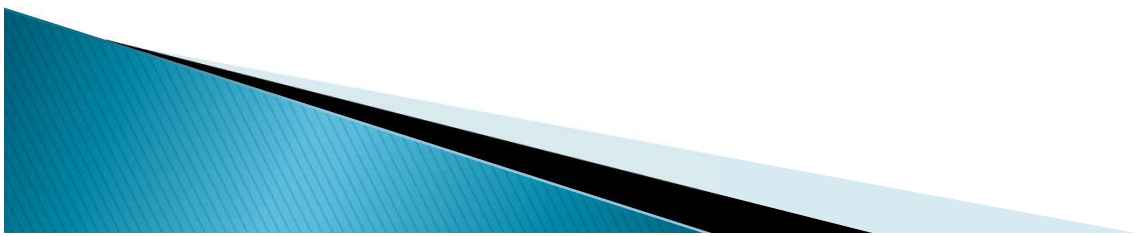
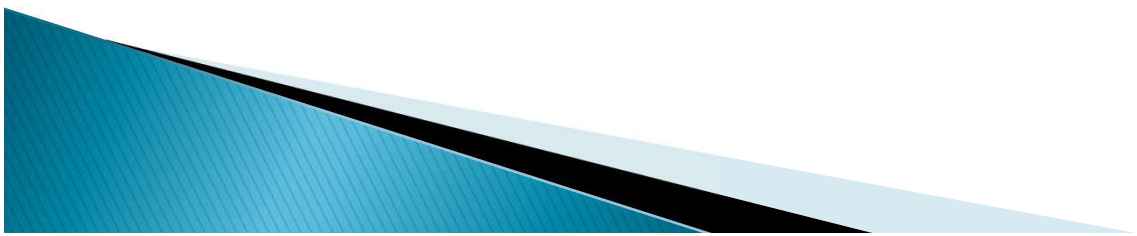
# Software

- For the course, we will be using JCreator as the IDE (Integrated Development Environment).
- We strongly advise that you install these to your own computers.
- If you do not have your own computer, the computer labs on campus have the software.



# Lab sessions

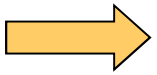
- You will have 11 lab assignments which you are required to solve individually during the lab sessions. The minimum lab grade will be discarded at the end. There will be no make-up for the lab assignment you miss.
- At the end of the lab, assistants will check and grade the assignment.
- TAs will also ask students to explain their solution in order to ensure that they really have understood the concepts involved.



# Communication

- If you have difficulty in understanding course subjects, go to the teaching assistants or ask for office hours and ask them.  Be wise and make use of the resources provided to you.

# Outline



**Object-Oriented Programming**

**The Java Programming Language**

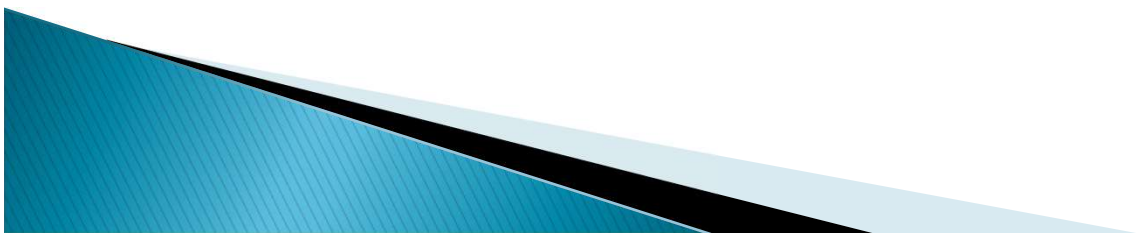
**Program Development**

# Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact

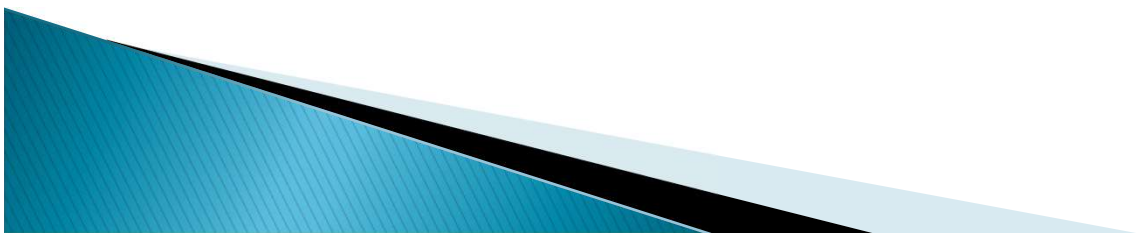
# Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- Object-oriented paradigm focuses on objects, data structures that have attributes (fields) and behaviours (methods).



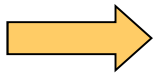
# Object-Oriented Programming

- Java is an object-oriented programming language
- As the term implies, an object is a fundamental entity in a Java program
- Objects can be used effectively to represent real-world entities
- Objects have state (data) and behaviour (methods).
- For instance, an object might represent a particular employee in a company where each employee object handles the processing and data management related to that employee.



# Outline

**Object-Oriented Programming**

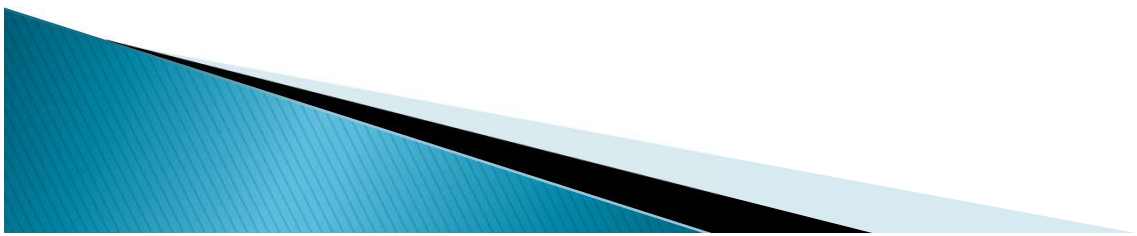


**Program Development**

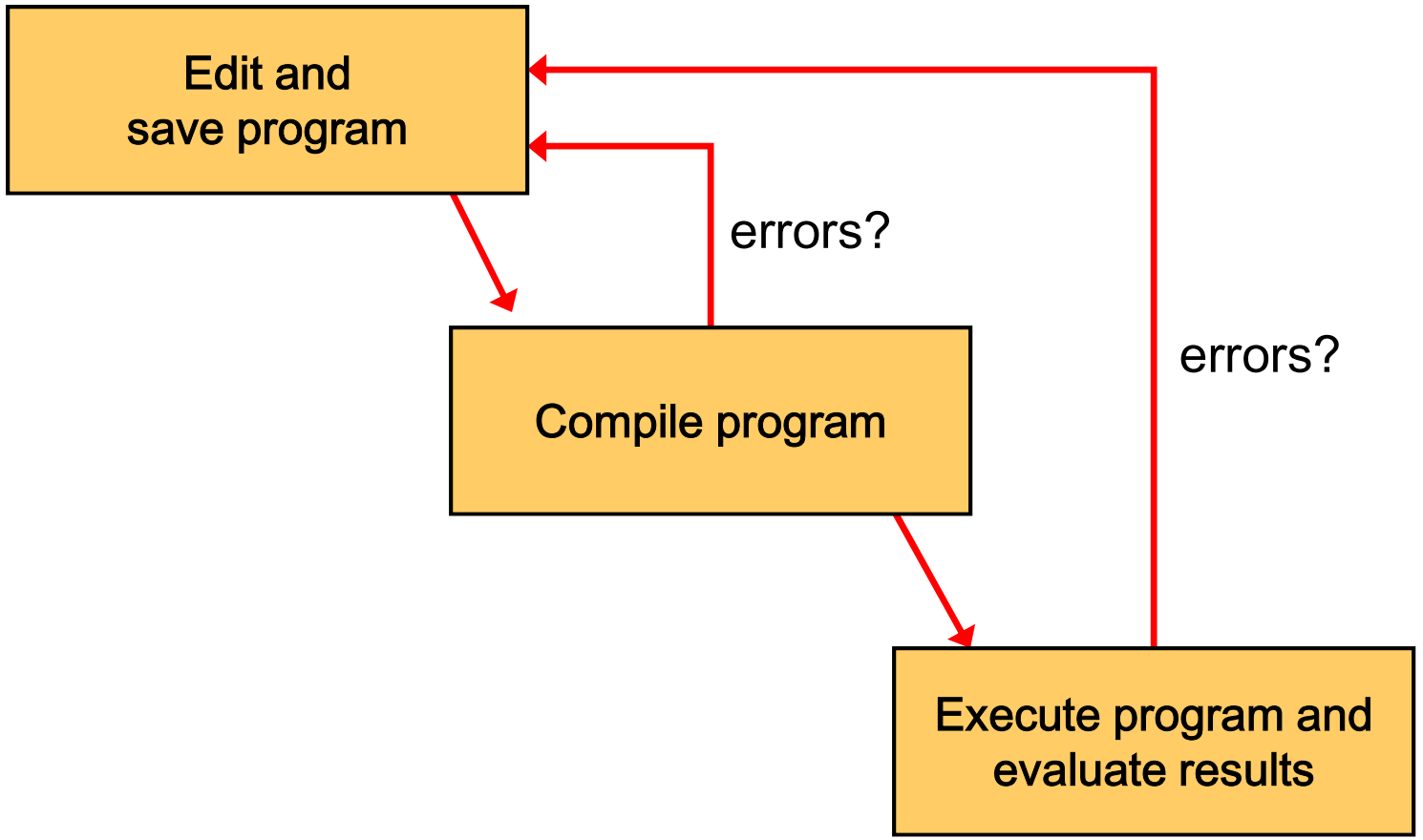
**The Java Programming Language**

# Program Development

- The mechanics of developing a program include several activities:
  - writing the program in a specific programming language (such as Java)
  - translating the program into a form that the computer can execute
  - investigating and fixing various types of errors that can occur
- Software tools can be used to help with all parts of this process



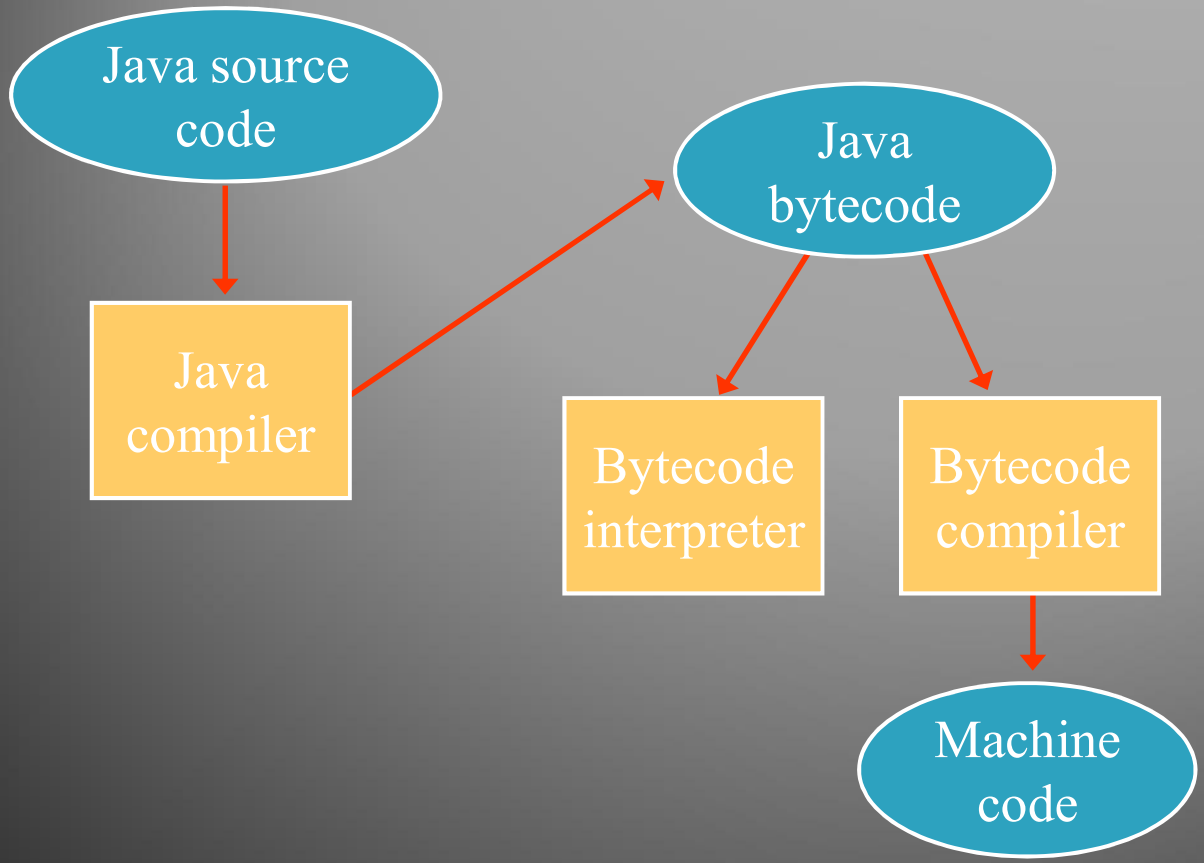
# Basic Program Development



# Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*

# Java Translation

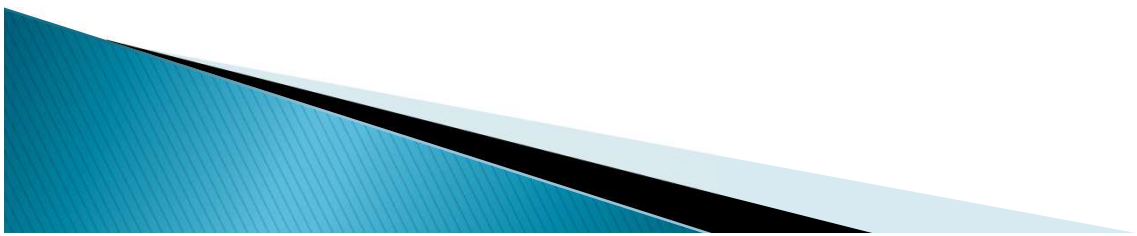


# Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

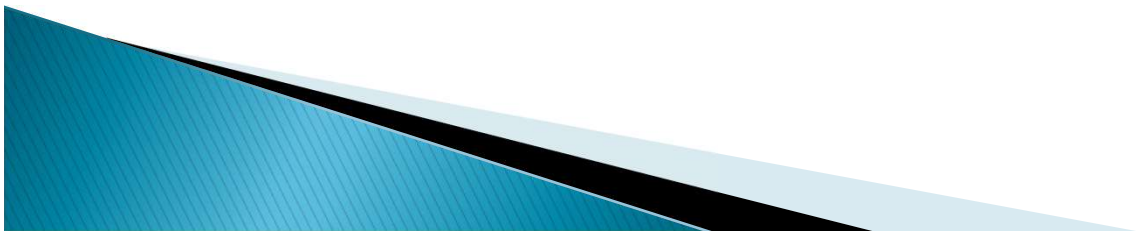
# Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (*compile-time errors*)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)



# Development Environments

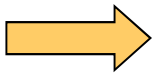
- There are many programs that support the development of Java software, including:
  - Java Development Kit (JDK)
  - Eclipse
  - NetBeans
  - BlueJ
  - jGRASP
- Though the details of these environments differ, the basic compilation and execution process is essentially the same



# Outline

**Object-Oriented Programming**

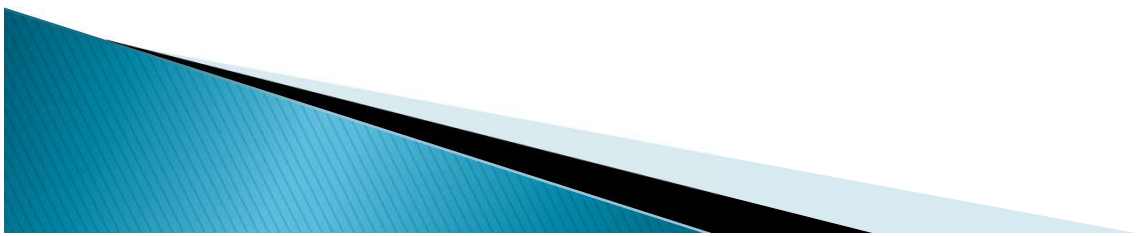
**Program Development**



**The Java Programming Language**

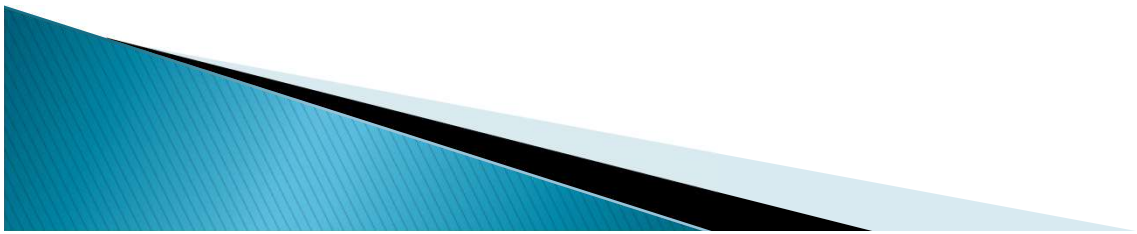
# Java

- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and it's popularity has grown quickly since
- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*



# Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java stand-alone application always contains a method called **`main`**
- See `Lincoln.java`



```
//********************************************************************
//  Lincoln.java       Author: Lewis/Loftus
//
//  Demonstrates the basic structure of a Java application.
//********************************************************************

public class Lincoln
{
   //-----------------------------------------------------------------
   //  Prints a presidential quote.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("A quote by Abraham Lincoln:");

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

## Output

```
A quote by Abraham Lincoln:
Whatever you are, be a good one.
```

```
//********
//  Lincol                                                    **********
//
//  Demons
//*******************************************************************

public class Lincoln
{
   //-----------------------------------------------------------------
   //  Prints a presidential quote.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("A quote by Abraham Lincoln:");

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

# Java Program Structure

```
//  comments about the class
public class MyProgram
{


}
```

class header

class body

Comments can be placed almost anywhere

# Java Program Structure

```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main (String[] args)
   {

   }

}
```

method header

method body

# Comments

- Comments should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks        */

/** this is a javadoc comment   */
```

# Identifiers

- *Identifiers* are the "words" in a program
- A Java identifier can be made up of letters, digits, the underscore character ( _ ), and the dollar sign
- Identifiers cannot begin with a digit
- Java is *case sensitive*: `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
  - *title case* for class names – `Lincoln`
  - *upper case* for constants – `MAXIMUM`

# Identifiers

- Sometimes the programmer chooses the identifer(such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

# Reserved Words

- The Java reserved words:

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

# Quick Check

Which of the following are valid Java identifiers?

```
grade
quizGrade
NetworkConnection
frame2
3rdTestScore
MAXIMUM
MIN_CAPACITY
student#
Shelves1&2
```

# Quick Check

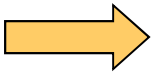
Which of the following are valid Java identifiers?

| | |
|---|---|
| `grade` | Valid |
| `quizGrade` | Valid |
| `NetworkConnection` | Valid |
| `frame2` | Valid |
| `3rdTestScore` | Invalid – cannot begin with a digit |
| `MAXIMUM` | Valid |
| `MIN_CAPACITY` | Valid |
| `student#` | Invalid – cannot contain the '#' character |
| `Shelves1&2` | Invalid – cannot contain the '&' character |

# White Space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation
- See `Lincoln2.java` and `Lincoln3.java`

# Outline



**Variables and Assignment**

**Primitive Data Types**

**Character Strings**

**Expressions**

**Data Conversion**

**Interactive Programs**

# Variables

- A *variable* is a name for a location in memory that holds a value

- Think of it as a box

# Variables

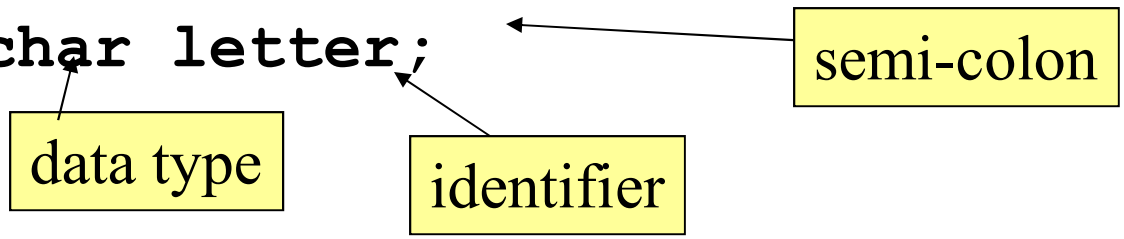
- Think of it as a box

- A variable has three things:
  - a name : that is how we refer to it
  - a type : what kind of a thing is stored in the box
  - a value : what is in the box

# Variable Declaration

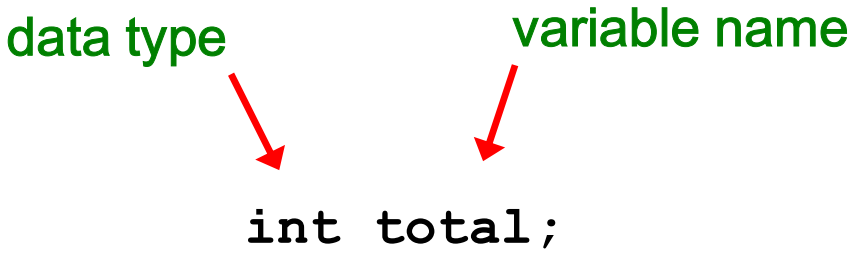
- Before you use a variable, you must declare it. (Not all languages require this, but Java certainly does.)
- Examples:

```
/*  Creates an integer variable */
int number;
/*  Creates two double variables */
double price, tax;
/*  Creates a character variable */
char letter;
```



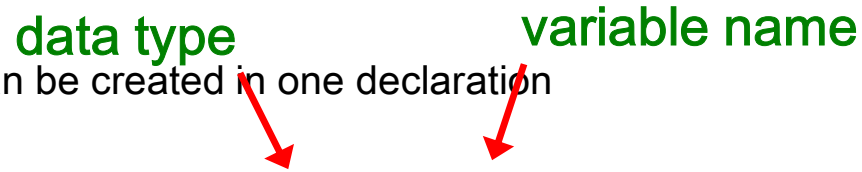
# Variable declaration

- A *variable declaration* specifies the variable's name and the type of information that it will hold:
- Before you use a variable, you must declare it. (Not all languages require this, but Java certainly does.)



# Variable declaration

- A *variable declaration* specifies the variable's name and the type of information that it will hold:




- Multiple variables can be created in one declaration

```
int total;
```

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

# Variable declaration

- Examples:

```
//  Declares an integer variable
int number;

 //  Declares two double variables
double price, tax;

 //  Declares a character variable
char letter;
```

# Variable declaration

- Examples:

```
//  Declares an integer variable
int number;

 //  Declares two double variables
double price, tax;

 //  Declares a character variable
char letter;
```

# Variable Initialization

- A variable can be given an initial value in the declaration. This is called variable initialization.

```
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.");
   }
}
```

```
//********************************************************************
//  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.");
   }
}
```

**Output**

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

# Assignment

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

```
total = 55;
```

# 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`

```
//********************************************************************
//  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.");
   }
}
```

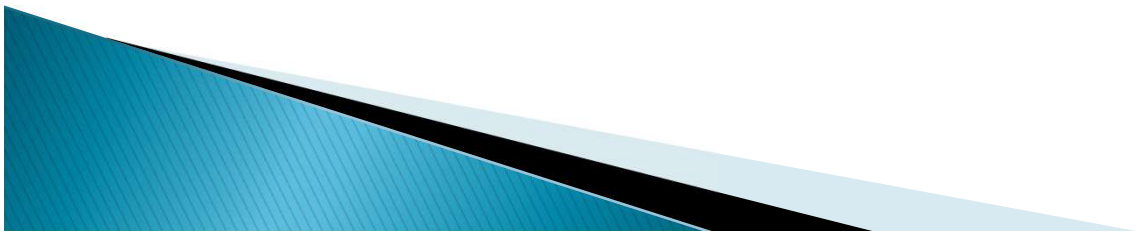
## Output

```
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, does not vary (cannot exist in the left hand side of the assignment operator in an assignment statement, because its value is **`final`**ized)



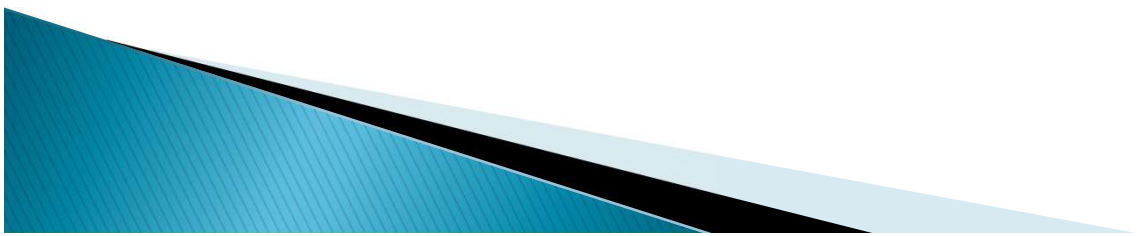
# Constants

- 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, does not vary
- 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

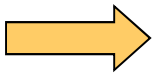
```
final int MIN_HEIGHT = 69;
```

# Constants

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



# Outline



**Variables and Assignment**

**Primitive Data Types**

**Character Strings**

**Expressions**

**Data Conversion**

**Interactive Programs**

# Java Data Types

- There are 2 basic data types in Java:
  - Primitive data types:
    - byte, short, int, long, float, double, char, boolean
  - Non-Primitive (Reference) data types
    - Examples: String, File, Scanner, ArrayList,..

# 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 (8 significant figures)`**
  - **`double (16 significant figures)`**
- 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:

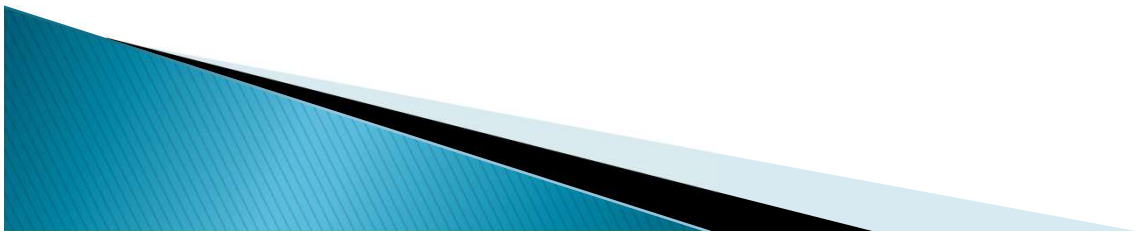
| Type | Storage | Min Value | Max Value |
|---|---|---|---|
| `byte` | 8 bits | -128 | 127 |
| `short` | 16 bits | -32,768 | 32,767 |
| `int` | 32 bits | -2,147,483,648 | 2,147,483,647 |
| `long` | 64 bits | $< -9 \times 10^{18}$ | $> 9 \times 10^{18}$ |
| `float` | 32 bits | $+/- 3.4 \times 10^{38}$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

# 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



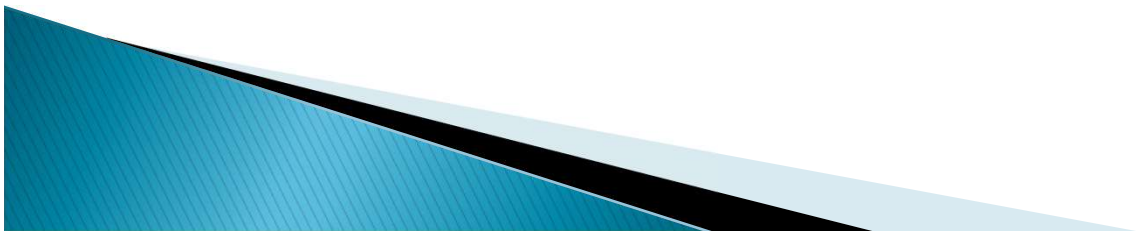
# Characters

- 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 = ' ';
```



# Character Sets

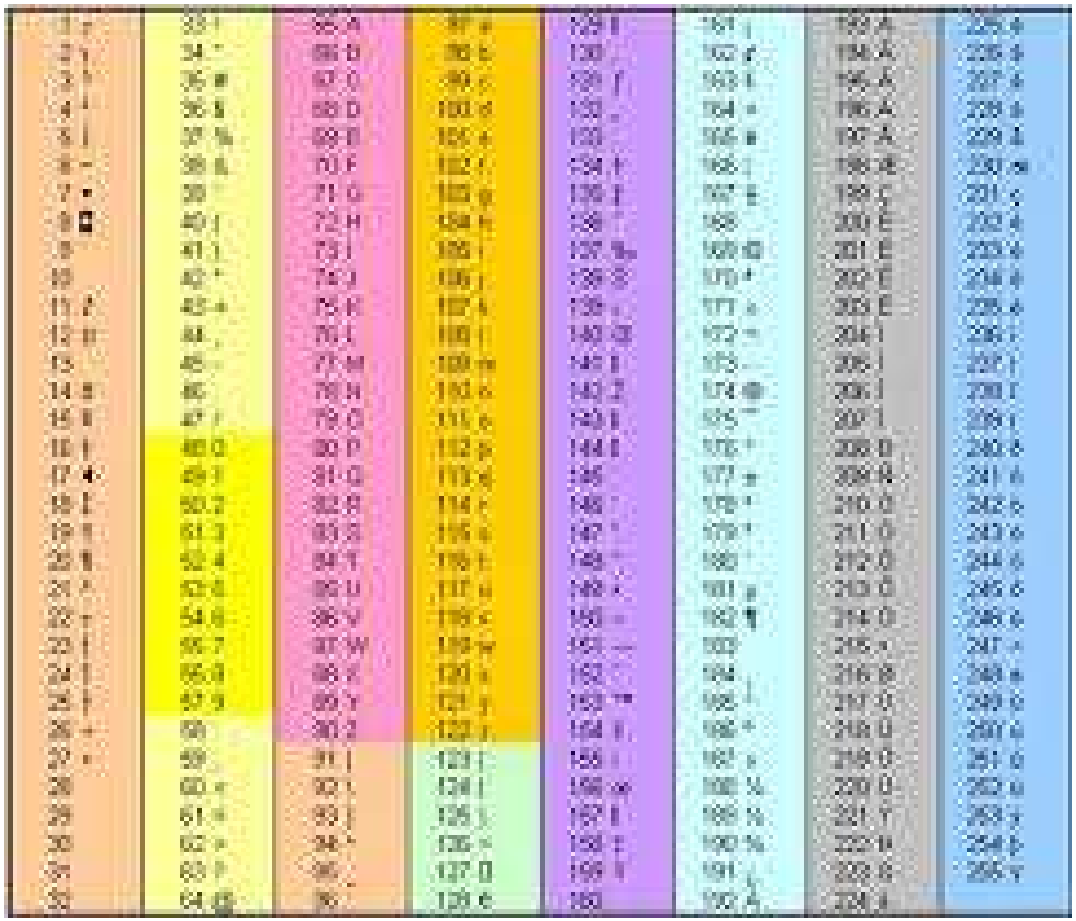
- A *character set* is an ordered list of characters, with each character corresponding to a unique number
- A `char` variable in Java can store any character from the *Unicode character set*
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an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

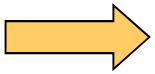
# ASCII Table



# Outline

**Variables and Assignment**

**Primitive Data Types**



**Character Strings**

**Expressions**

**Data Conversion**

**Interactive Programs**

# Character Strings

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

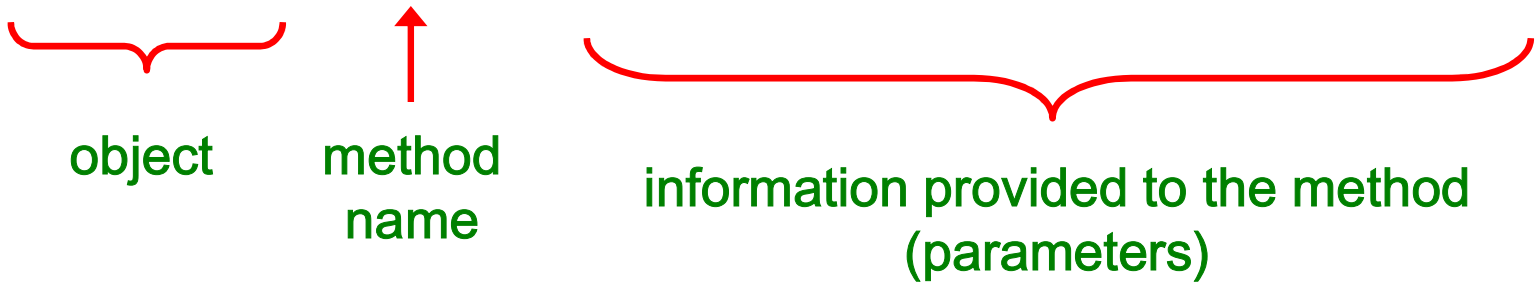
```
"This is a string literal."
"X"
```

- Every character string is an object in Java, defined by the `String` class
- A String object is an ordered set of characters. The number of characters can be 0. A String of 0 characters is called an empty String, which is expressed as ""
- Note that 'A' has the data type `char` whereas "A" is a String object

# The println Method

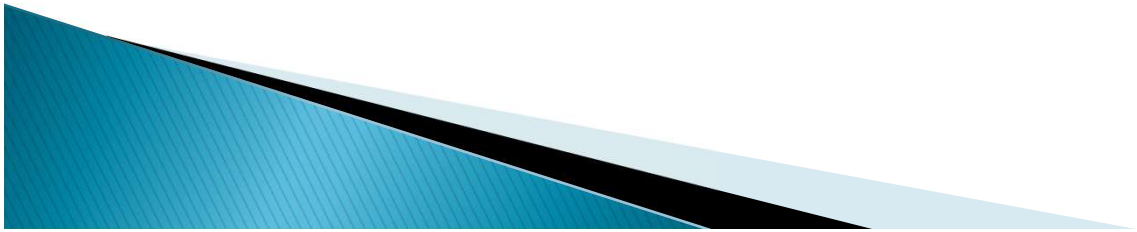
- In the `Lincoln` program from Chapter 1, we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

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



# 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 `Welcome2.java`
- See `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!");  // appears on first output line
      System.out.println ("Houston, we have a problem.");
   }
}
```

## Output

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

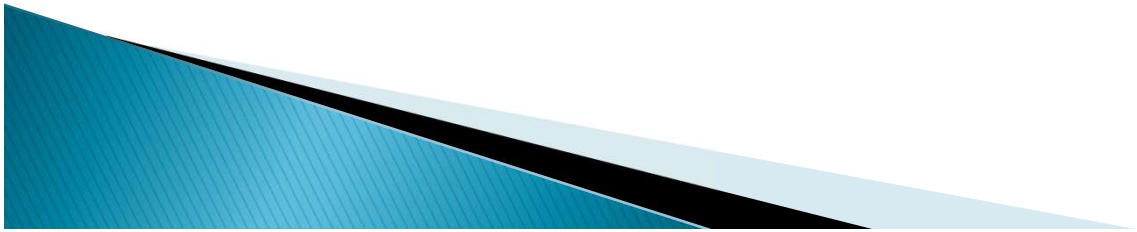
```
//****                                                          ****
//  Co
//
//  De
//****                                                          ****

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");
    }
}
```

```
//**********
//  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));
   }
}
```

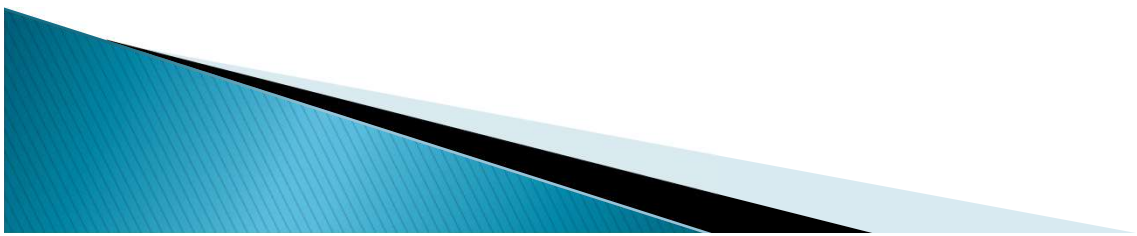
**Output**

```
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);
```

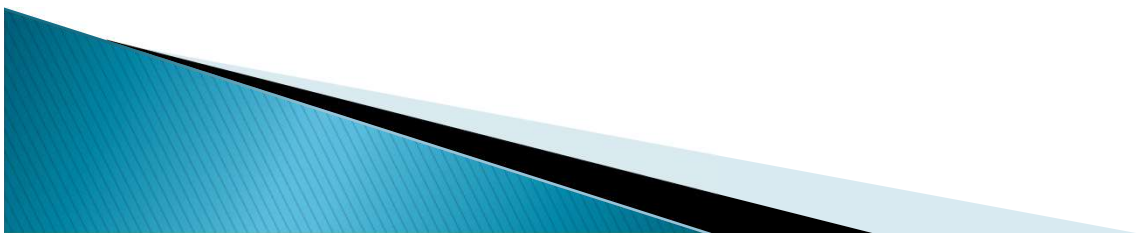


# Quick Check

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
```



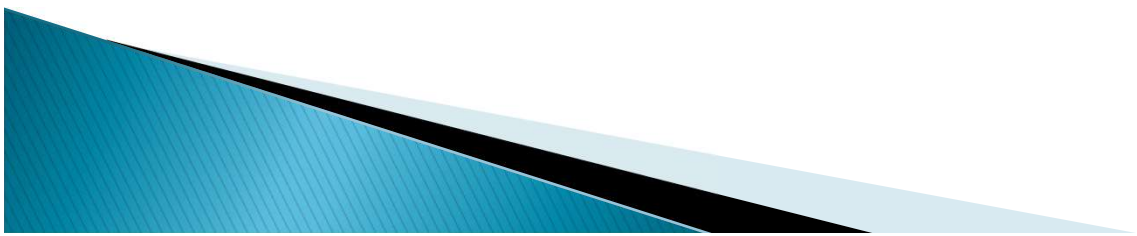
# Escape Sequences

- 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 |

- Note: One must use disable the "Capture output" check box in the default run application tool in Jcreator to be able to see the affects of \b \r
- See `Welcome3.java` and `Roses.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

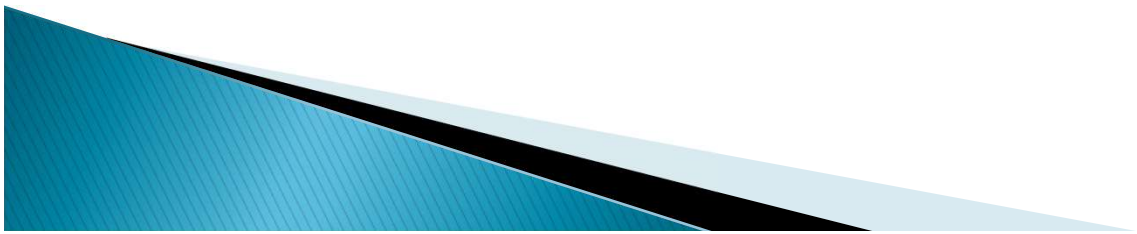
```
Roses are red,
      Violets are blue,
Sugar is sweet,
      But I have "commitment issues",
      So I'd rather just be friends
      At this point in our relationship.
```

```
//****
//  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.

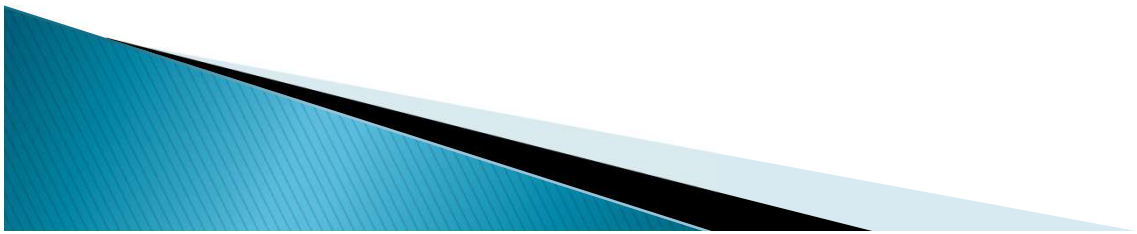


# Quick Check

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.");
```

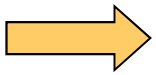


# Outline

**Character Strings**

**Variables and Assignment**

**Primitive Data Types**



**Expressions**

**Data Conversion**

**Interactive Programs**

# 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

•See TempConverter.java

# Division

- 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
```

# Remainder

- 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  =  3
```

# 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