Another Example: An Object!

```
class Factorial
  { public int fact (int n)
    //=====================
      { int k, result=1;
        for (k=1; k<=n; k=k+1)
          { result = result * k;
          }
        return(result);
      } //fact
  } // class Factorial

public class Program3
  { public static void main (String args[])
    //=================================
      { Factorial f = new Factorial();
        System.out.println("Factorial of 7 = ”
                                + f.fact(7));
      } // main
  } // class Program3
```

Two Classes

```
class Factorial
  { ...
  } // class Factorial

public class Program3
  { ...
  } // class Program3
```

- Only one **`public`** class (filename: **`Program3.java`**)
- Good rule of thumb:
  - One class per file

Creating an Object

```
public class Program3
  { public static void main (String args[])
    //=====================================
      { Factorial f = new Factorial();
        System.out.println("Factorial of 7 = ”
                                + f.fact(7));
      } // main
  } // class Program3
```

Creating an Object (cont.)

```
Factorial f = new Factorial();
```

- **`f`** is called a *reference variable*
- **`new`** creates a new object (*instantiation*)

Reference Variables

- Normal variables
  - Hold a value
- Reference variables
  - Refer to an object

```
int x = 3;
Factorial f = new Factorial();
```

Using the Object

`f.fact(7)`

- Calls the **`fact`** method in the object referred to by **`f`**

Chapter 2
Types, Values, Variables and Expressions

Primitive Data Types

- Java has eight primitive data types:
  - **`int`**
  - **`double`**

  - **`byte, short, long`**
  - **`float`**
  - **`char`**
  - **`boolean`**

The `int` Type

- Same as **`Integer`** in Visual BASIC
- Signed integer values from –2 147 483 648 to 2 147 483 647 (32 bits)
- E.g.
  - `int k = 2000000000;`
  - `int m = -7;`

The **`double`** Type

- Same as **`Double`** in Visual BASIC
- Floating point values from –1.79769313486231570E+308 to 1.79769313486231570E+308

- E.g.
  - **`double d = 57.64;`**
  - **`double e = -7.23E-5;`**

Strings

- Are objects (*not* primitive data types)
- Similar to the **`String`** type in Visual BASIC
- Escape sequences: begin with backslash ****
- E.g.
  - **`String s = ”George”;`**
  - **`String t = ”He said \”Hi\”.”;`**
  - **`String f = ”C:\\Windows”;`**

Characters

- The **`char`** type

- E.g.
  - `char ch = 'a';`
  - `ch = '?';`
  - `ch = '\n';`

Boolean Values

- The **`boolean`** type
- Only two possible values:
  - **`true`**
  - **`false`**

- E.g.
  - **`boolean b = false;`**
  - **`b = (x <= 10);`**

Operators

- Arithmetic:
  - `+ - * /`
  - Mod (remainder): `%`

- E.g.
  - `x = 2 * a + b;`
  - `y = x % 3;`

Operators (cont.)

- Relational
  - **==** **!=**
  - **<** **<=** **>** **>=**

- Logical
  - And: **&&**
  - Or: **||**
  - Not: **!**

Operators (cont.)

- E.g.
  - `if (x != 0)...`
  - `if (ch >= ’a’ && ch <= ’z’)...`
  - `if (! s.equals(”George”))...`

Expression Compatibility

- Types may be mixed in expressions
- "Smaller" types are automatically *promoted* to "bigger" types

```
int k, j;
double d, e;
...
System.out.println(d + k); // OK
d = k; // OK
j = e; // Not OK!
```

Explicit Type Conversion

- How do we force the compiler to allow *demotion*?
- Use a *cast* (or *type-cast*)
  - **`(type)expression`**

```
int j;
double e;
...
j = (int)e; // OK
```