

CS114

Introduction to Programming for Engineers

Week 1

10/06/2013

Course objective

- Upon satisfactory completion, students will:
 - Be able to implement, compile, test, and run simple Java programs
 - Be able to use simple data structures

In other words..

What the class is really about

There are two main goals of this course:

1. Core concepts of programming languages
2. Basics of java

plus

3. Learn the Principles of Software Development

Course website

- We'll be using Moodle
- It will contain the following information:
 - Administrative information
 - Course Syllabus
 - Lab assignment & solutions
 - Class notes
 - Class programs
 - Class handouts
 - Discussion forum

Software

- For the course, we will be using JCreator as the IDE.
- 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 7 lab assignments on Fridays each week, which you are required to solve individually during the lab sessions.
- 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.

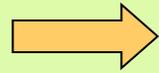
Communication

- 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.

Most important rule of the class: Be **Civil** and **Reasonable** at all times

- In an effort to make this class enjoyable for everybody...
 - Please be on time to class!
 - Please do not talk to your friends in class! It disturbs everyone, and makes it hard to concentrate. You don't only harm yourself but your fellows.
 - No cell-phones!

Outline



The Java Programming Language

Program Development

Object-Oriented Programming

Java

- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and its 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

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

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

```
*****
```

```
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)
    {
        }
    }
}
```



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 identifier (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:

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

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?

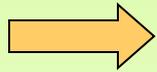
| | |
|--------------------------------|---|
| <code>grade</code> | Valid |
| <code>quizGrade</code> | Valid |
| <code>NetworkConnection</code> | Valid |
| <code>frame2</code> | Valid |
| <code>3rdTestScore</code> | Invalid – cannot begin with a digit |
| <code>MAXIMUM</code> | Valid |
| <code>MIN_CAPACITY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

The Java Programming Language



Program Development

Object-Oriented Programming

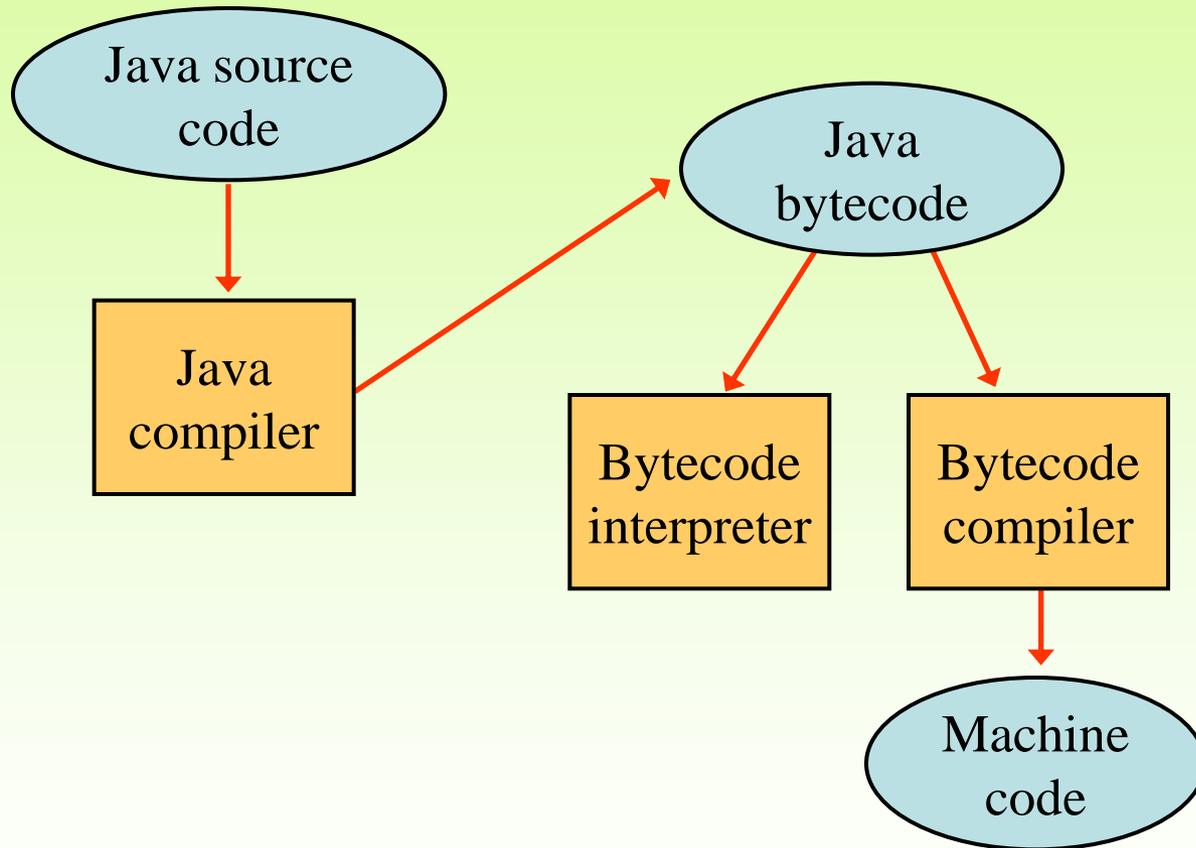
Program Development

- 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

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



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

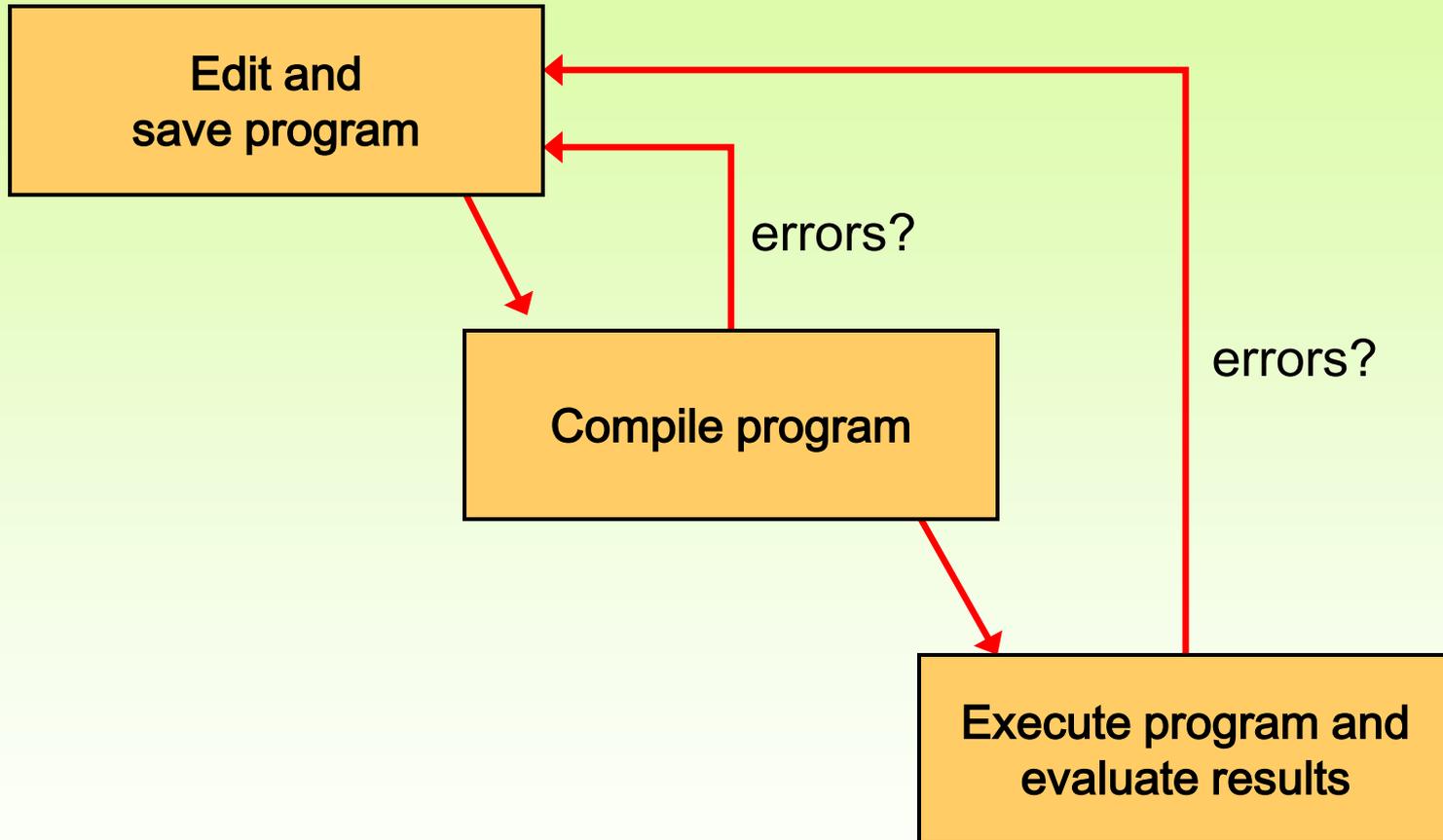
Syntax and Semantics

- 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*)

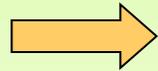
Basic Program Development



Outline

The Java Programming Language

Program Development



Object-Oriented Programming

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
- We will dissect our solutions into pieces called objects and classes

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
- For instance, an object might represent a particular employee in a company
- Each employee object handles the processing and data management related to that employee

Objects

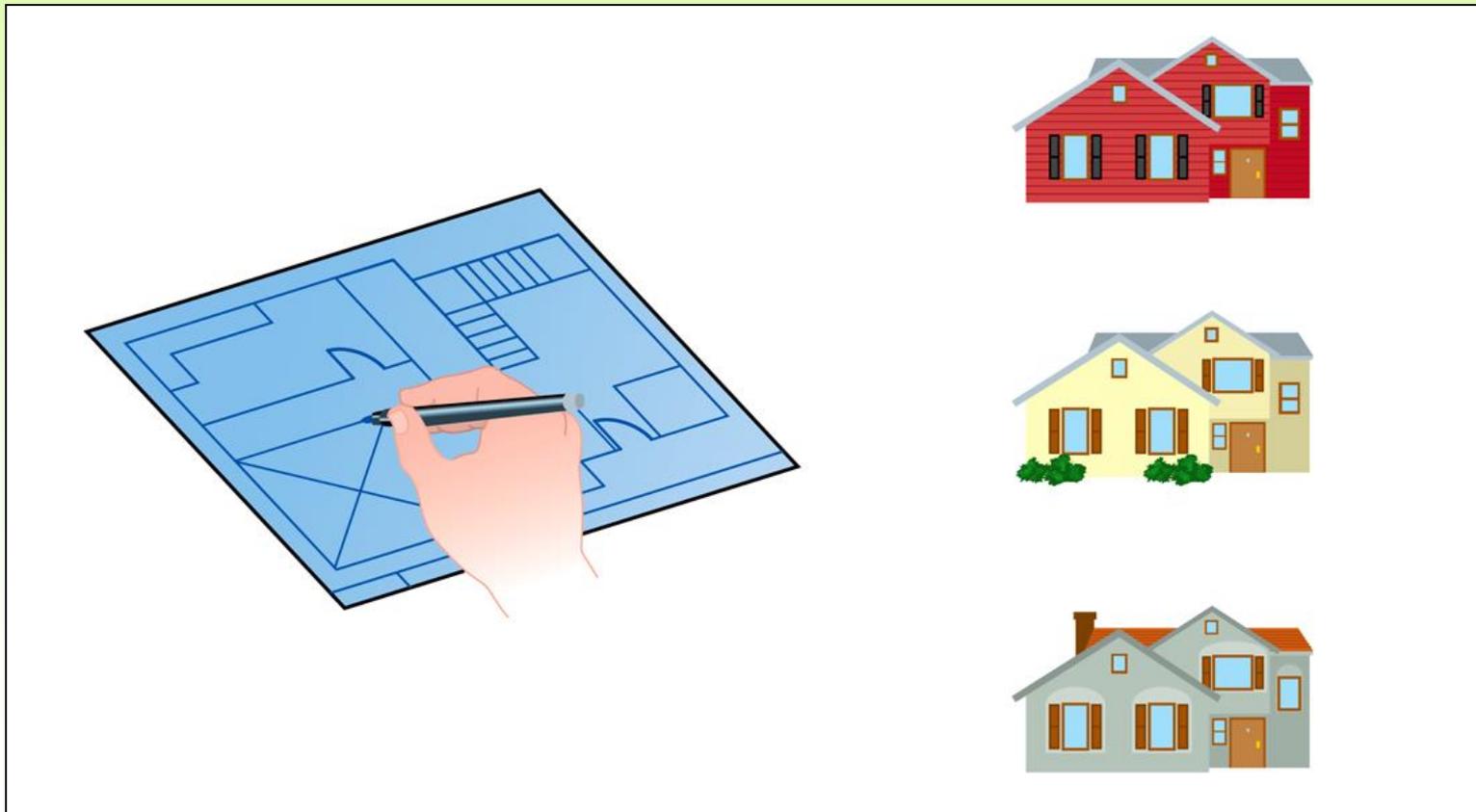
- An object has:
 - *state* - descriptive characteristics
 - *behaviors* - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviors associated with a bank account include the ability to make deposits and withdrawals
- Note that the behavior of an object might change its state

Classes

- An object is defined by a *class*
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class

Class = Blueprint

- One blueprint to create several similar, but different, houses:

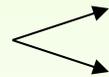


Objects and Classes

**A class
(the concept)**



**Multiple objects
from the same class**



**An object
(the realization)**

John's Bank Account
Balance: \$5,257

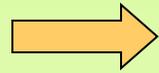
Bill's Bank Account
Balance: \$1,245,069

Mary's Bank Account
Balance: \$16,833

Summary

- Chapter 1 focused on:
 - components of a computer
 - how those components interact
 - how computers store and manipulate information
 - computer networks
 - the Internet and the World Wide Web
 - programming and programming languages
 - an introduction to Java
 - an overview of object-oriented concepts

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;
```

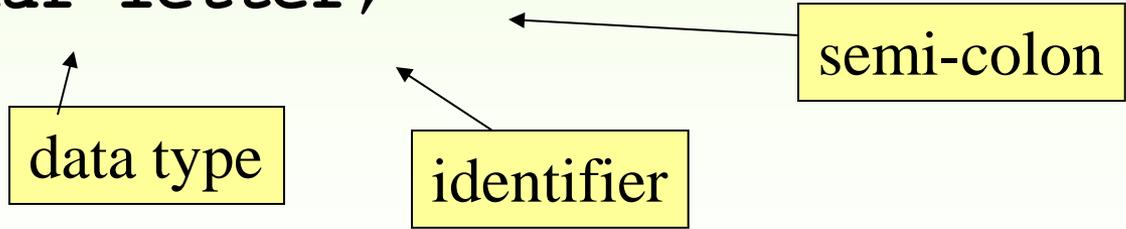
```
/* Creates two double variables */
```

```
double price, tax;
```

```
/* Creates a character variable */
```

```
char letter;
```

data type



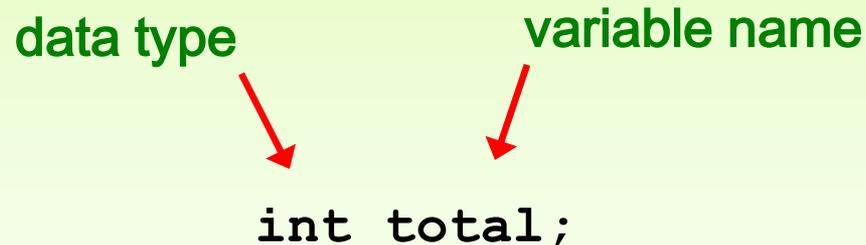
identifier

semi-colon

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.)

data type variable name



```
int total;
```

Variable declaration

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

data type variable name



```
int total;
```

- Multiple variables can be created in one declaration

```
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.");
    }
}

```

Output

```

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

```

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

```
//*****  
// Geometry.java  
//  
// Demonstrate  
// value stored  
//*****
```

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

```
*****
```

```
change the
```

```
*****
```

```
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 **finalized**)

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

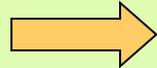
```
final int MIN_HEIGHT = 69;
```

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:

| <u>Type</u> | <u>Storage</u> | <u>Min Value</u> | <u>Max Value</u> |
|-------------|----------------|--|----------------------|
| 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 | +/- 3.4×10^{38} with 7 significant digits | |
| double | 64 bits | +/- 1.7×10^{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

| | | | | | | | |
|------|-------|------|-------|--------|-------|-------|-------|
| 1 r | 33 | 65 A | 97 a | 129 i | 161 | 193 A | 225 a |
| 2 l | 34 * | 66 B | 98 b | 130 , | 162 f | 194 A | 226 b |
| 3 ^ | 35 # | 67 C | 99 c | 131 / | 163 k | 195 A | 227 k |
| 4 ^ | 36 \$ | 68 D | 100 d | 132 - | 164 * | 196 A | 228 s |
| 5 | 37 % | 69 E | 101 e | 133 . | 165 w | 197 A | 229 i |
| 6 - | 38 & | 70 F | 102 f | 134 + | 166 l | 198 A | 230 m |
| 7 * | 39 ^ | 71 G | 103 g | 135 = | 167 s | 199 C | 231 c |
| 8 0 | 40 | 72 H | 104 h | 136 ~ | 168 | 200 E | 232 k |
| 9 | 41 j | 73 I | 105 i | 137 % | 169 0 | 201 E | 233 s |
| 10 | 42 * | 74 J | 106 j | 138 0 | 170 + | 202 E | 234 k |
| 11 z | 43 + | 75 K | 107 k | 139 < | 171 < | 203 E | 235 s |
| 12 # | 44 , | 76 L | 108 l | 140 0 | 172 ~ | 204 I | 236 i |
| 13 | 45 - | 77 M | 109 m | 141 I | 173 - | 205 I | 237 i |
| 14 # | 46 . | 78 N | 110 n | 142 J | 174 0 | 206 I | 238 i |
| 15 # | 47 / | 79 O | 111 o | 143 I | 175 ~ | 207 I | 239 i |
| 16 + | 48 0 | 80 P | 112 p | 144 I | 176 * | 208 D | 240 s |
| 17 # | 49 1 | 81 Q | 113 q | 145 | 177 s | 209 R | 241 a |
| 18 I | 50 2 | 82 R | 114 r | 146 ~ | 178 * | 210 0 | 242 s |
| 19 # | 51 3 | 83 S | 115 s | 147 ^ | 179 * | 211 0 | 243 s |
| 20 # | 52 4 | 84 T | 116 t | 148 ~ | 180 ^ | 212 0 | 244 s |
| 21 ^ | 53 5 | 85 U | 117 u | 149 * | 181 u | 213 0 | 245 s |
| 22 z | 54 6 | 86 V | 118 v | 150 - | 182 u | 214 0 | 246 s |
| 23 I | 55 7 | 87 W | 119 w | 151 - | 183 | 215 * | 247 + |
| 24 I | 56 8 | 88 X | 120 x | 152 ~ | 184 , | 216 0 | 248 s |
| 25 F | 57 9 | 89 Y | 121 y | 153 ** | 185 ^ | 217 0 | 249 s |
| 26 - | 58 | 90 Z | 122 z | 154 I | 186 * | 218 0 | 250 s |
| 27 * | 59 . | 91 | 123 | 155 < | 187 s | 219 0 | 251 s |
| 28 | 60 < | 92 L | 124 | 156 0 | 188 % | 220 0 | 252 s |
| 29 | 61 = | 93 | 125 | 157 I | 189 % | 221 Y | 253 y |
| 30 | 62 > | 94 ^ | 126 ~ | 158 I | 190 % | 222 0 | 254 s |
| 31 | 63 ? | 95 ~ | 127 0 | 159 I | 191 L | 223 s | 255 y |
| 32 | 64 @ | 96 ~ | 128 e | 160 | 192 A | 224 i | |

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

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

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

```
****  
  
****
```

```
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

```
//*****  
// Addition  
//  
// Demonstrates concatenating two numbers and adding them together.  
// concatenates two numbers and prints the results.  
//*****
```

24 and 45 concatenated: 2445

24 and 45 added: 69

string

```
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));  
    }  
}
```

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:

| <u>Escape Sequence</u> | <u>Meaning</u> |
|------------------------|-----------------|
| <code>\b</code> | backspace |
| <code>\t</code> | tab |
| <code>\n</code> | newline |
| <code>\r</code> | carriage return |
| <code>\"</code> | double quote |
| <code>'</code> | single quote |
| <code>\\</code> | 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

```
/** **
// Ro Roses are red,
//
// De Violets are blue,
/** **
Sugar is sweet,
But I have "commitment issues",
So I'd rather just be friends
At this point in our relationship.
-----
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\\n\"tonight,\" 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)

$$14 / 3 \text{ equals } 4$$

$$8 / 12 \text{ equals } 0$$

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

Quick Check

In what order are the operators evaluated in the following expressions?

$a + b + c + d + e$

$a + b * c - d / e$

$a / (b + c) - d \% e$

$a / (b * (c + (d - e)))$

Quick Check

In what order are the operators evaluated in the following expressions?

$$a + b + c + d + e$$

1 2 3 4

$$a + b * c - d / e$$

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

$$a / (b * (c + (d - e)))$$

4 3 2 1

Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer = sum / 4 + MAX * lowest;
```

(4) (1) (3) (2)



Then the result is stored in the variable on the left hand side

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)

Increment and Decrement

- The increment (++) and decrement (--) operators use only one operand
- The statement

```
count++;
```

is functionally equivalent to

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

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators in Java, including the following:

| <u>Operator</u> | <u>Example</u> | <u>Equivalent To</u> |
|-----------------|----------------|----------------------|
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the `+=` operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (`+=`) is always consistent with the behavior of the corresponding operator (`+`)

Outline

Variables and Assignment

Primitive Data Types

Character Strings

Expressions

 **Data Conversion**

Interactive Programs

Data Conversion

- Sometimes it is convenient to convert data from one type to another
- For example, in a particular situation we may want to treat an integer as a floating point value
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversion

- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)
- In Java, data conversions can occur in three ways:
 - assignment conversion
 - promotion
 - casting

Data Conversion

Widening Conversions

| From | To |
|-------|------------------------------------|
| byte | short, int, long, float, or double |
| short | int, long, float, or double |
| char | int, long, float, or double |
| int | long, float, or double |
| long | float or double |
| float | double |

Narrowing Conversions

| From | To |
|--------|--|
| byte | char |
| short | byte or char |
| char | byte or short |
| int | byte, short, or char |
| long | byte, short, char, or int |
| float | byte, short, char, int, or long |
| double | byte, short, char, int, long, or float |

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- Example:

```
int dollars = 20;  
double money = dollars;
```

- Only widening conversions can happen via assignment
- Note that the value or type of `dollars` did not change

Promotion

- *Promotion* happens automatically when operators in expressions convert their operands
- Example:

```
int count = 12;  
double sum = 490.27;  
result = sum / count;
```

- The value of `count` is converted to a floating point value to perform the division calculation

Casting

- *Casting* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;  
float result = (float) total / 6;
```

- Without the cast, the fractional part of the answer would be lost

Quick check

- What is the result of the following assignments?

```
int a = 7, b = 3;  
double c, d;  
c = a / b;  
d = (double) a / b;
```

Quick check

- What is the result of the following assignments?

```
int a = 7, b = 3;  
double c, d;  
c = a / b;  
d = (double) a / b;
```

c = 2.0

d = 2.3333333333333333

Outline

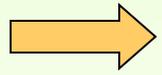
Variables and Assignment

Primitive Data Types

Character Strings

Expressions

Data Conversion

 **Interactive Programs**

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- The following line creates a `Scanner` object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` package, and must be imported in order to be used
- All import statements must be inserted in the class before the class header:

```
import java.util.Scanner;
public class Example {
...
}
```

- You can use the wildcard `*` to import all the classes within a package:

```
import java.util.*;
```

Some methods from Scanner class

| Data Type | Corresponding method |
|-----------|----------------------|
| byte | nextByte() |
| short | nextShort() |
| int | nextInt() |
| long | nextLong() |
| float | nextFloat() |
| double | nextDouble() |
| boolean | nextBoolean() |
| char | next().charAt(0) |
| String | next() nextLine() |

Some methods from Scanner class

Scanner class uses whitespace (spaces, tabs, new-lines) to extract a token from the source. We call the whitespace (shown as \s) as the default delimiter. The delimiter can be changed through the **useDelimiter**

Method. Example:

```
Scanner scan = new Scanner (System.in);  
scan.useDelimiter("; ");
```

Some methods from Scanner class

- Normally, the Scanner class uses the computers own Locale settings to convert a token into its data type such as a double. But, US Locale uses a `.` as a decimal point whereas Turkish Locale uses a `,` as a decimal point. If you use the wrong decimal point, you may end up with a run-time error when using `nextFloat()` or `nextDouble()`
- Use `scan.useLocale(Locale.US)` to avoid such problems and hence always use `.` as the decimal point! But, you must also import the Locale class:

```
import java.util.Locale;
```

nextLine() method of Scanner class

The nextLine method reads all of the input until the end of the line is found.

See [Echo.java](#)

You must be careful when using nextLine() after using any of the other next(), nextInt(), nextDouble(),... methods. You may need to call the nextLine() method twice! (First one consumes the new line character not consumed before by the non-nextLine() method, and the second one reads the real string until end-of-line character.) Alternatively, one may prefer to reserve one Scanner object for nextLine() and another for non-nextLine() methods!

```

//*****
//  Echo.java      Author: Lewis/Loftus
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a string from the user.
//*****

import java.util.Scanner;

public class Echo
{
    //-----
    //  Reads a character string from the user and prints it.
    //-----
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);

        System.out.println ("Enter a line of text:");

        message = scan.nextLine();

        System.out.println ("You entered: \"" + message + "\"");
    }
}

```

Sample Run

```
/**  
// Echo  
//  
// Description  
// to  
/**
```

Enter a line of text:

You want fries with that?

You entered: "You want fries with that?"

```
***  
  
s  
  
***
```

```
import java.util.Scanner;
```

```
public class Echo
```

```
{
```

```
    //-----  
    // Reads a character string from the user and prints it.  
    //-----
```

```
    public static void main (String[] args)
```

```
    {
```

```
        String message;
```

```
        Scanner scan = new Scanner (System.in);
```

```
        System.out.println ("Enter a line of text:");
```

```
        message = scan.nextLine();
```

```
        System.out.println ("You entered: \"" + message + "\"");
```

```
    }
```

```
}
```

Input Tokens

- Unless specified otherwise (by the `useDelimiter` method) , *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
- See [GasMileage.java](#)

```
//*****
// GasMileage.java      Author: Lewis/Loftus
//
// Demonstrates the use of the Scanner class to read numeric data.
//*****

import java.util.Scanner;

public class GasMileage
{
    //-----
    // Calculates fuel efficiency based on values entered by the
    // user.
    //-----
    public static void main (String[] args)
    {
        int miles;
        double gallons, mpg;

        Scanner scan = new Scanner (System.in);
```

continue

continue

```
System.out.print ("Enter the number of miles: ");
miles = scan.nextInt();

System.out.print ("Enter the gallons of fuel used: ");
gallons = scan.nextDouble();

mpg = miles / gallons;

System.out.println ("Miles Per Gallon: " + mpg);
}
}
```

Sample Run

continue

```
Enter the number of miles: 328
Enter the gallons of fuel used: 11.2
Miles Per Gallon: 29.28571428571429
```

Sy

```
mi
gallons = scan.nextDouble();
```

```
mpg = miles / gallons;
```

```
System.out.println ("Miles Per Gallon: " + mpg);
```

```
}
```

```
}
```

More on Scanner

- See [UserAddition.java](#)

Summary

- character strings
- primitive data
- the declaration and use of variables
- expressions and operator precedence
- data conversions
- accepting input from the user