Var args

Methods with Variable Argument Lists (var-args)

As of 5.0, Java allows you to create methods that can take a variable number of arguments. Depending on where you look, you might hear this capability referred to as "variable-length argument lists," "variable arguments," "var-args," "varargs," or our personal favorite (from the department of obfuscation), "variable arity parameter." They're all the same thing, and we'll use the term "var-args" from here on out.
As a bit of background, we'd like to clarify how we're going to use the terms "argument" and "parameter" throughout this book.

■ arguments The things you specify between the parentheses when you're invoking a method:

```
doStuff("a", 2); // invoking doStuff, so a & 2 are arguments
```

■ parameters The things in the method's signature that indicate what the method must receive when it's invoked

```
void doStuff(String s, int a) { } // we're expecting two
 // parameters: String and int
```

We'll cover using var-arg methods more in the next few chapters, for now let's review the declaration rules for var-args:

■ Var-arg type When you declare a var-arg parameter, you must specify the type of the argument(s) this parameter of your method can receive. (This can be a primitive type or an object type.)
■ Basic syntax To declare a method using a var-arg parameter, you follow the type with an ellipsis (...), a space, and then the name of the array that will hold the parameters received.
■ Other parameters It's legal to have other parameters in a method that uses a var-arg.
■ Var-args limits The var-arg must be the last parameter in the method's signature, and you can have only one var-arg in a method.
Let's look at some legal and illegal var-arg declarations:
Legal:

```
void doStuff(int... x) { } // expects from 0 to many ints
 // as parameters
void doStuff2(char c, int... x) { } // expects first a char,
 // then 0 to many ints
void doStuff3(Animal... animal) { } // 0 to many Animals
```

Illegal:

```
void doStuff4(int x...) { } // bad syntax
void doStuff5(int... x, char... y) { } // too many var-args
```

```
void doStuff6(String... s, byte b) { } // var-arg must be last



public class VarArgsExample {

    public static void main(String[] args) {
        printNumbers(1, 2, 3);       // Passing three arguments
        printNumbers(4, 5, 6, 7);    // Passing four arguments
        printNumbers(8);             // Passing a single argument
        printNumbers();              // Passing no arguments
    }

    public static void printNumbers(int... numbers) {
        System.out.println("The numbers are:");
        for (int number : numbers) {
            System.out.println(number);
        }
    }
}
```

In the above example, we have a method called printNumbers that accepts a variable number of integer arguments using varargs (int... numbers). Inside the method, we can treat the numbers parameter as an array.

When calling the printNumbers method, you can pass any number of integer values as arguments, including zero arguments. The varargs parameter allows you to access all the passed values within the method.

Note that if you have multiple parameters in a method, the varargs parameter should be the last one. For example, public static void exampleMethod(int fixedParam, String... varargsParam) { ... }.

Using varargs can simplify the method signature and provide flexibility when dealing with varying numbers of arguments.

```java
package com.hdfc.var.args;


//VarArgsTest.main()
//VarArgsTest.Helper.main()
//className.staticMethodName()
//top level class cannot be static
public class VarArgsTest {

    public static int a;
    public int b;

    public static void staticMethod(){
```

```
        System.out.println(a);//VarArgsTest.a
       // System.out.println(this.b);//not a static

        Helper.main();
        //Helper.test();
    }

    public VarArgsTest(){
        System.out.println(a);//VarArgsTest.a
        System.out.println(b);
    }


    public  enum MyEnum{

    }

    public  interface MyInterface{

    }

    public  class Helper2{

    }

    public static class Helper{

         void test(){
            int a1 = VarArgsTest.a;
            //int b1 = VarArgsTest.b;
        }

        //compiler javac Hello.java
        //run java Hello "first" "1"  "9.9"
        public static void main(String... args) {

            Helper object= new Helper();
            object.test();

            VarArgsTest.doStuff("hello", 1);
            VarArgsTest.sum(1);
            VarArgsTest.sum(1, 2, 3);
            VarArgsTest.sum(1, 2, 3, 4);
            VarArgsTest.sum(1, 2, 3,4,5);

            VarArgsTest.multiply();
            VarArgsTest.multiply("one");
            VarArgsTest.multiply("one", "two");
        }
    }



    public static void doStuff(String a, int b){
        //
    }

    //one method can have only one var args
    //three dots ...
    //int... b -> 0 or more
    //int  b... is not allowed
    //Object... c  can be used with any object type
    //var-arg should be last parameter

    public static void multiply(String... b){
        //
    }

    public static void sum(int a, int... b){
        //
```

 Contact: 8087883669

```
    }
}
```

```
public enum Color {
    RED,
    WHITE
}
package com.hdfc.var.args;

public class TestEnum {

    enum MONTH {JAN, FEB}

    public static void main(String... parameter) {
        // class Helper3{}
        // Helper3 o = new Helper3();

        enum Traffic {RED, YELLOW, GREEN}
        ;
        System.out.println(Traffic.values());

        Color[] values = Color.values();

        Color black = Color.valueOf("WHITE");//using String get the enum constant
        System.out.println(black);

        int ordinal = Color.WHITE.ordinal();//index number
        String name = Color.WHITE.name();//index number
```

 Contact: 8087883669

```
        }
}
```

```
package com.hdfc.var.args;

public class Test {

    public static void main(String[] args) {
        VarArgsTest o = new VarArgsTest();
        o.staticMethod(); //no compilation error when accessing
static method on object
        VarArgsTest.staticMethod(); //this is correct way to access
static method


        //Helper helper = new Helper(); cannot access directly

        VarArgsTest outer = new VarArgsTest();
        VarArgsTest.Helper inner = new VarArgsTest.Helper();
        VarArgsTest.Helper inner2 = new VarArgsTest.Helper();
        inner.test(); //non-static method required object of Helper
class
        inner.main(); //no need of object

        //VarArgsTest.Helper.test();
        VarArgsTest.Helper.main();///static, directly accessing us-
ing className.staticMEthod

        //String a = "abc";
        //String a2 = "abc";
    }
}
```