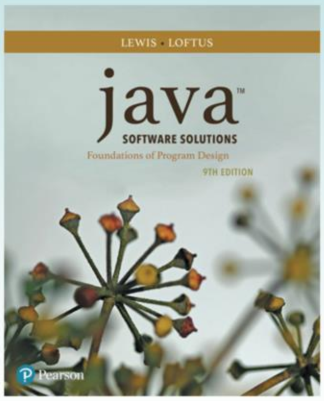
Chapter 2
Data and Expressions

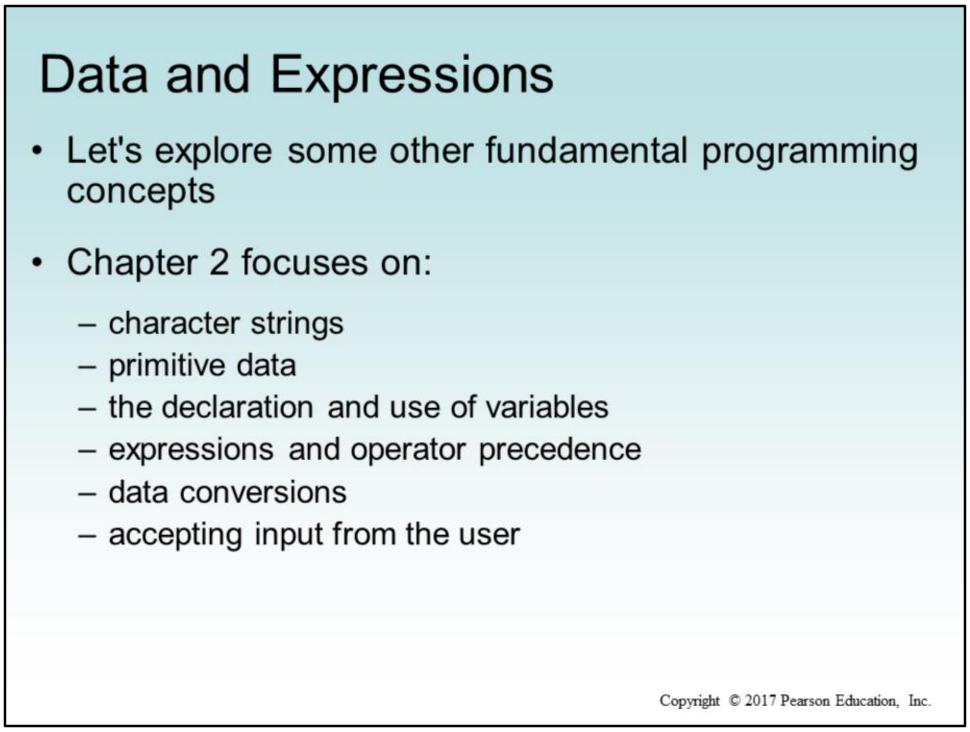


Java Software Solutions

Foundations of Program Design

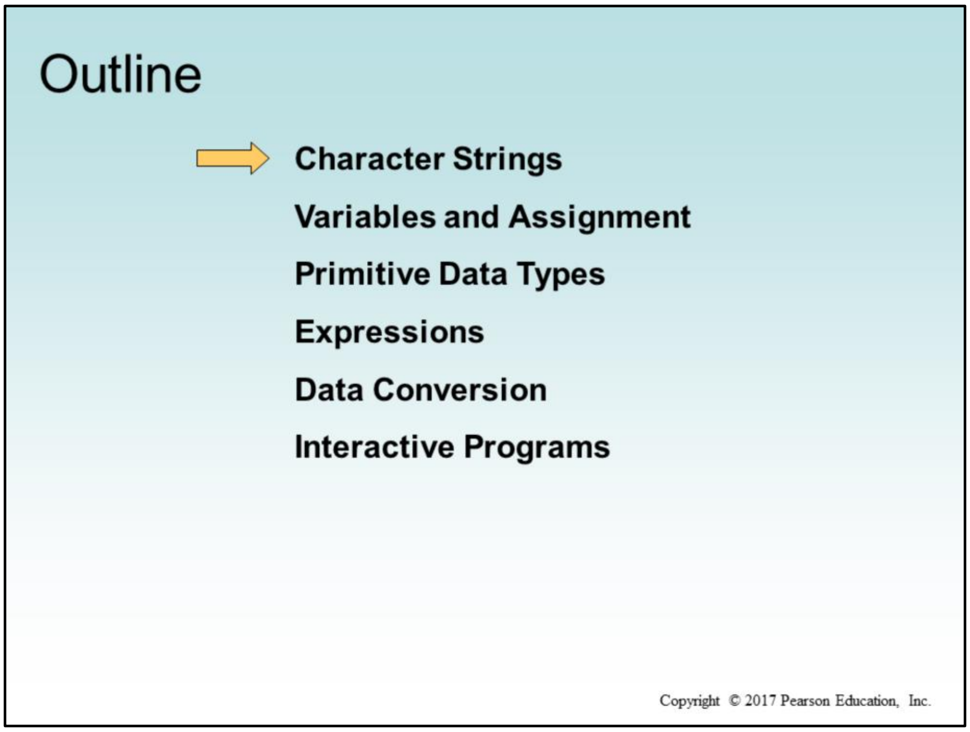
9th Edition

John Lewis
William Loftus

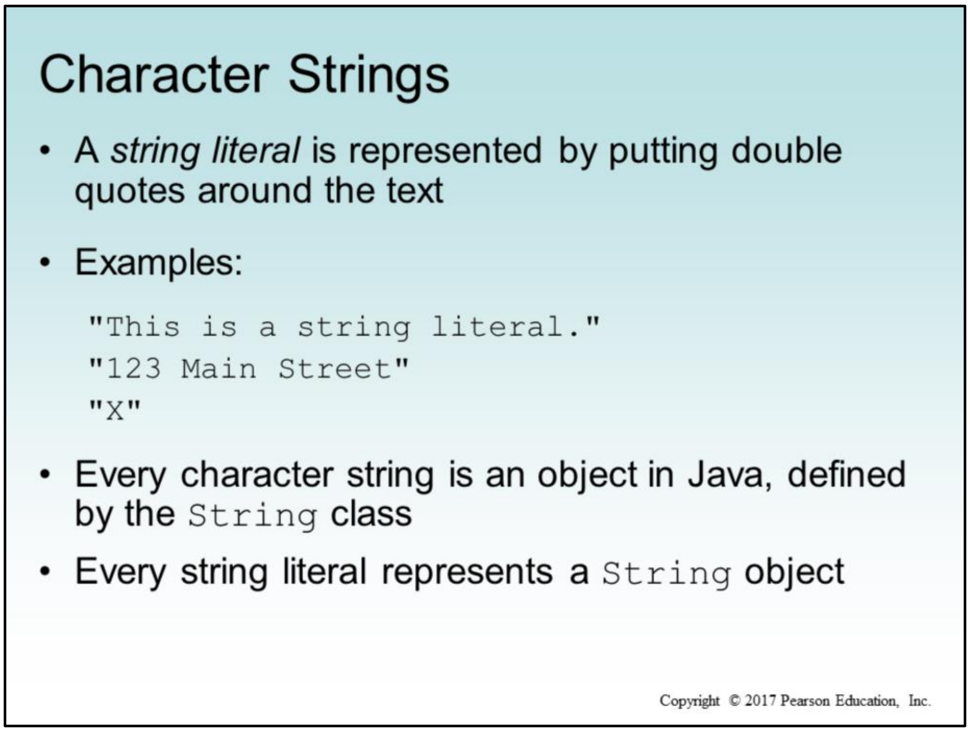


Data and Expressions

- Let's explore some other fundamental programming concepts
- Chapter 2 focuses on:
  - character strings
  - primitive data
  - the declaration and use of variables
  - expressions and operator precedence
  - data conversions
  - accepting input from the user



Outline

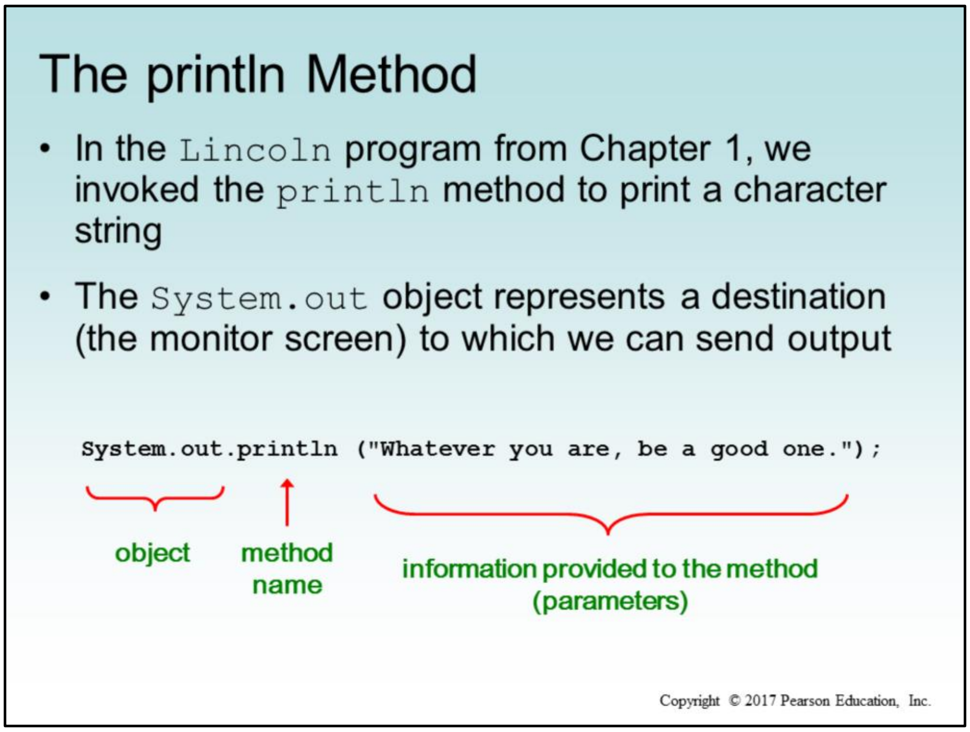


Character Strings

- A *string literal* is represented by putting double quotes around the text
- Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

- Every character string is an object in Java, defined by the `String` class
- Every string literal represents a `String` object

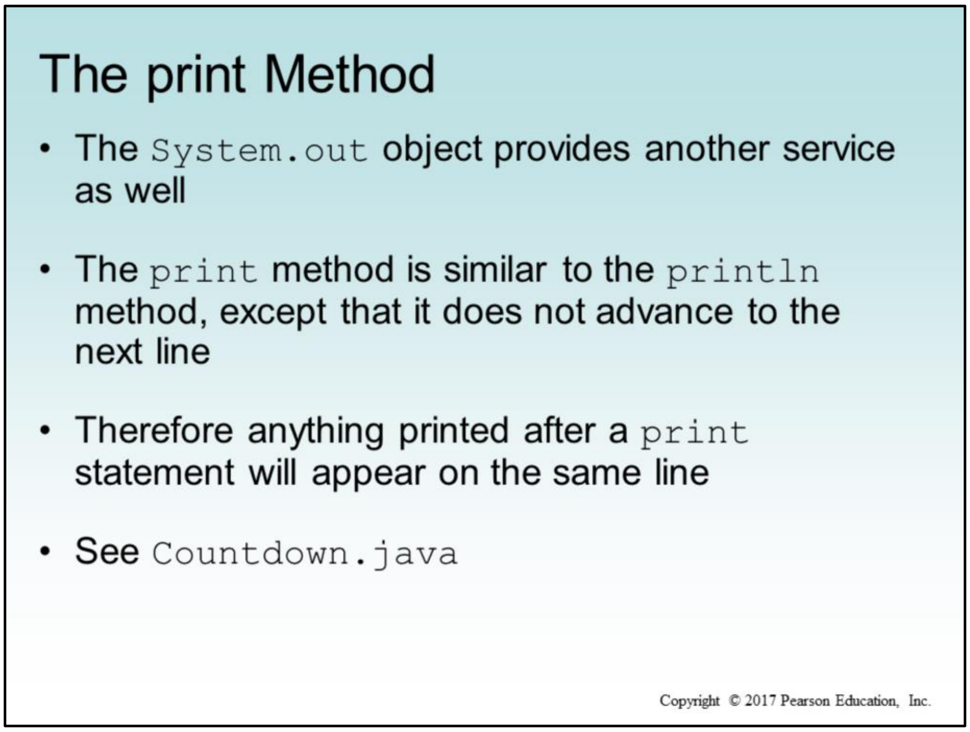


The println Method

- In the `Lincoln` program from Chapter 1, we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

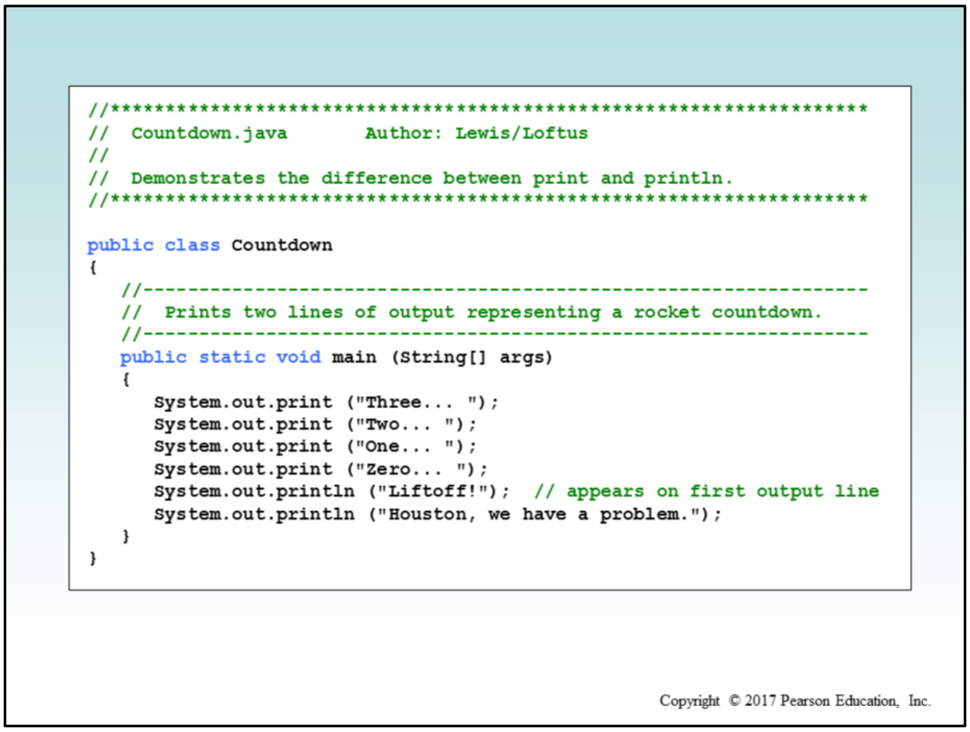
```
System.out.println ("Whatever you are, be a good one.");
```

$$\underbrace{\texttt{System.out}}_{\text{object}}.\underset{\substack{\uparrow\\ \text{method}\\ \text{name}}}{\texttt{println}}\ \underbrace{\texttt{("Whatever you are, be a good one.")}}_{\substack{\text{information provided to the method}\\ \text{(parameters)}}};$$

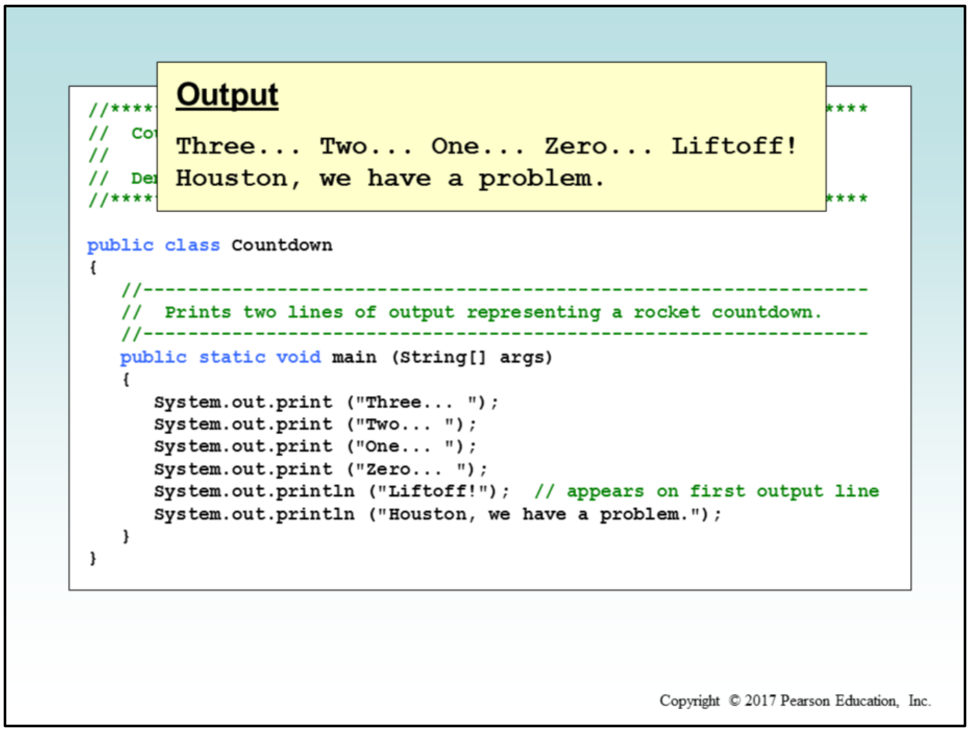


The print Method

- 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
- See `Countdown.java`



```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!");  // appears on first output line
      System.out.println ("Houston, we have a problem.");
   }
}
```



Output

```
Three... Two... One... Zero... Liftoff!
Houston, we have a problem.
```

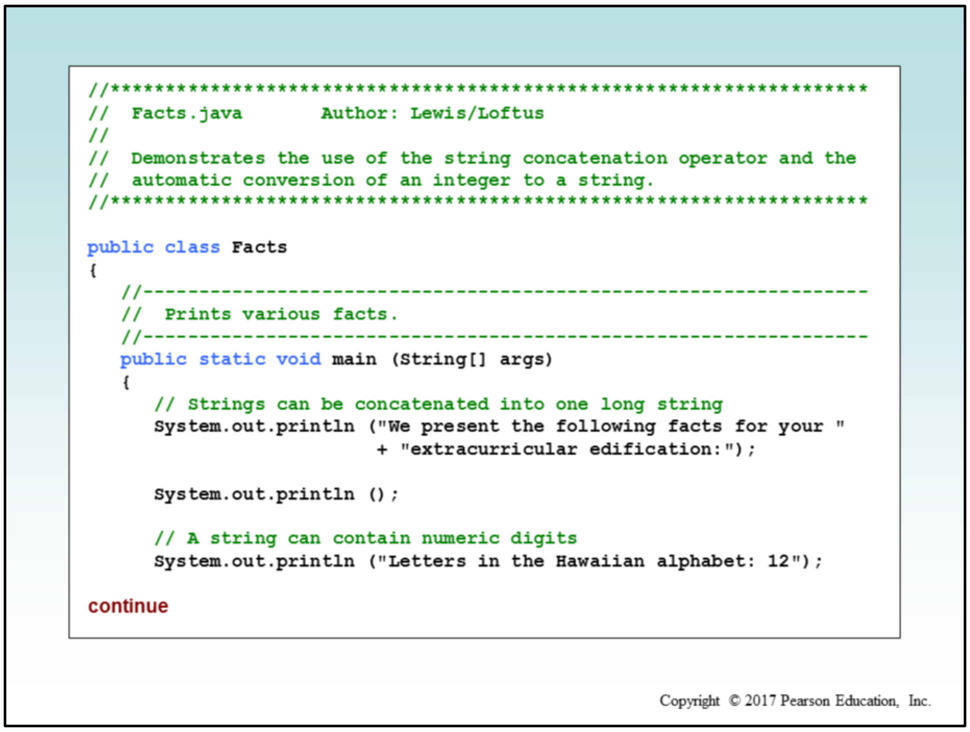
```
//****                                                            ****
//  Co
//
//  De
//****                                                            ****

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!");  // appears on first output line
      System.out.println ("Houston, we have a problem.");
   }
}
```

String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another

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

- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program
- See `Facts.java`



```
//********************************************************************
//  Facts.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the string concatenation operator and the
//  automatic conversion of an integer to a string.
//********************************************************************

public class Facts
{
   //-----------------------------------------------------------------
   //  Prints various facts.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      // Strings can be concatenated into one long string
      System.out.println ("We present the following facts for your "
                          + "extracurricular edification:");

      System.out.println ();

      // A string can contain numeric digits
      System.out.println ("Letters in the Hawaiian alphabet: 12");
```

continue

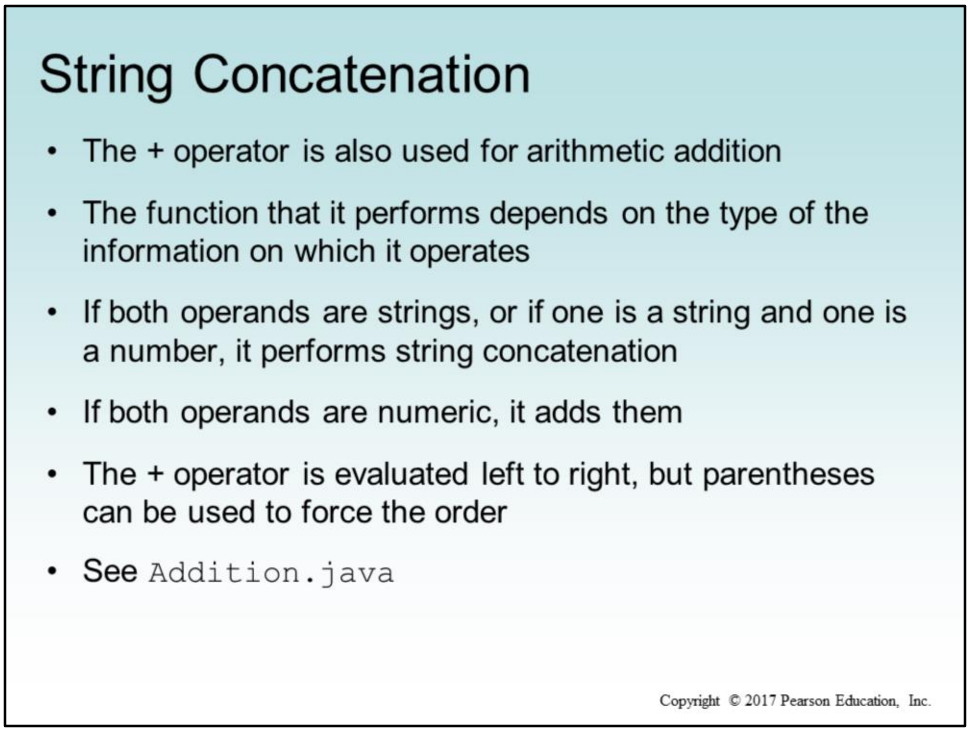
continue

```
      // A numeric value can be concatenated to a string
      System.out.println ("Dialing code for Antarctica: " + 672);

      System.out.println ("Year in which Leonardo da Vinci invented "
                          + "the parachute: " + 1515);

      System.out.println ("Speed of ketchup: " + 40 + " km per year");
   }
}
```

Output

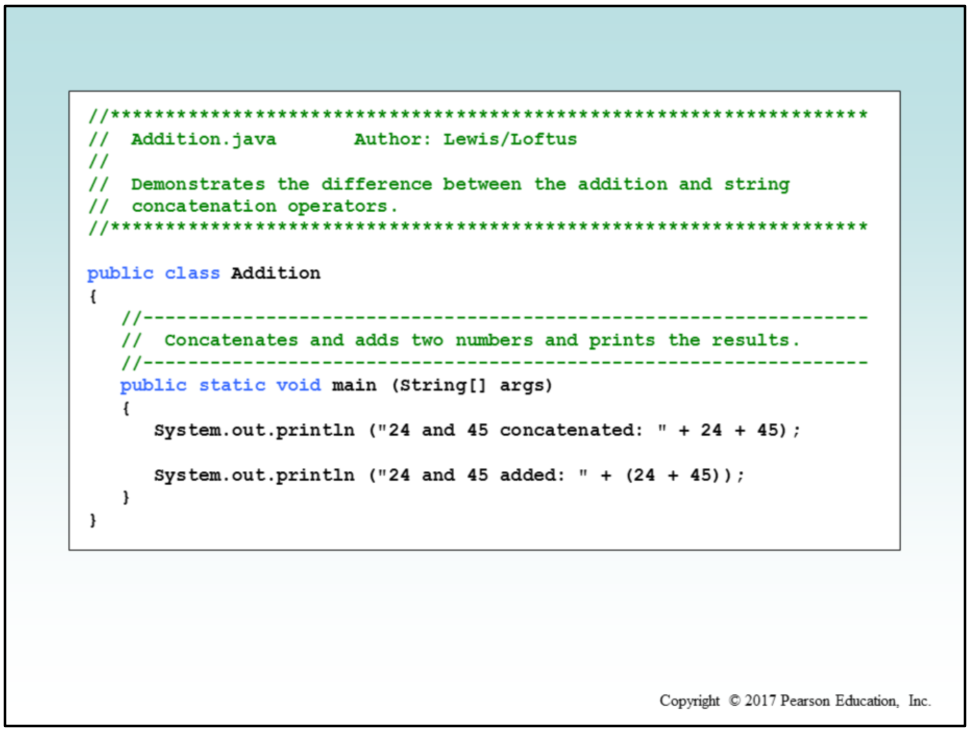
```
We present the following facts for your extracurricular edification:

Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

```
      System.out.println ("Speed of ketchup: " + 40 + " km per year");
   }
}
```



String Concatenation

- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right, but parentheses can be used to force the order
- See `Addition.java`



```
//********************************************************************
//  Addition.java       Author: Lewis/Loftus
//
//  Demonstrates the difference between the addition and string
//  concatenation operators.
//********************************************************************

public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("24 and 45 concatenated: " + 24 + 45);

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

```
//***********                                              *************
//  Addition.
//
//  Demonstra                                              string
//  concatena
//********************************************************************

public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("24 and 45 concatenated: " + 24 + 45);

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

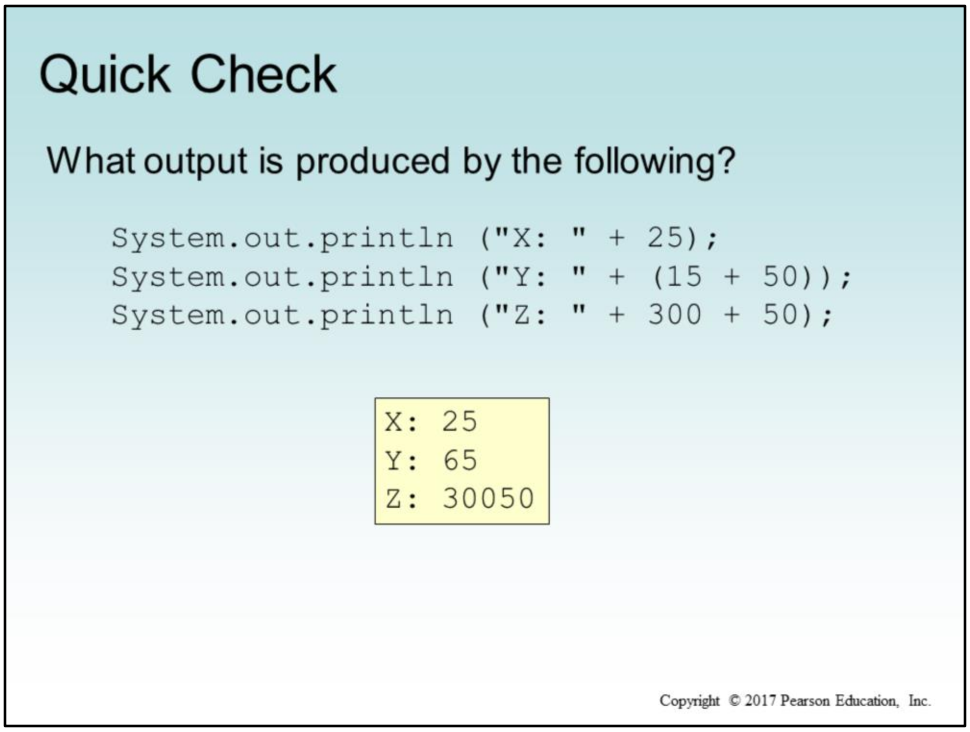
Output

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

Quick Check

What output is produced by the following?

```
System.out.println ("X: " + 25);
System.out.println ("Y: " + (15 + 50));
System.out.println ("Z: " + 300 + 50);
```

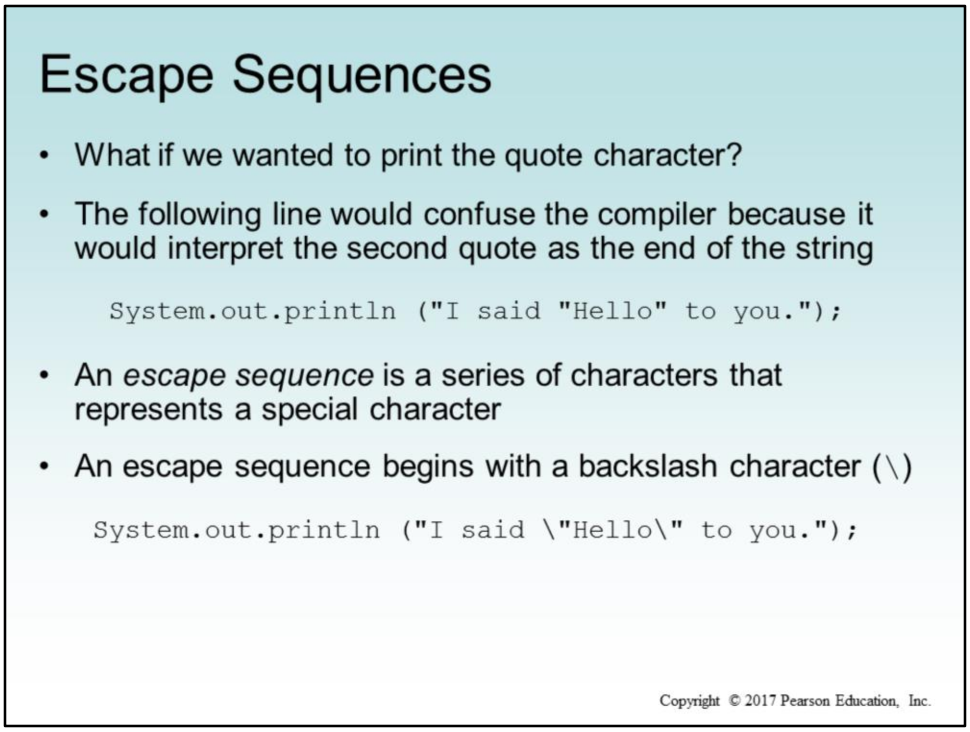


Quick Check

What output is produced by the following?

```
System.out.println ("X: " + 25);
System.out.println ("Y: " + (15 + 50));
System.out.println ("Z: " + 300 + 50);
```

```
X: 25
Y: 65
Z: 30050
```



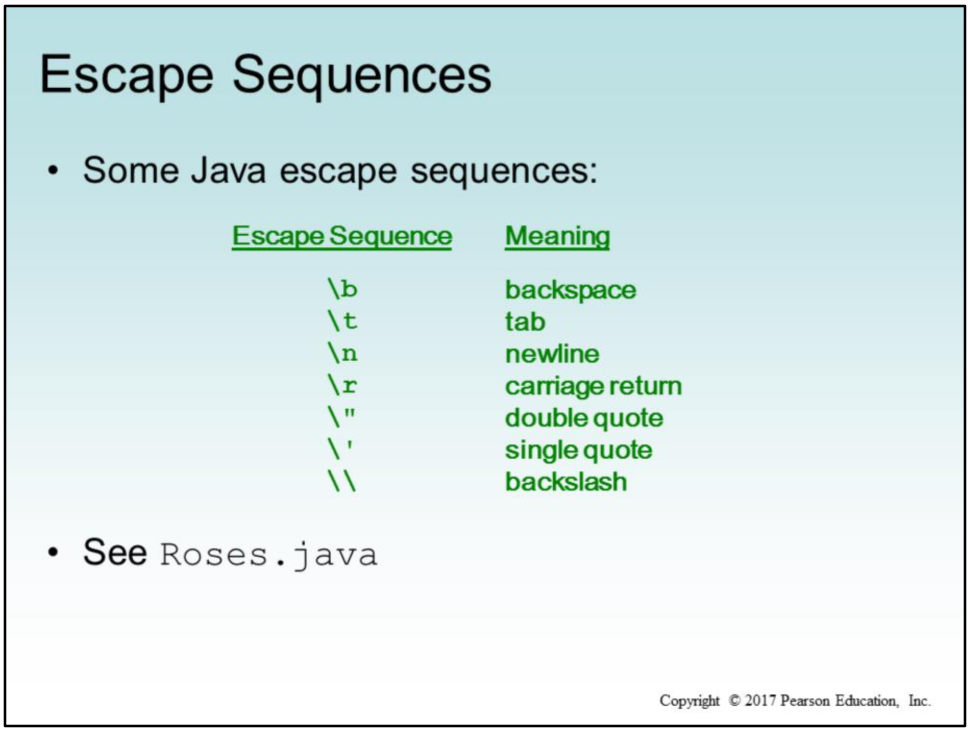
Escape Sequences

- 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.");
```

- 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.");
```



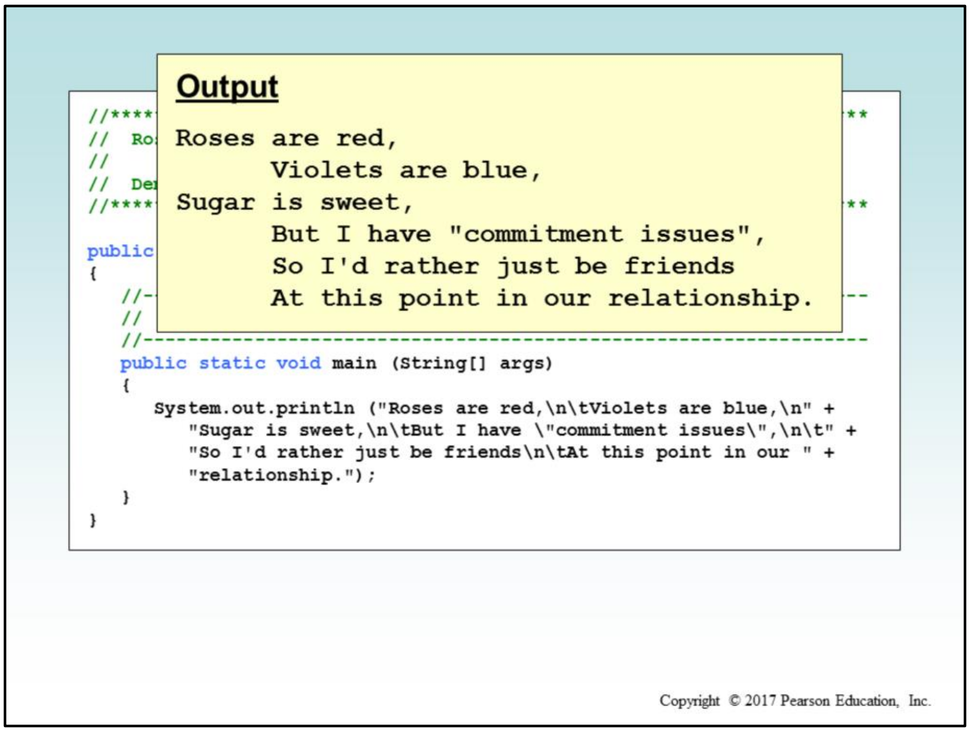
Escape Sequences

- Some Java escape sequences:

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

- See `Roses.java`

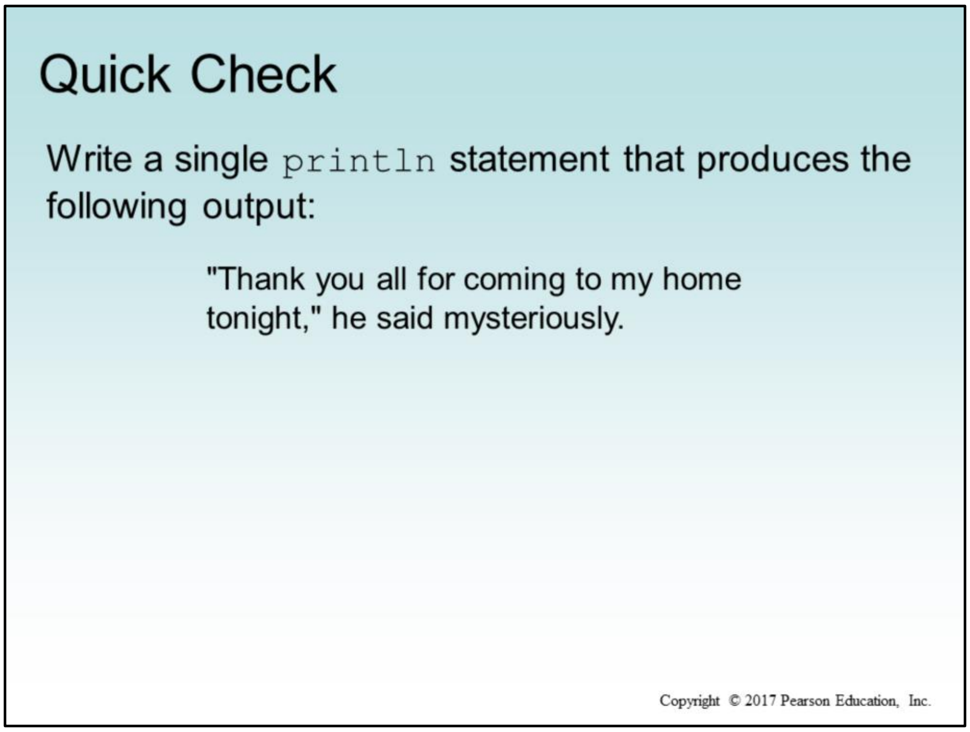
```java
//*********************************************************************
//  Roses.java       Author: Lewis/Loftus
//
//  Demonstrates the use of escape sequences.
//*********************************************************************

public class Roses
{
   //-----------------------------------------------------------------
   //  Prints a poem (of sorts) on multiple lines.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```



Output

```
Roses are red,
      Violets are blue,
Sugar is sweet,
      But I have "commitment issues",
      So I'd rather just be friends
      At this point in our relationship.
```

```
//****                                                    **
//  Ro
//
//  De
//****                                                    **

public
{
   //-                                                   ---
   //
   //-----------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```



Quick Check

Write a single `println` statement that produces the following output:

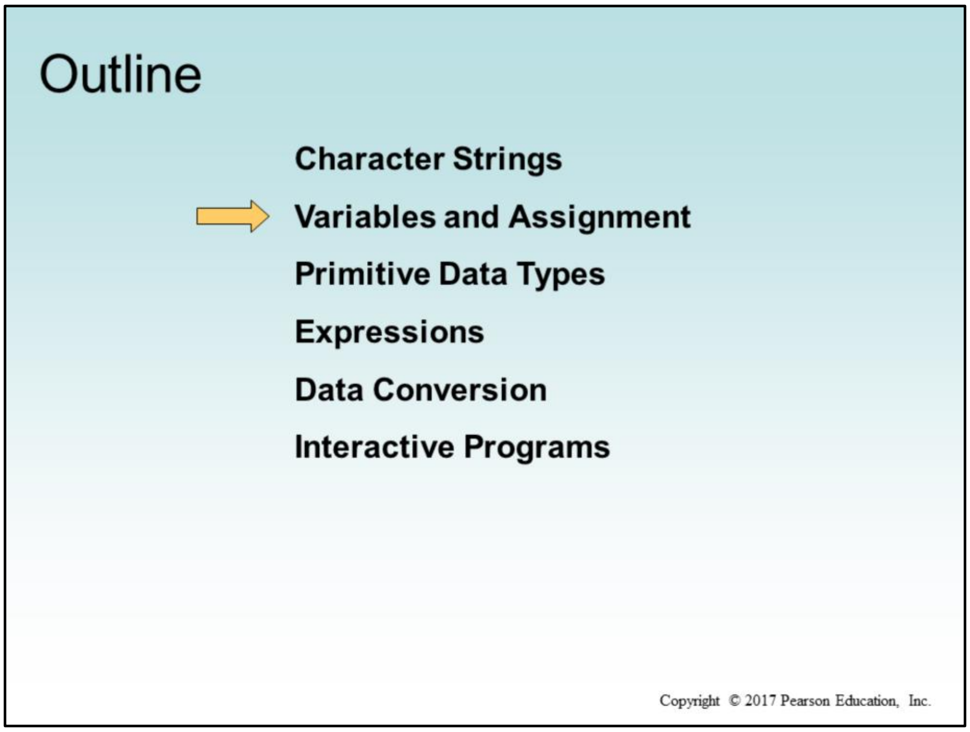
```
"Thank you all for coming to my home
tonight," he said mysteriously.
```

Quick Check

Write a single `println` statement that produces the following output:

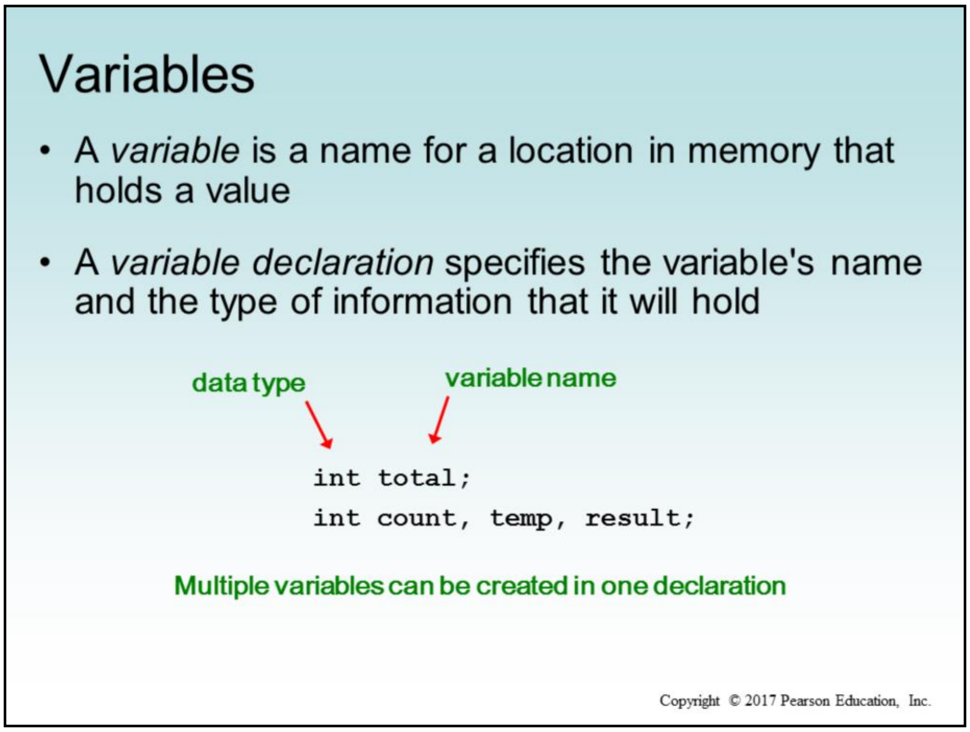
```
"Thank you all for coming to my home
tonight," he said mysteriously.
```

```
System.out.println ("\"Thank you all for " +
   "coming to my home\ntonight,\" he said " +
   "mysteriously.");
```



Outline

- **Character Strings**
- ➡ **Variables and Assignment**
- **Primitive Data Types**
- **Expressions**
- **Data Conversion**
- **Interactive Programs**



Variables

- A *variable* is a name for a location in memory that holds a value
- A *variable declaration* specifies the variable's name and the type of information that it will hold

data type
variable name

```
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration

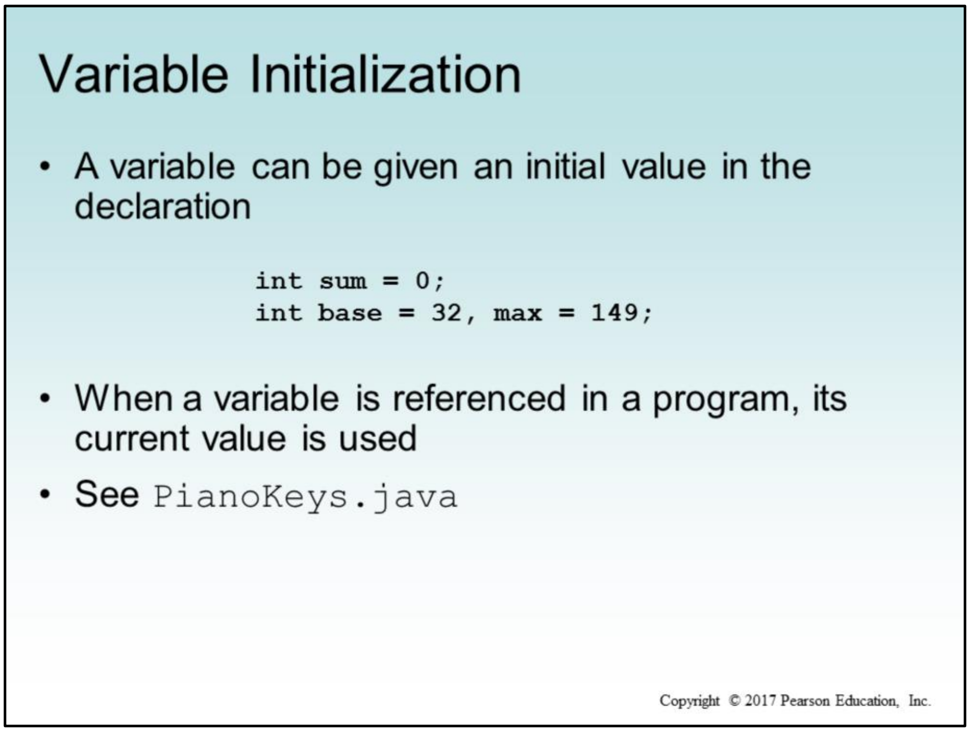
-Recall we discussed that every area in memory has an address

-Instead of accessing memory using this address, we can use a variable

-When we declare a variable, memory at a specific address is reserved for our use

-Another term for reserving memory is **allocation** (e.g. memory is allocated)

-We use the variable name to access the information stored at this reserved memory location

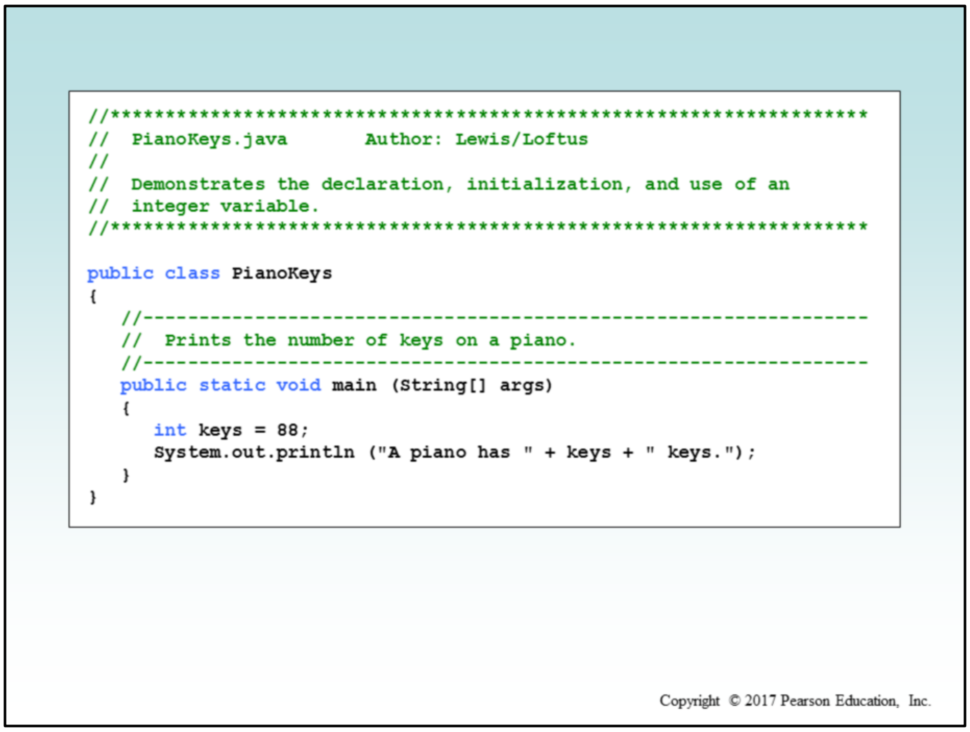


Variable Initialization

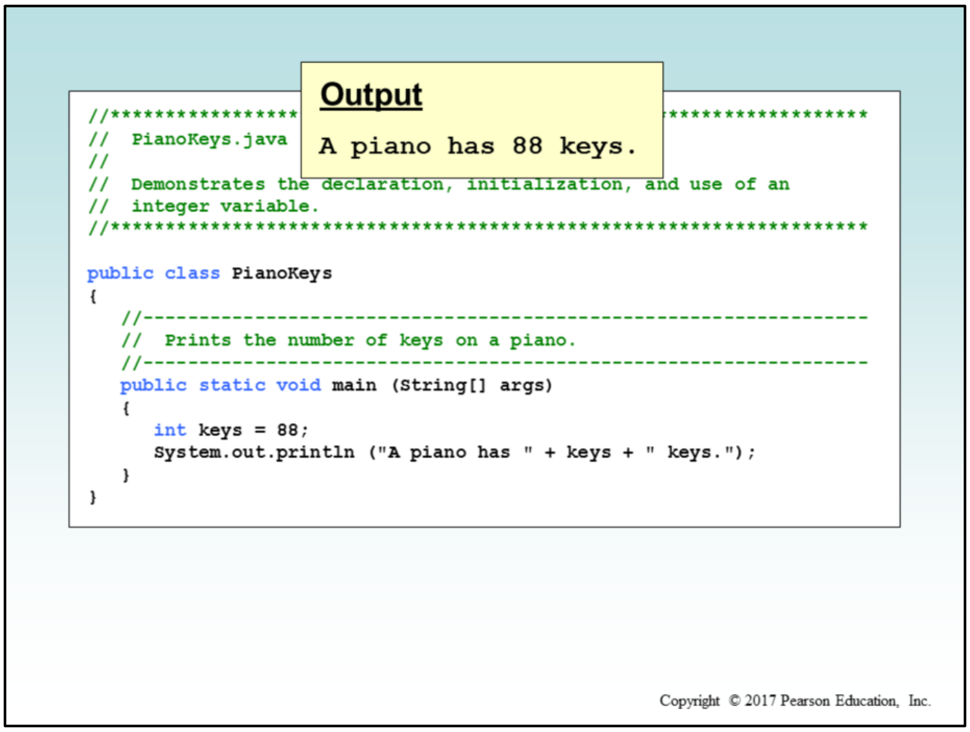
- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- See `PianoKeys.java`



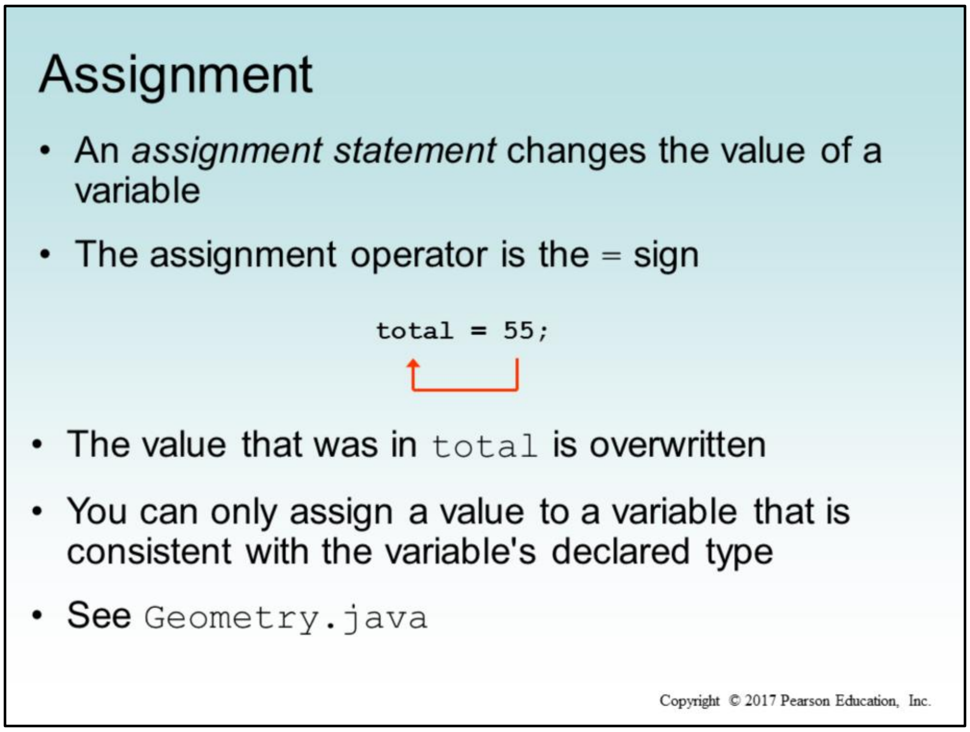
```
//********************************************************************
//  PianoKeys.java       Author: Lewis/Loftus
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable.
//********************************************************************

public class PianoKeys
{
   //-----------------------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int keys = 88;
      System.out.println ("A piano has " + keys + " keys.");
   }
}
```



Output

```
A piano has 88 keys.
```

```java
//*****************                            *******************
//  PianoKeys.java
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable.
//********************************************************************

public class PianoKeys
{
   //-----------------------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int keys = 88;
      System.out.println ("A piano has " + keys + " keys.");
   }
}
```

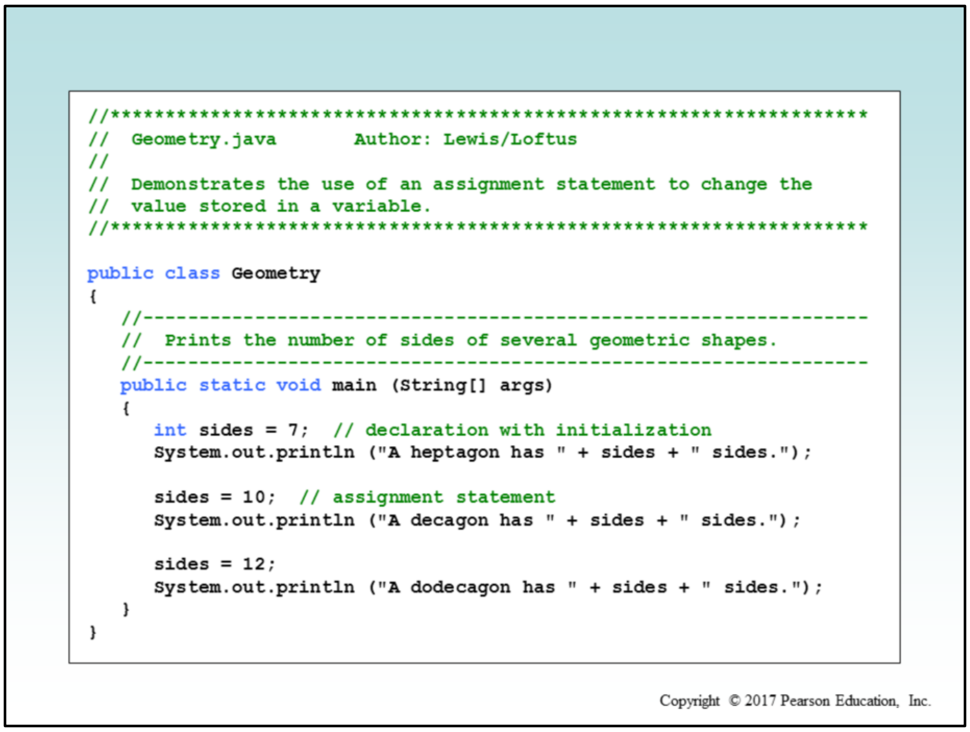


Assignment

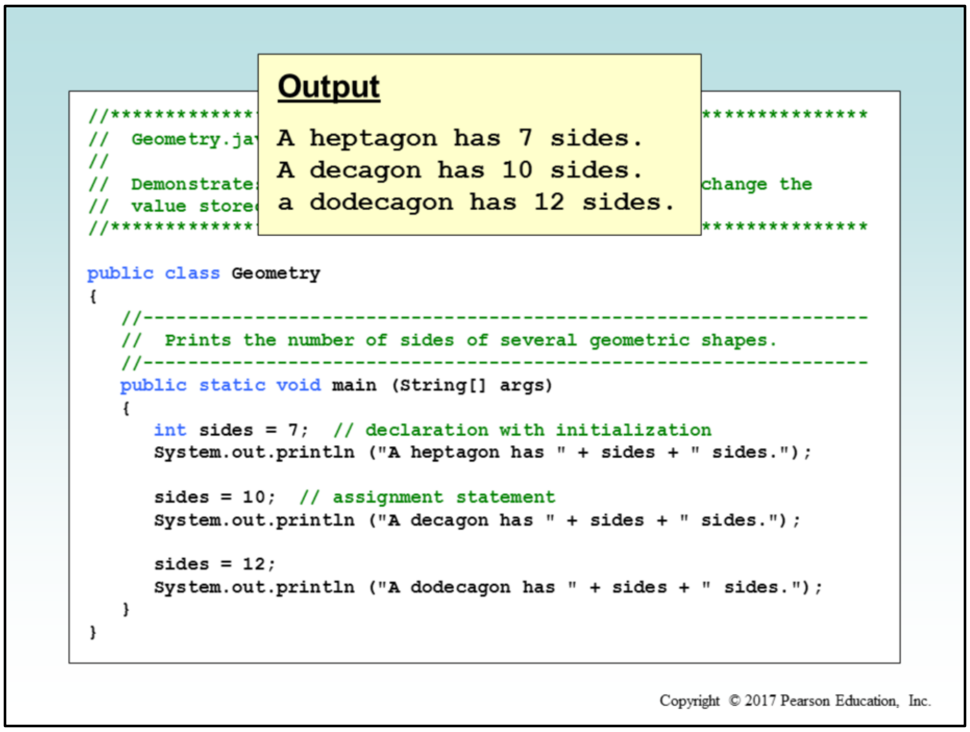
- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The value that was in `total` is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type
- See `Geometry.java`



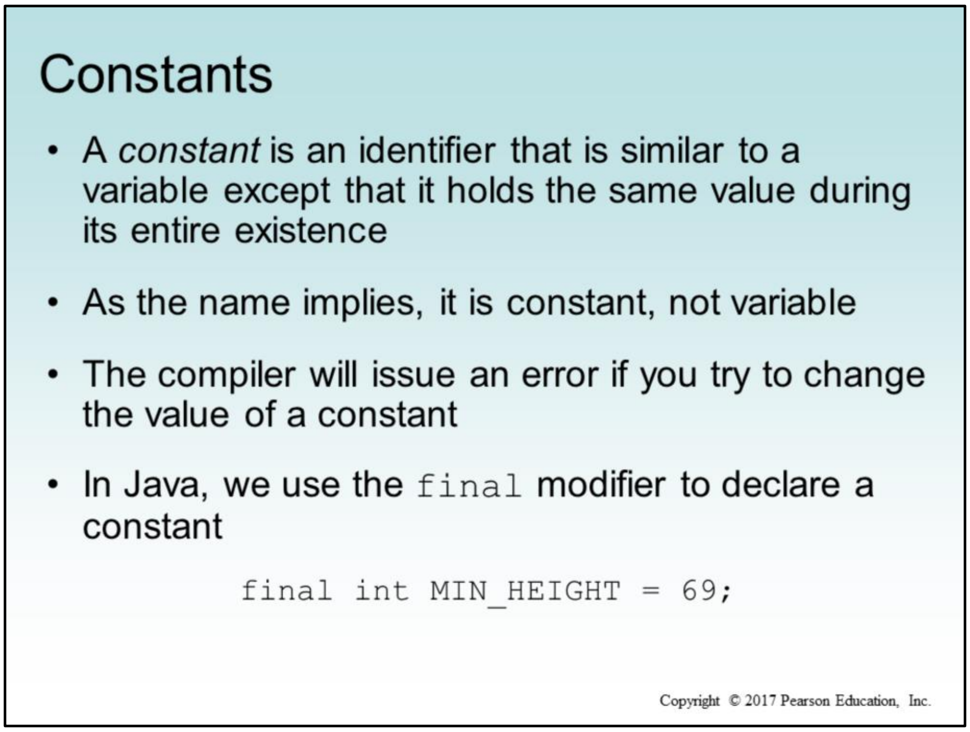
```java
//********************************************************************
//  Geometry.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an assignment statement to change the
//  value stored in a variable.
//********************************************************************

public class Geometry
{
   //-----------------------------------------------------------------
   //  Prints the number of sides of several geometric shapes.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sides = 7;  // declaration with initialization
      System.out.println ("A heptagon has " + sides + " sides.");

      sides = 10;  // assignment statement
      System.out.println ("A decagon has " + sides + " sides.");

      sides = 12;
      System.out.println ("A dodecagon has " + sides + " sides.");
   }
}
```



Output

```
A heptagon has 7 sides.
A decagon has 10 sides.
a dodecagon has 12 sides.
```

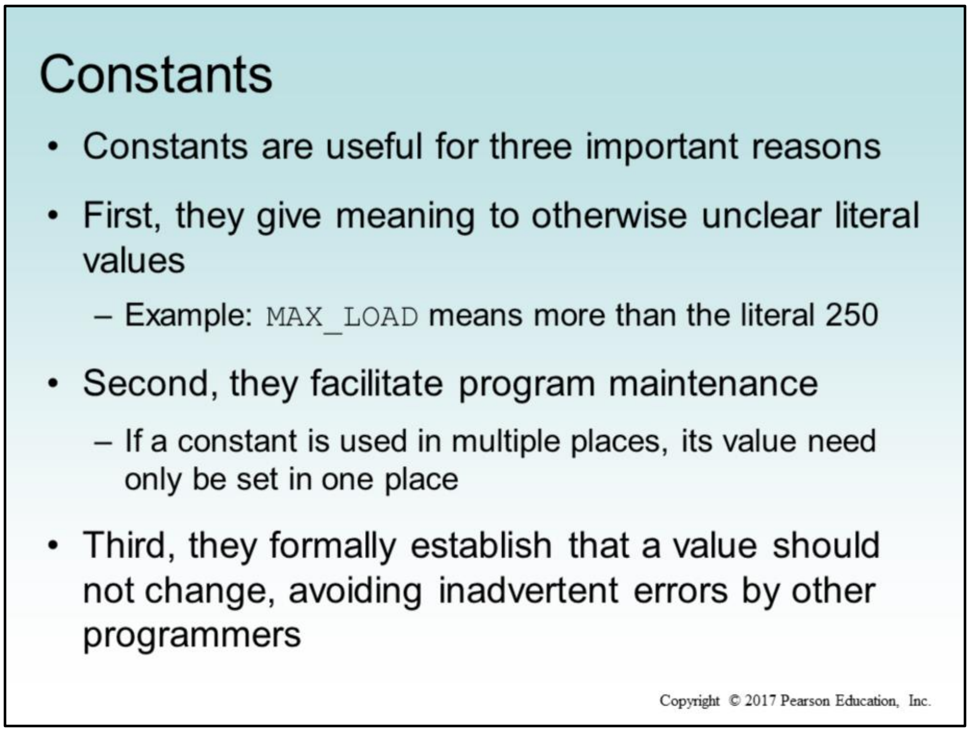
```
//*************                                            ***************
//  Geometry.ja
//
//  Demonstrate:                                           change the
//  value store
//*************                                            ***************

public class Geometry
{
   //-----------------------------------------------------------------
   //  Prints the number of sides of several geometric shapes.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sides = 7;  // declaration with initialization
      System.out.println ("A heptagon has " + sides + " sides.");

      sides = 10;  // assignment statement
      System.out.println ("A decagon has " + sides + " sides.");

      sides = 12;
      System.out.println ("A dodecagon has " + sides + " sides.");
   }
}
```



Constants

- A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the `final` modifier to declare a constant

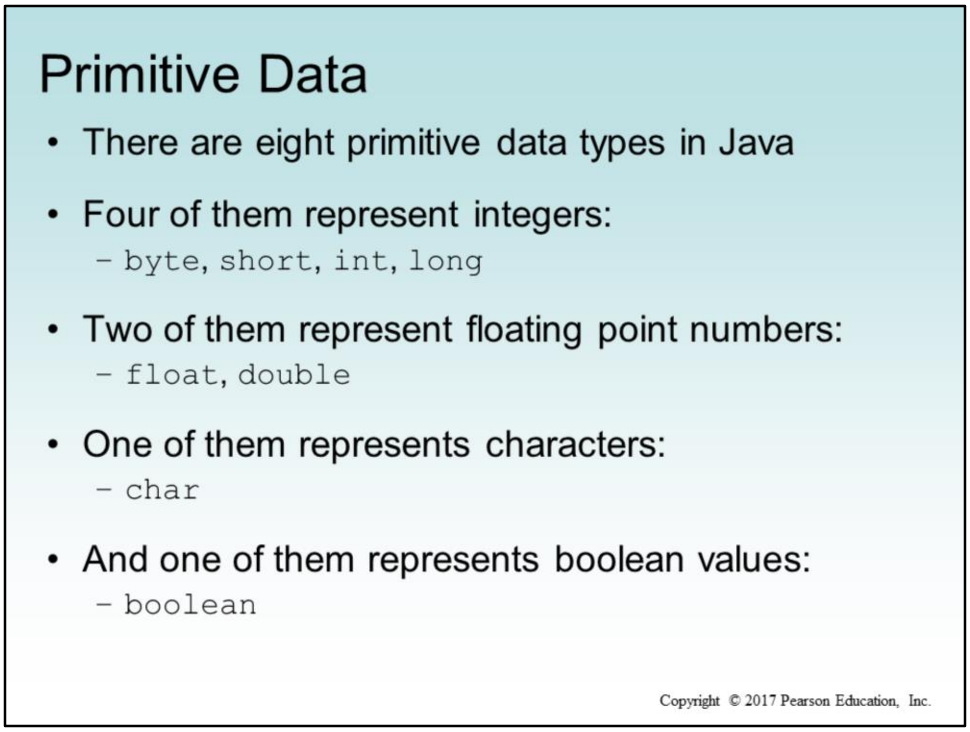
```
final int MIN_HEIGHT = 69;
```


Constants

- Constants are useful for three important reasons
- First, they give meaning to otherwise unclear literal values
  - Example: `MAX_LOAD` means more than the literal 250
- Second, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be set in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

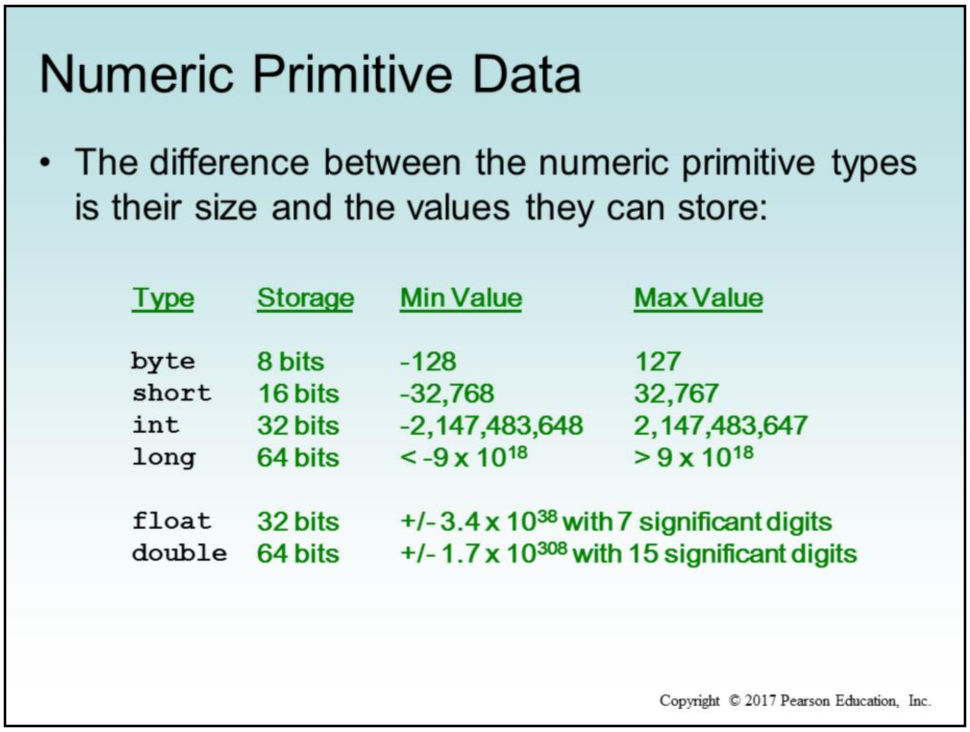
Outline

- Character Strings
- Variables and Assignment
- ➡ **Primitive Data Types**
- Expressions
- Data Conversion
- Interactive Programs



Primitive Data

- There are eight primitive data types in Java
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`



Numeric Primitive Data

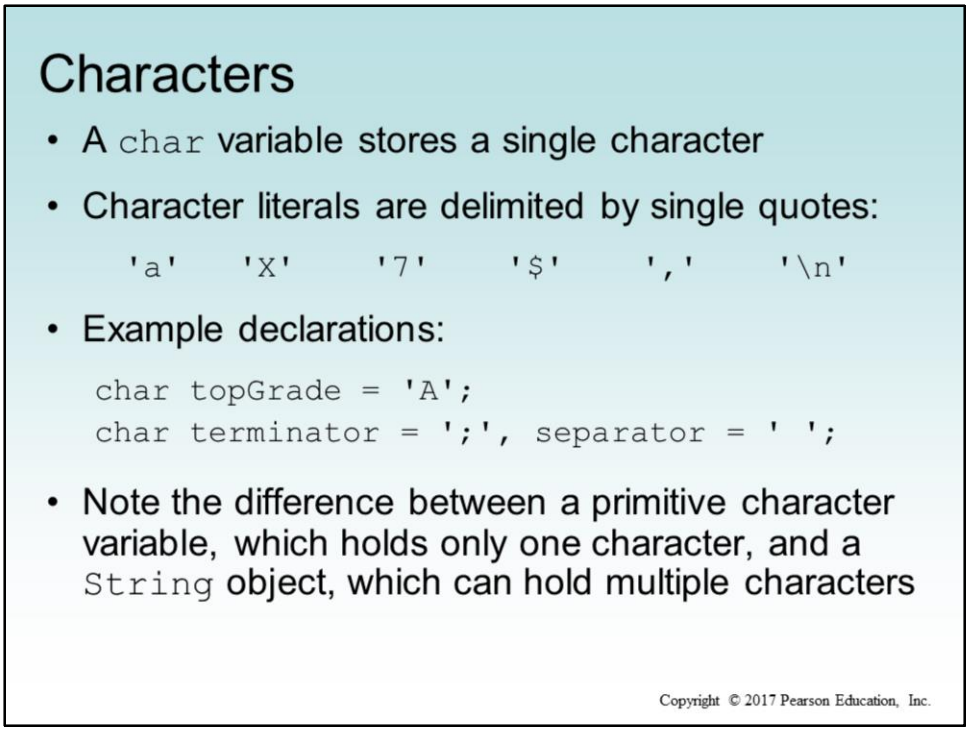
- The difference between the numeric primitive types is their size and the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

-We see here that understanding memory, bits, and bytes are critical as we write programs

-Different data types take up different amounts of memory storage

-Understanding memory is important when we declare variables of different data types



Characters

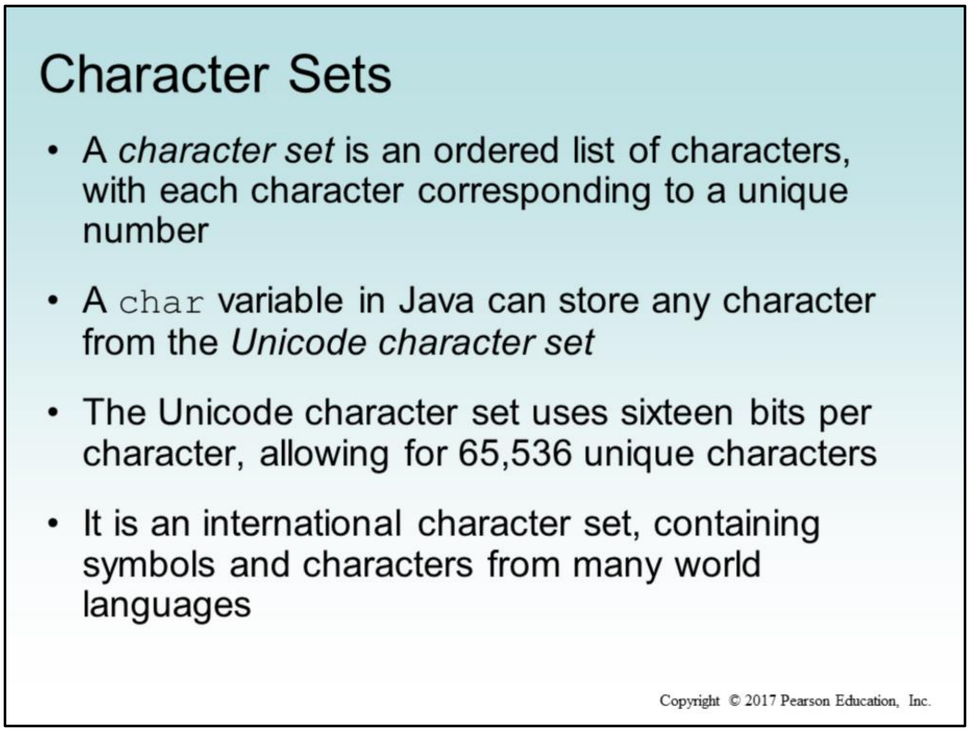
- A `char` variable stores a single character
- Character literals are delimited by single quotes:

```
'a'   'X'    '7'    '$'    ','    '\n'
```

- Example declarations:

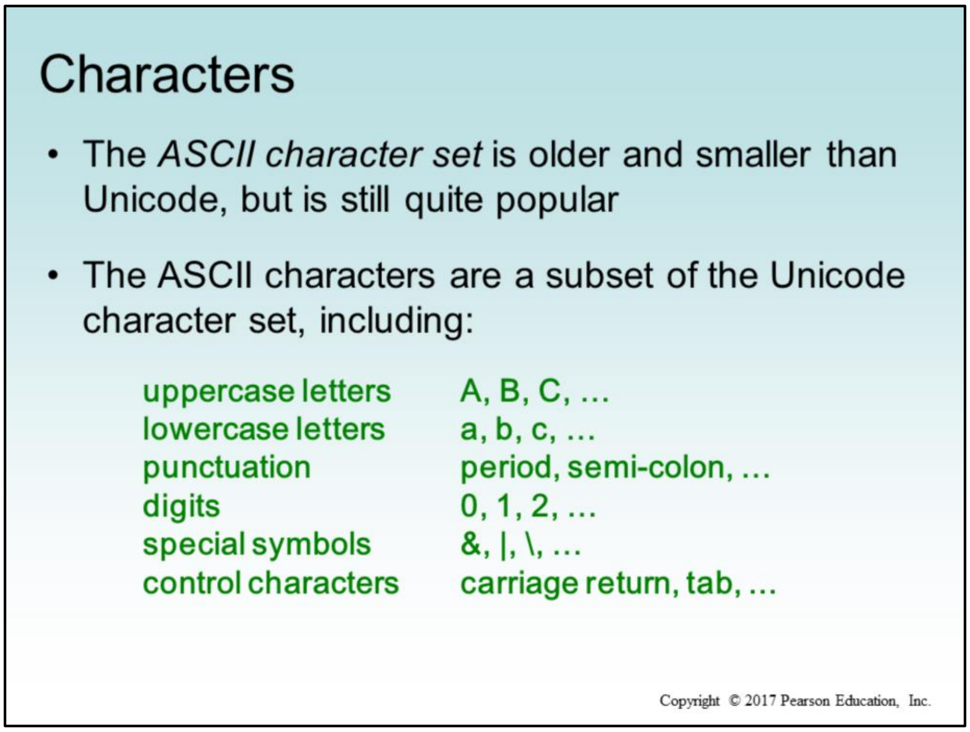
```
char topGrade = 'A';
char terminator = ';', separator = ' ';
```

- Note the difference between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters



Character Sets

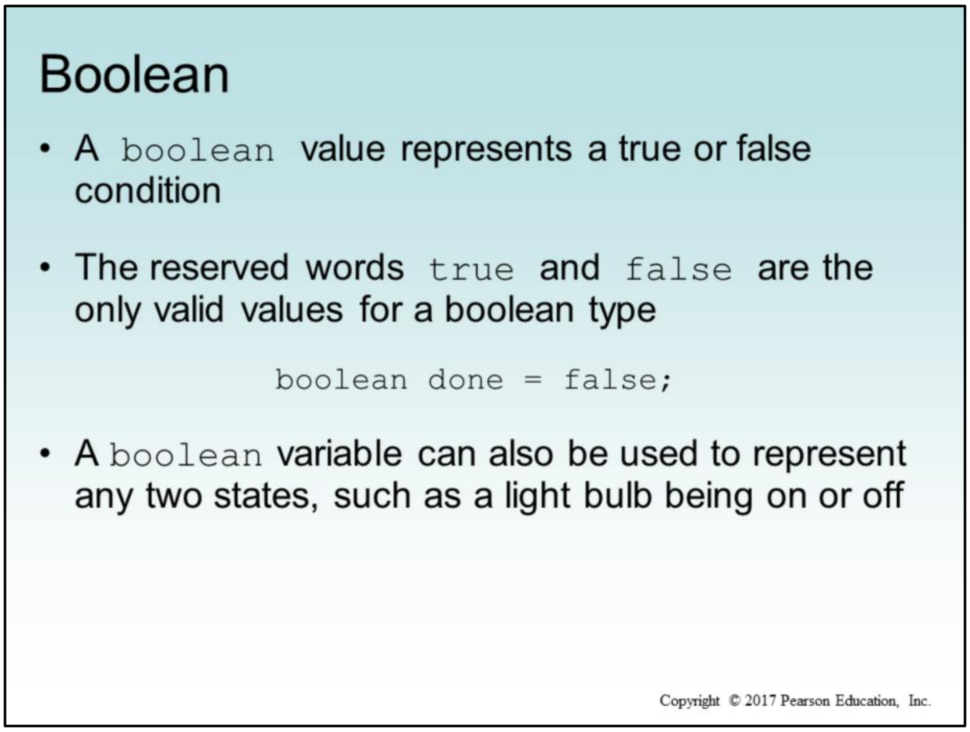
- A *character set* is an ordered list of characters, with each character corresponding to a unique number
- A `char` variable in Java can store any character from the *Unicode character set*
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages



Characters

- The *ASCII character set* is older and smaller than Unicode, but is still quite popular
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, …



Boolean

- A `boolean` value represents a true or false condition
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off