**Variable-length arguments**

Here is the **vaTest( )** method that takes a regular argument and a variable-length argument:

**class** apples {

// Here, msg is a normal parameter and v is a

// varargs parameter.

**static** **void** vaTest(String msg, **int** ... v) {

System.*out*.print(msg + v.length +" Contents: ");

**for**(**int** x : v)

System.*out*.print(x + " ");

System.*out*.println();

}

**public** **static** **void** main(String args[]){

*vaTest*("One vararg: ", 10);

*vaTest*("Three varargs: ", 1, 2, 3);

*vaTest*("No varargs: ");

}

}

**Result:**

One vararg: 1 Contents: 10

Three varargs: 3 Contents: 1 2 3

No varargs: 0 Contents:

**Overloading Vararg Methods**

You can overload a method that takes a variable-length argument. For example, the

following program overloads **vaTest( )** three times:

// Varargs and overloading.

**class** VarArgs3 {

**static** **void** vaTest(**int** ... v) {

System.*out*.print("vaTest(int ...): " + "Number of args: " + v.length +" Contents: ");

**for**(**int** x : v)

System.*out*.print(x + " ");

System.*out*.println();

}

**static** **void** vaTest(**boolean** ... v) {

System.*out*.print("vaTest(boolean ...) " + "Number of args: " + v.length + " Contents: ");

**for**(**boolean** x : v)

System.*out*.print(x + " ");

System.*out*.println();

}

**static** **void** vaTest(String msg, **int** ... v) {

System.*out*.print("vaTest(String, int ...): " + msg + v.length + " Contents: ");

**for**(**int** x : v)

System.*out*.print(x + " ");

System.*out*.println();

}

**public** **static** **void** main(String args[]){

*vaTest*(1, 2, 3);

*vaTest*("Testing: ", 10, 20);

*vaTest*(**true**, **false**, **false**);

}

}

**Result:**

vaTest(int ...): Number of args: 3 Contents: 1 2 3

vaTest(String, int ...): Testing: 2 Contents: 10 20

vaTest(boolean ...) Number of args: 3 Contents: true false false

But in some cases, it can result in ambiguity. For example, if vaTest(); was called in *main*, the computer wouldn’t know which overloaded method to call, which results in an error.

Another example would be:

static void vaTest(int ... v) { // ...

static void vaTest(int n, int ... v) { // ...

Although the parameter lists of **vaTest( )** differ, there is no way for the compiler to resolve the following call:

vaTest(1)