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.");
  }
}
```

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

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`

```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.
```

```
//*****************************************************************
//  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;
```

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`		-128	127
`short`	8 bits	-32,768	32,767
`int`	16 bits	-2,147,483,648	2,147,483,647
`long`	32 bits	< -9 x 1018	> 9 x 1018
	64 bits		
`float`		+/- 3.4 x 1038 with 7 significant digits	
`double`	32 bits	+/- 1.7 x 10308 with 15 significant digits	
	64 bits		

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

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

Expressions

- An *expression* is a combination of one or more operators and operands
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

- If either or both operands are floating point values, then the result is a floating point value

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3   equals
         4

8 / 12   equals
         0
```

- The remainder operator (%) returns the remainder after dividing the first operand by the second

```
14 % 3   equals
         2

8 % 12   equals
         8
```

Quick Check

What are the results of the following expressions?

```
    12 / 2
12.0 / 2.0
    10 / 4
  10 / 4.0
    4 / 10
  4.0 / 10
    12 % 3
    10 % 3
    3 % 10
```

Quick Check

What are the results of the following expressions?

```
    12 / 2  =  6
12.0 / 2.0  =  6.0
    10 / 4  =  2
  10 / 4.0  =  2.5
    4 / 10  =  0
  4.0 / 10  =  0.4
    12 % 3  =  0
    10 % 3  =  1
    3 % 10  =  0
```

Operator Precedence

- Operators can be combined into larger expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated before addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

```
count = count + 1;
```

Then the result is stored back into `count` (overwriting the original value)