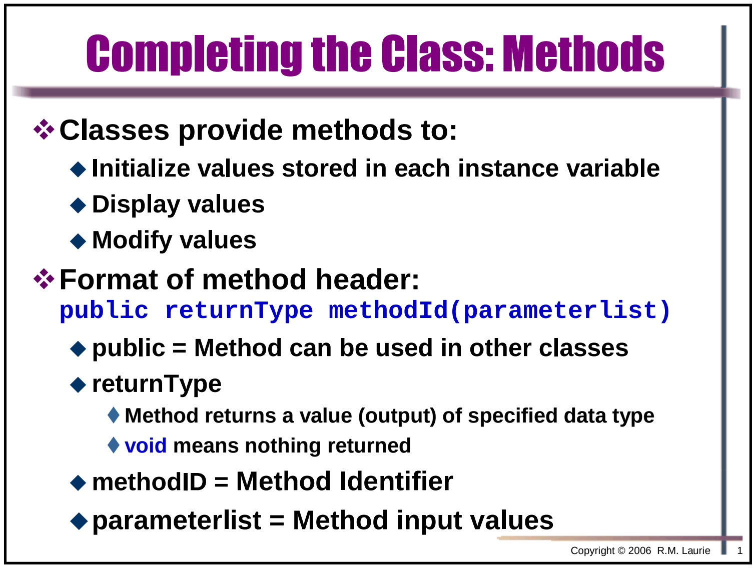
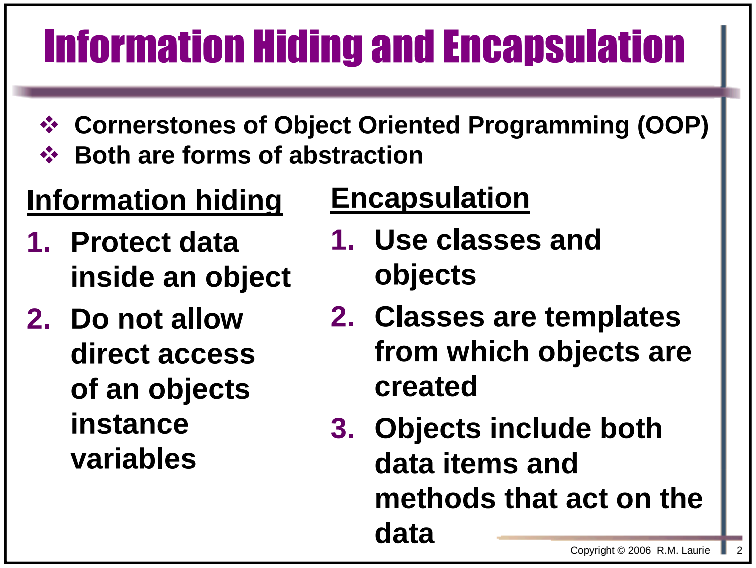


Completing the Class: Methods

- Classes provide methods to:
  - Initialize values stored in each instance variable
  - Display values
  - Modify values
- Format of method header:

```
public returnType methodId(parameterlist)
```

  - public = Method can be used in other classes
  - returnType
    - Method returns a value (output) of specified data type
    - void means nothing returned
  - methodID = Method Identifier
  - parameterlist = Method input values



Information Hiding and Encapsulation

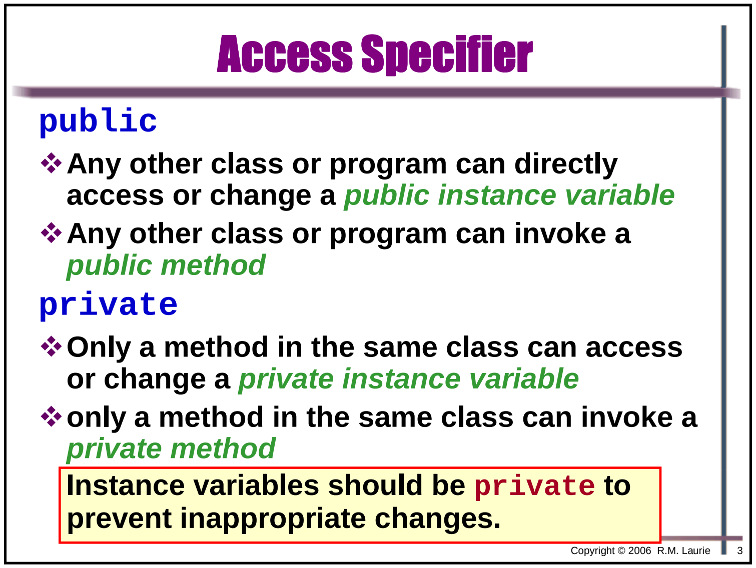
- Cornerstones of Object Oriented Programming (OOP)
- Both are forms of abstraction

Information hiding

1. Protect data inside an object
2. Do not allow direct access of an objects instance variables

Encapsulation

1. Use classes and objects
2. Classes are templates from which objects are created
3. Objects include both data items and methods that act on the data



Access Specifier

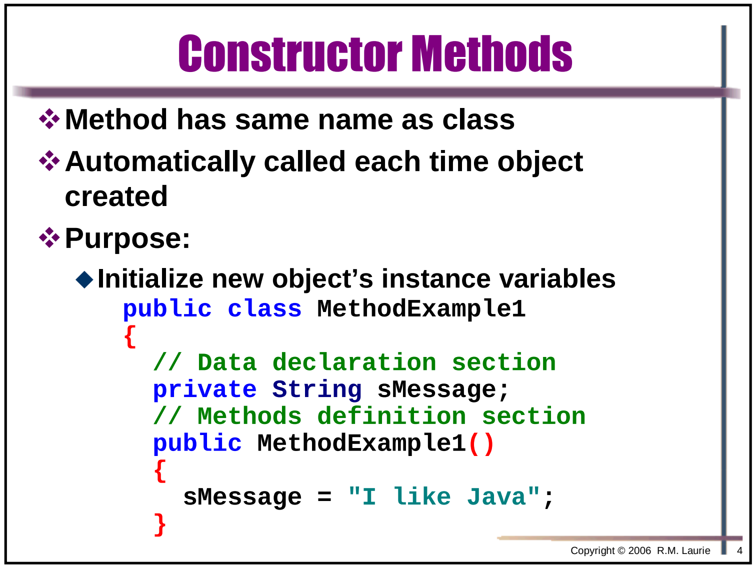
`public`

- Any other class or program can directly access or change a *public instance variable*
- Any other class or program can invoke a *public method*

`private`

- Only a method in the same class can access or change a *private instance variable*
- only a method in the same class can invoke a *private method*

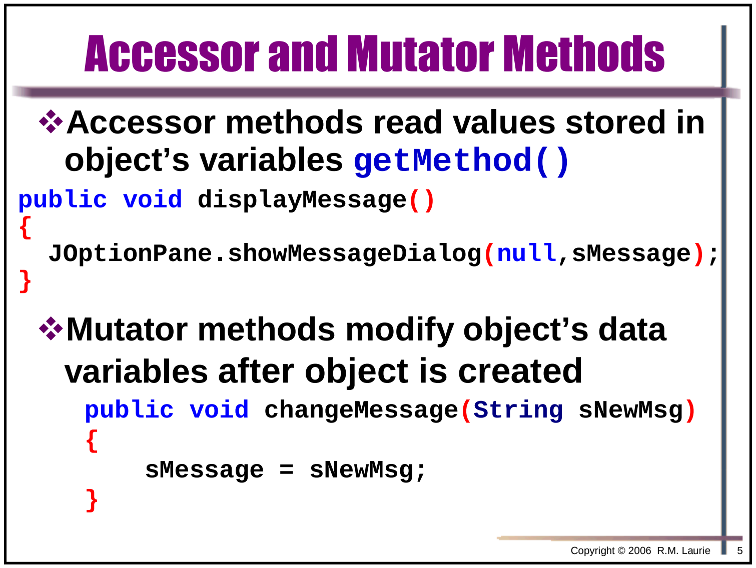
Instance variables should be `private` to prevent inappropriate changes.



Constructor Methods

- Method has same name as class
- Automatically called each time object created
- Purpose:
  - Initialize new object's instance variables

```
public class MethodExample1
{
  // Data declaration section
  private String sMessage;
  // Methods definition section
  public MethodExample1()
  {
    sMessage = "I like Java";
  }
```



Accessor and Mutator Methods

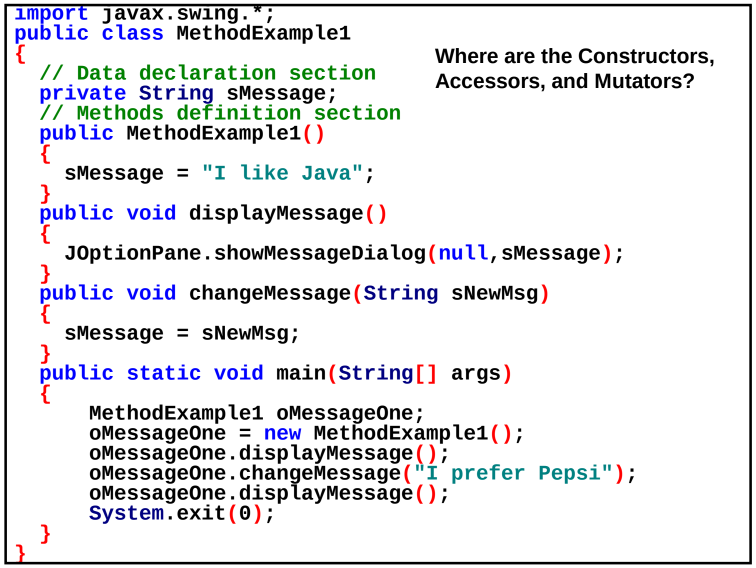
- Accessor methods read values stored in object’s variables `getMethod()`

```
public void displayMessage()
{
  JOptionPane.showMessageDialog(null,sMessage);
}
```

- Mutator methods modify object’s data variables after object is created

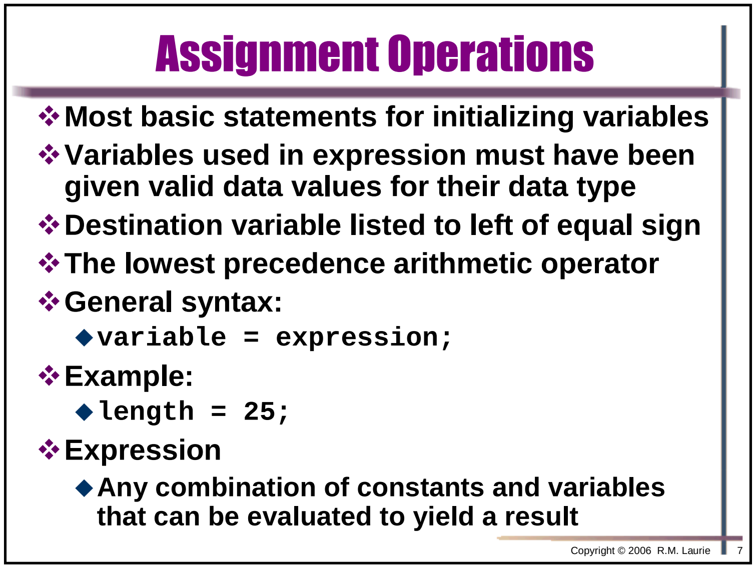
```
public void changeMessage(String sNewMsg)
{
     sMessage = sNewMsg;
}
```

Copyright © 2006 R.M. Laurie 5



```
import javax.swing.*;
public class MethodExample1
{
  // Data declaration section
  private String sMessage;
  // Methods definition section
  public MethodExample1()
  {
    sMessage = "I like Java";
  }
  public void displayMessage()
  {
    JOptionPane.showMessageDialog(null,sMessage);
  }
  public void changeMessage(String sNewMsg)
  {
    sMessage = sNewMsg;
  }
  public static void main(String[] args)
  {
      MethodExample1 oMessageOne;
      oMessageOne = new MethodExample1();
      oMessageOne.displayMessage();
      oMessageOne.changeMessage("I prefer Pepsi");
      oMessageOne.displayMessage();
      System.exit(0);
  }
}
```

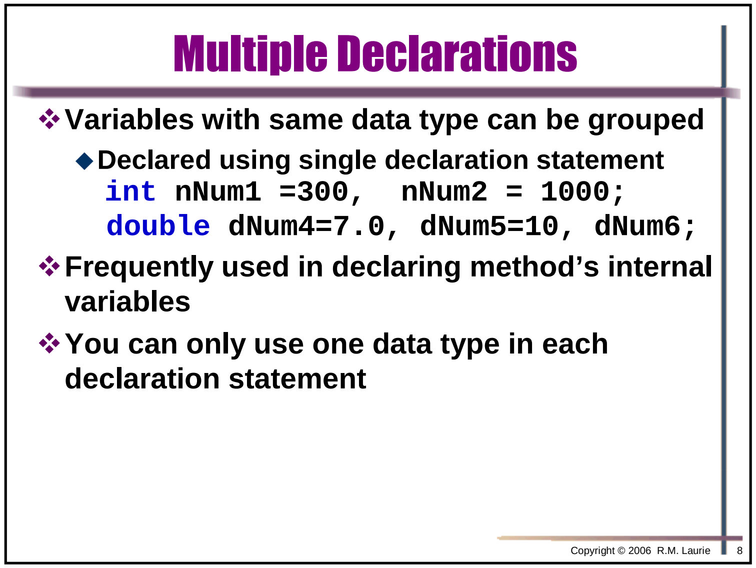
Where are the Constructors, Accessors, and Mutators?



Assignment Operations

- Most basic statements for initializing variables
- Variables used in expression must have been given valid data values for their data type
- Destination variable listed to left of equal sign
- The lowest precedence arithmetic operator
- General syntax:
  - `variable = expression;`
- Example:
  - `length = 25;`
- Expression
  - Any combination of constants and variables that can be evaluated to yield a result

Copyright © 2006 R.M. Laurie 7



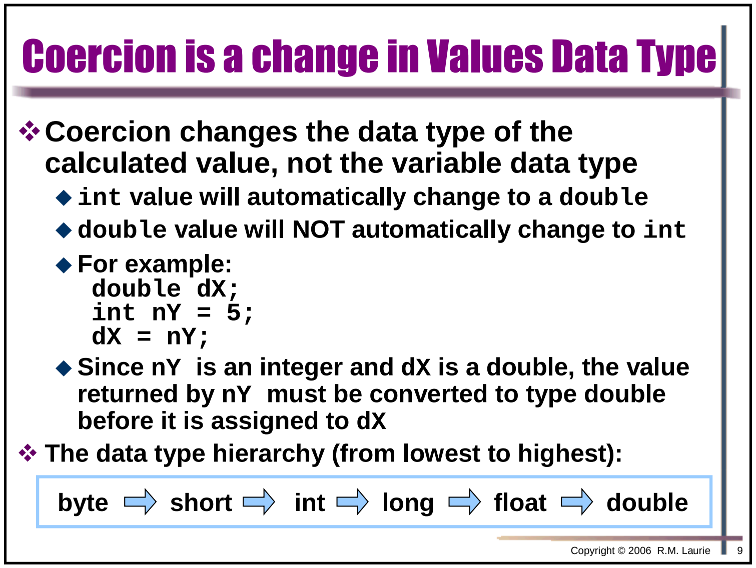
Multiple Declarations

- Variables with same data type can be grouped
  - Declared using single declaration statement

```
int nNum1 =300,  nNum2 = 1000;
double dNum4=7.0, dNum5=10, dNum6;
```

- Frequently used in declaring method’s internal variables
- You can only use one data type in each declaration statement

Copyright © 2006 R.M. Laurie 8



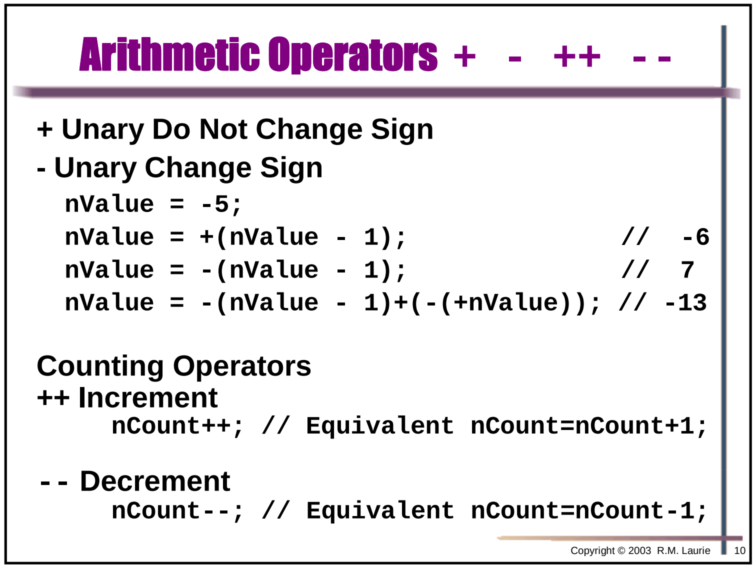
Coercion is a change in Values Data Type

- Coercion changes the data type of the calculated value, not the variable data type
  - `int` value will automatically change to a `double`
  - `double` value will NOT automatically change to `int`
  - For example:

```
double dX;
int nY = 5;
dX = nY;
```

  - Since nY is an integer and dX is a double, the value returned by nY must be converted to type double before it is assigned to dX
- The data type hierarchy (from lowest to highest):

byte ⇨ short ⇨ int ⇨ long ⇨ float ⇨ double



Arithmetic Operators + - ++ - -

+ Unary Do Not Change Sign

- Unary Change Sign

```
nValue = -5;
nValue = +(nValue - 1);            //  -6
nValue = -(nValue - 1);            //   7
nValue = -(nValue - 1)+(-(+nValue)); // -13
```

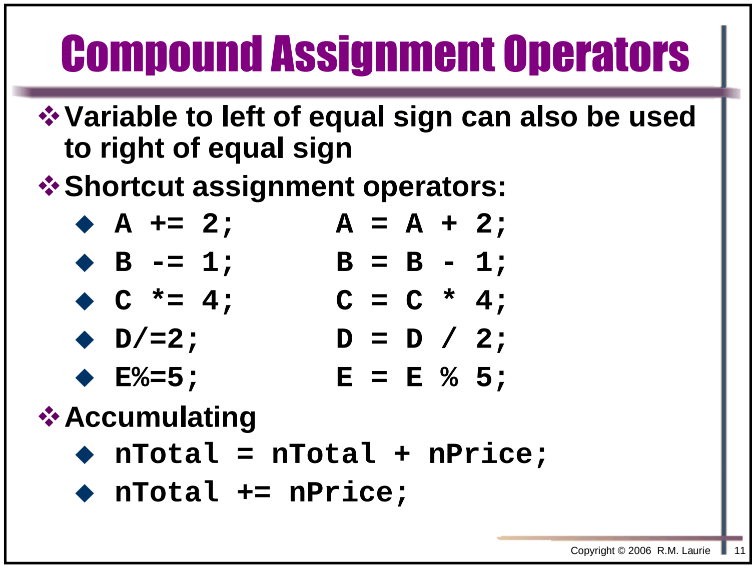
Counting Operators

++ Increment

```
    nCount++; // Equivalent nCount=nCount+1;
```

-- Decrement

```
    nCount--; // Equivalent nCount=nCount-1;
```



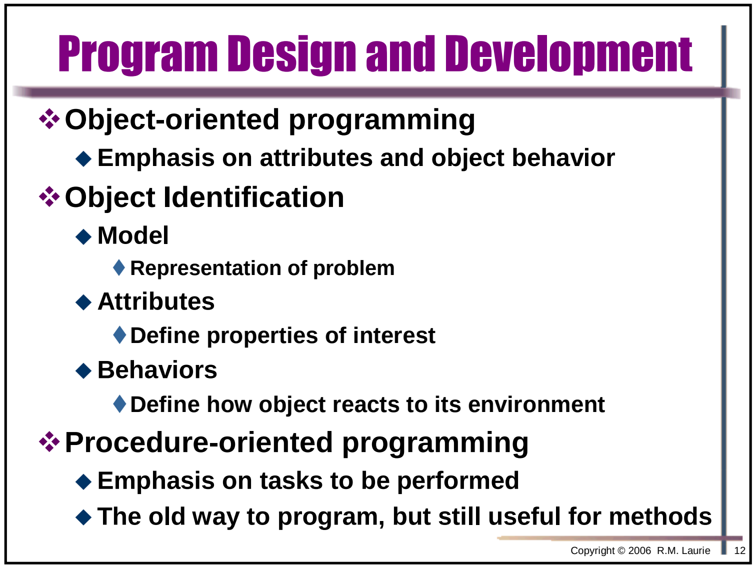
Compound Assignment Operators

- Variable to left of equal sign can also be used to right of equal sign
- Shortcut assignment operators:

```
A += 2;       A = A + 2;
B -= 1;       B = B - 1;
C *= 4;       C = C * 4;
D/=2;         D = D / 2;
E%=5;         E = E % 5;
```

- Accumulating

```
nTotal = nTotal + nPrice;
nTotal += nPrice;
```



Program Design and Development

- Object-oriented programming
  - Emphasis on attributes and object behavior
- Object Identification
  - Model
    - Representation of problem
  - Attributes
    - Define properties of interest
  - Behaviors
    - Define how object reacts to its environment
- Procedure-oriented programming
  - Emphasis on tasks to be performed
  - The old way to program, but still useful for methods