VARIABLE-LENGTH AND COMMAND LINE ARGUMENTS

Lecture 11

BITS Pilani, K.K.Birla Goa Campus

Prof. Anita Agrawal

Command-Line Arguments

- **Command line arguments** is a methodology in which user will give inputs through the console.
- A Java application can accept any number of arguments from the command line.
- Syntax: java class_name command_line_arguments
- JVM wraps the command-line arguments and supplies to args[ ].
  - Can be checked by using args.length

Example

…….cla3

- The space character separates command-line arguments. And hence each word is displayed on a new line by itself.

Variable Arguments (var…args)

- Varargs…(Variable Arguments): A feature in Java that simplifies the creation of methods that need to take a variable number of arguments.

- A method that takes a variable number of arguments is a varargs method.

- An argument that can accept variable number of values is a vararg.

- Syntax:
- A variable-length argument is specified by three periods(…).

Example:

```
public static void fun(int ... a)
 {
    // method body
  }
```

How varargs work behind the scene…

- The **...** syntax tells the Java compiler that the method can be called with zero or more arguments.
- void f1(int …nums)
- As a result, *nums* variable is implicitly declared as an array of type int[ ]. Thus, inside the method, *nums* variable is accessed using the array syntax.
- In case of no args, the length of nums is 0.

Example..

Other parameters

- The varargs method can have other parameters too in addition to varargs.
- But there can exist only one varargs parameter.
- The varargs parameter should be written last in the parameter list of method declaration.

Example:

- int sum(int x, float y, double … z)

  - The first two arguments are matched with the first two parameters and the remaining arguments belong to z

Erroneous varargs examples

- Specifying varargs as the middle parameter of method instead of last one:

int sum (int x, double … z, float y,)

- Specifying two varargs in a single method:

void method(string...abc, int... x)

- Specifying varargs as the first parameter of method instead of last one:

void method(int...x, String abc)

Example: Method using NoVaragas

```
class A {

    public int addnum(int a, int b){
        return a+b;
    }

    public int addnum(int a, int b, int c){
        return a+b+c;
    }

    public static void main( String[] args ) {
        A obj = new A();
        System.out.println(obj.addnum(1, 2));
        System.out.println(obj.addnum(1, 2, 3));
    }
}
```

Output:
3
6

- addnum() method had to be overloaded to make it work for 3 arguments.
- What if the user wants to add 5 numbers or 10 or 100?
- This can be handled in a neat way with the use of varargs.

- As you can see, varargs can be really useful in some situations. However, if you are certain about the number of arguments passed to a method, use method overloading instead. For example, if you are certain that sumNumber() method will be used only to calculate the sum of either 2 or 3 arguments, use overloading like in the first example.

Varargs method overloading

- Similar to typical methods, we can overload vararg methods too.

Varargs method… Important Points

- While defining method signature, always keep varargs at the last
- The variable argument must be the last argument passed to the method. Let's consider, you invoked display() method like this:

- display(1, 2, 3, 4);

- And, your display() method is defined as:

- public void display(int ... q, int p){
- // method body
- }

- In this case, compiler cannot figure out the number of arguments passed to q.

- However, if you define your method as:

- public void display(int p, int ... q) {
- // method body
- }

- The Java compiler assigns the first argument to p, and the remaining int arguments are assigned to q.

Ambiguity in varargs method overloading

```
class A {

    static void test(int ... vargs) {
        // method body
    }

    static void test(int n, int ... vargs) {
        // method body
    }
}
```

- In the above program, the compiler gets confused if you try to invoke the test() method even though test() methods are overloaded and accepts different number of arguments.

- The compiler doesn’t know which method to call. The compiler may think, you are trying to call test(int ... vargs) with one varargs argument. Also, the compiler may think, you are trying to call test(int n, int ... vargs) with argument passed to the first parameter with empty second parameter.

- Since there are two possibilities, it causes ambiguity. Because of this, sometimes you may need to use two different method names instead of overloading varargs method.