# Data Types

(both types are considered “variables”)

❖ Reference Data Types

- Objects are reference data types
- The identifier (i.e., object name) of the object is associated with a piece of memory containing an address. This address **refers** to another piece of memory containing object itself.

❖ Primitive Data Types

- The identifier (i.e., variable name) of the variable is associated with a piece of memory containing the actual data.
- There are eight (8) primitive data types:

Numerical Data Types

| | |
|---|---|
| byte | |
| short | Corresponds to Integers (numbers without fractions) |
| int | (typically use int for most Integers) |
| long | |
| float | Corresponds to Real Numbers (numbers with fractions) |
| double | (typically use double for most Real Numbers) |

Other Data Types

| | |
|---|---|
| char | Corresponds to a single alphabetical character/symbol |
| boolean | Corresponds to a value of either true or false |

# Variables

❖ A named memory location containing a certain type of data

❖ Three (3) Properties

1. Name: associates the variable with a particular memory location
2. Type: tells the computer how much memory to set aside for a particular variable
3. Value: the actual value sitting in the memory location

❖ Variable Declaration

- Associates a name with a memory location
- The value in the memory location can change
- Syntax:

```
<data type> <variable name>;
```

| | |
|---|---|
| <data type>: | The type of data assigned to the memory location being allocated |
| <variable name>: | any valid identifier |

- Examples:

```
int age;
float gpa;
long nationalDebt;
```

# Variables...continued

❖ Shortcut:

- When declaring more than one variable of the same type, can declare them all in one statement
- Syntax:

  &lt;data type&gt; &lt;var name&gt;, &lt;var name&gt;,&lt;var name&gt;;

- Examples:

```
double interestRate1, interestRate2;
int height, width, depth;
```

❖ NOTE: Cannot declare a variable more than once!

- Example:

```
int number;
float number;
```

# Assignment Statement

- Places a value into a variable using =, the "assignment" operator

- Syntax:

```
<variable> = <expression>;
```

| | |
|---|---|
| <variable>: | any previously declared variable |
| <expression>: | any expression that evaluates to a value of the same type as the variable |

Examples:

```
gpa = 3.74;
nationalDebt = 5000000000000;
```

- The first time a variable is assigned a value, it is said to be "initialized"

- A variable's value is overwritten when new assignment statements follow the initial assignment statement.

  Examples:

```
gpa = 4.0;
nationalDebt = 1000000000000000;
```

# Assignment Statement...continued

❖ Shortcut #1:

- Can declare and initialize a variable in one statement
- Syntax:

  <data type> <variable name> = <expression>;

  Examples:

  ```
  double prime = .065;
  double interestRate = prime + .0125;
  ```

❖ Shortcut #2:

- Can declare and initialize >1 variable in one statement

  Example:

  ```
  int x=0, y=1, z=2;
  int a, b, c=5;
  ```

- This usually is considered BAD programming practice and should only be done in a limited number of circumstances.

# Draw a Memory Diagram for the following Java code:

```
int deposit;
double intRate = .0785;

deposit = 100;
deposit = 200;

Calculator calc;
Account account1;
Account account2 = new Account (intRate);
account1 = account2;
account1 = new Account (intRate);
```

# Constants

- Associates a name with an unchanging value
- Syntax:

```
final <data type> <constant name> = <value>;
```

- the constant is declared and assigned a value in one step
- Java convention: <constant name> refers to an identifier with ALL_CAPITAL_LETTERS and with words separated by underscores
- Examples:

```
final double PI = 3.1415926;

final int DAYS_IN_WEEK = 7;
```

- Why Constants?
  1. Gives a name to an unchanging value
  2. Makes programs more readable and understandable
  3. Easier to update in one location rather than multiple locations

- Symbolic Constants vs. Literal Constants

  Symbolic Constant: a name associated with a value

  Literal Constant: the number itself

e.g.

```
PI      // Symbolic Constant

3.1415  // Literal Constant
```

# Arithmetic Expressions

- An expression involving numerical values that can be evaluated to some numerical value

- Consists of operands and operators

  - operand: The value or expression on which arithmetic is to be performed

  - operator: The symbol that signifies what type of arithmetic is to be performed

    - Binary operators: involve 2 operands

      Syntax: <operand> <operator> <operand>

      Example:
      ```
      2 + 5
      x / y
      ```

    - Unary operators: involve 1 operand

      Syntax: <operator> <operand>

      Example:
      ```
      -4.6
      +z      // rarely used
      ```

- Expressions
  - a part of a statement
  - no need for semi-colon at the end
  - Example:
    ```
    int x = (y / z) + 4;
    ```

  - Can have a multiple number of operands separated by a multiple number of operators

# Operators

+ - * / %

/ Division has two meanings depending on data type:

```
    int i1 = 8;
    int i2 = 6;
    double d1 = 8.0;
    double d2 = 6.0;

    int answer;
    double answer2;

    answer = i1 / i2;

    answer2 = d1 / d2;

    answer2 = i1 / d2;
```

% “Remainder Division” (aka “modulo” or “mod”)

```
    answer = i1 % i2;

    d1 = 22.5;
    d2 = 7.0;
    answer2 = d1 % d2;
```

Precendence Rules for operators

```
      11 + 22 * x – 2
```

# Type Casting

❖ Implicit Type Casting

Numeric Promotion

- Occurs AUTOMATICALLY when an arithmetic expression does not consist of variables and constants of the same data type
- The "promotion" is applied to the operands of an arithmetic operator
- The operand is converted from a lower to a higher precision
- Examples:

```
int i1 = 4;
double d1 = 6.0;

double answer = d1 / i1;
/* answer has the value 1.5 */
```

Assignment Conversion

- Occurs AUTOMATICALLY when a variable and the value of an expression in an assignment statement are not of the same data type
- Occurs ONLY if the data type of the variable has a higher precision than the data type of the expression
- Examples:

```
double d;
d = 5;     // d contains the value 5.0

int i;
i = 123.456;   // syntax error
```

# Type Casting...continued

❖ Explicit Type Casting

- uses the type cast operator:      (<data type>)
- Syntax:

  (<data type>) <expression>

- the type cast operator is a unary operator
- the type cast operator has higher precedence than any binary operator
- parentheses must enclose expressions to be type cast
- Examples

```
int i1 = 4;
int i2 = 6;
double d1 = 6.0;
double d2 = 8.0;

int answerI;
double answerD;

answerI = 8 / i2;

answerD = 8 / i2;

answerD = (double) 8 / i2;

answerI = i1 + i2;

answerI = (int) d1 + d2;

answerD = d2 / d1;

answerI = (int) d1 / i1;
```

# Math Class

- Contained in the package java.lang

- Contains functions (i.e., methods) that allow for operations other than
  + - * / %

- Methods are class methods (do not need to create a Math object in order to use the methods)

- Syntax for sending messages to class methods:

  <class name>.<method name> (<arguments>)

  NOTE: Sending a message to a class method is actually an expression that may evaluate to some value

- Examples:

```
double d = Math.pow (2.0, 3.0);

int i = Math.min (4, 8);
```

❖ See the following website for documentation on ALL predefined classes in Java, including the Math class (but not javabook!):

**http://java.sun.com/products/jdk/1.2/docs/api/index.html**

# class InputBox

- Contained in the package javabook

- Contains functions allowing for user input of numbers

- Requires that an “owner frame” be specified when creating an InputBox object (MainWindow object will be used)

- Sample Code to use InputBox:

```
MainWindow mw = new MainWindow (“myWindow”);
InputBox inBox = new InputBox (mw);
int x;
float y;

mw.show ();
x = inBox.getInteger (“Enter an integer”);
y = inBox.getFloat (“Enter the interest rate”);
```

# class OutputBox

- Contained in the package javabook

- Contains functions allowing for the display of a program's output (textual data only, no drawings)

- Requires that an "owner frame" be specified when creating an OutputBox object  (MainWindow object will be used)

- Sample Code using OutputBox:

```
MainWindow mw = new MainWindow (“myWindow”);
OutputBox outBox = new OutputBox (mw);

mw.show ();
outBox.show();

outBox.print (“Java is fun”);
```

# Concatenation Operator     +

- The symbol “+” is used both for addition and concatenation (considered an “overloaded” operator)

- Examples:

```
“James Bond’s code name is “ + 0 + 0 + 7

0 + 0 + 7 + “ is James Bond’s code name.”
```

---

```
int a = 53;
int b = 70;
int c = 3;

“The zip code is “ a + b + c

a + b + c + “ is the zip code.”
```

---

“The sum of 8 and 9 is ” + 8 + 9;

“The sum of 8 and 9 is ”  + (8 + 9);