this string with the number 45. Again the + operator has a string and a number. So it converts the number 45 to the string "45" and concatenates the strings, yielding "24 and 45 concatenated: 2445".



The + Operator

When there are multiple + operators in an expression, the expression is evaluated left to right

but parentheses can be used to force the order.

Subexpressions inside parentheses are evaluated first.

```
System.out.println ("24 and 45 added: " + (24 + 45));
```

outputs

```
24 and 45 added: 69
```

15

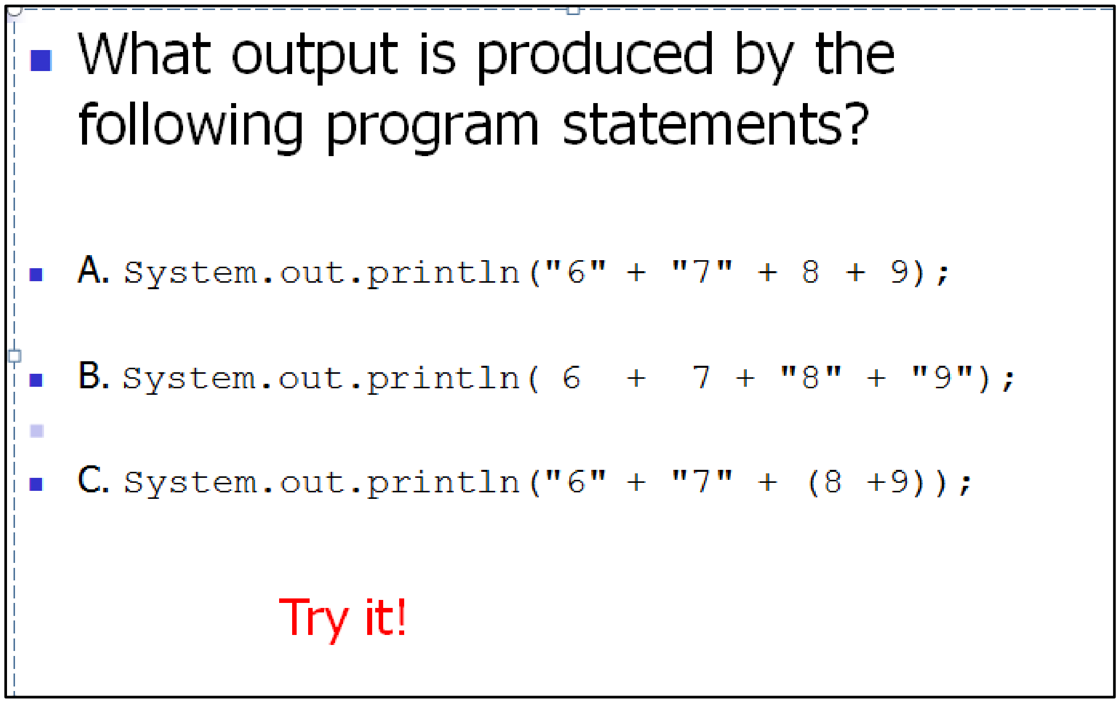
Sometimes the normal left to right evaluation doesn't do what we want. We can always force the compiler to evaluate the expression in a different order by using parentheses.

Subexpressions inside parentheses are always evaluated first.

In the example shown in this slide, the number 24 and the number 45 are inside parentheses. Because they are in parentheses, the compiler performs that operation first. Since both operands are numbers, the compiler performs addition, yielding the number 69.

Then the compiler concatenates the string "24 and 45 added: " and the number 69. Since this is a string and a number, it converts the number 69 to the string "69" and does string concatenation. The result, as shown in the slide, is the string

24 and 45 added: 69



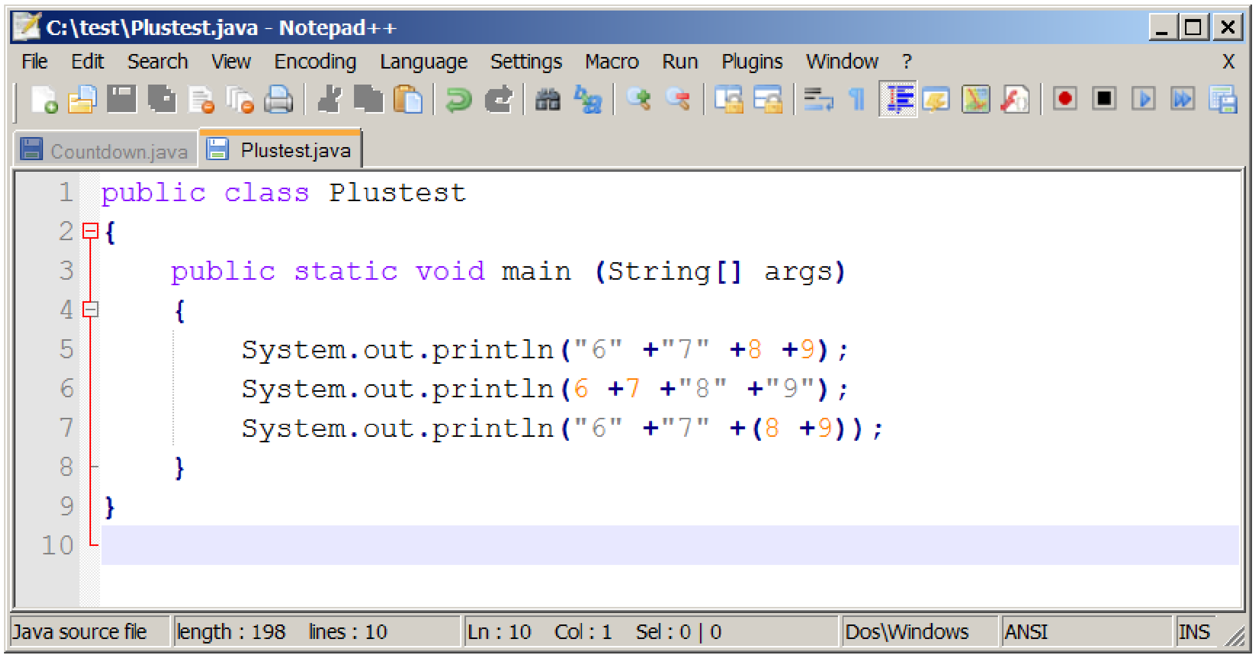
- What output is produced by the following program statements?

- A. `System.out.println("6" + "7" + 8 + 9);`
- B. `System.out.println( 6  +  7 + "8" + "9");`
- C. `System.out.println("6" + "7" + (8 +9));`

Try it!

As a test of your understanding, try to say what the output will be for each of these printlns.

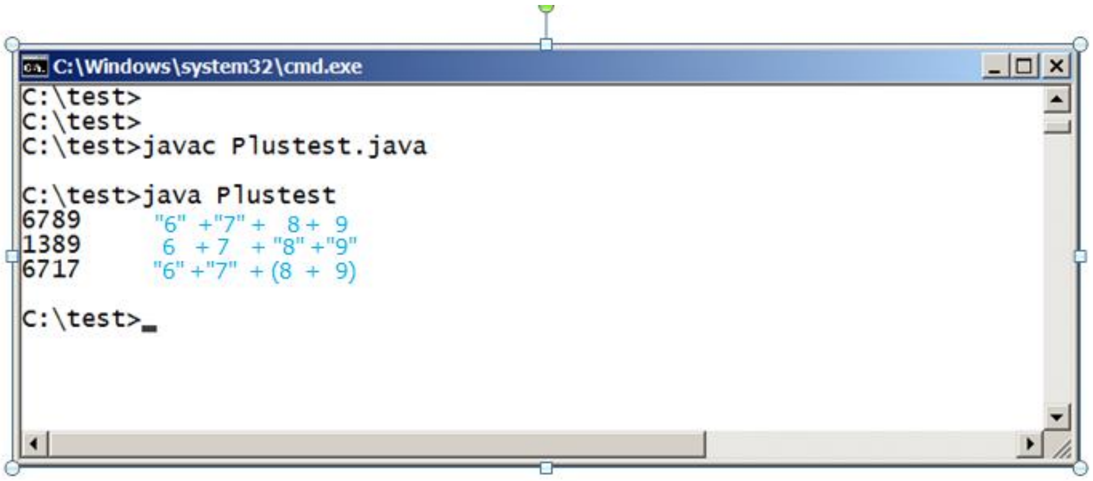
The next slides show the result of actually executing these statements in a Java program.


```
C:\test\Plustest.java - Notepad++
```

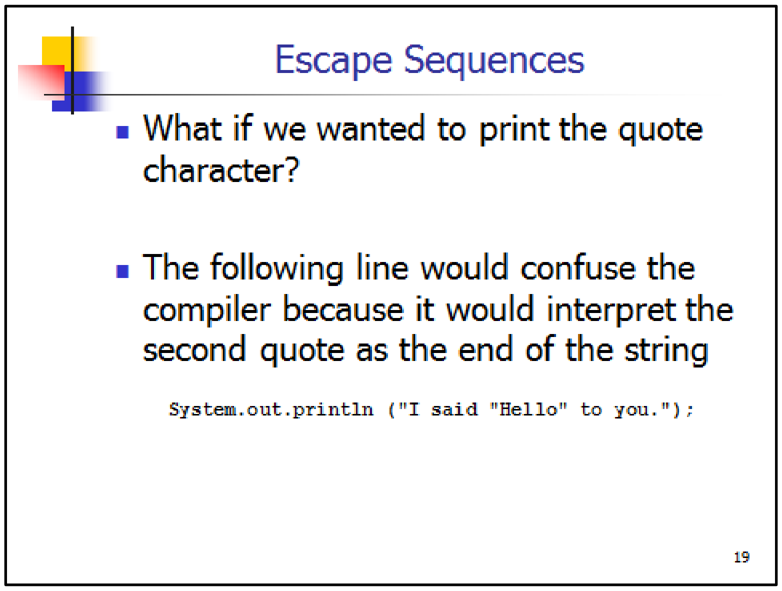
```java
public class Plustest
{
    public static void main (String[] args)
    {
        System.out.println("6" +"7" +8 +9);
        System.out.println(6 +7 +"8" +"9");
        System.out.println("6" +"7" +(8 +9));
    }
}

```

```
Java source file   length : 198   lines : 10   Ln : 10   Col : 1   Sel : 0 | 0   Dos\Windows   ANSI   INS
```

(No narration)


```
C:\Windows\system32\cmd.exe
C:\test>
C:\test>
C:\test>javac Plustest.java

C:\test>java Plustest
6789
1389
6717

C:\test>_
```

(No narration)

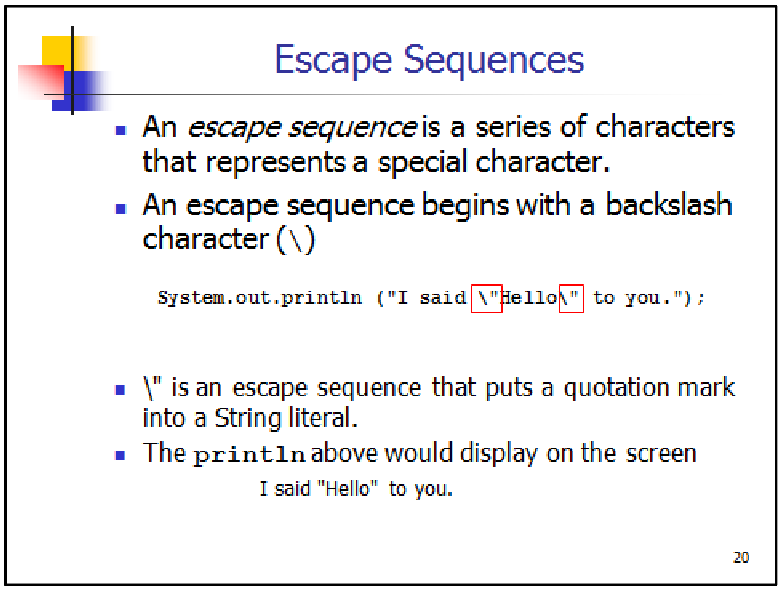


Escape Sequences

- What if we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

(No narration)



Escape Sequences

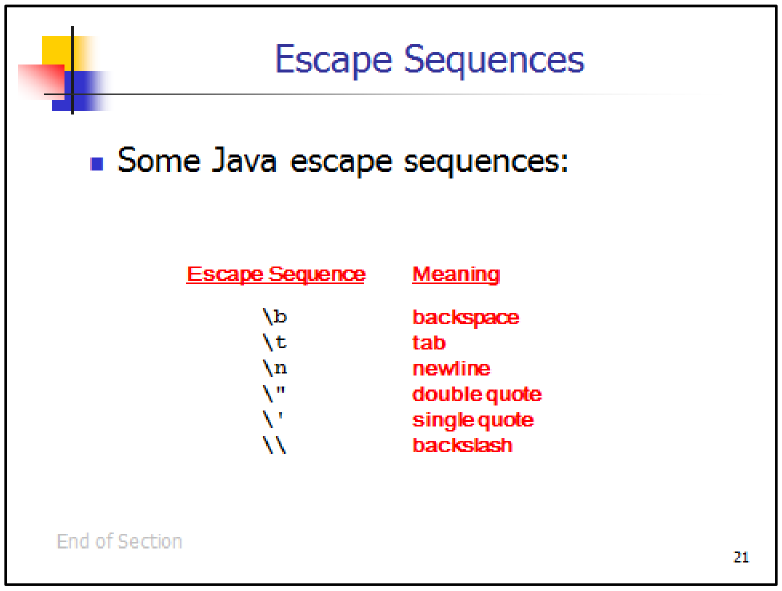
- An *escape sequence* is a series of characters that represents a special character.
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```

- \" is an escape sequence that puts a quotation mark into a String literal.
- The `println` above would display on the screen

  I said "Hello" to you.

(No narration)



Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\"`	double quote
`\'`	single quote
`\\`	backslash

End of Section

(No narration)