Session-12

Date 30/07/2023

Var-arg methods

(Variable number of arguments methods)

1.5 Version In Java Came.

Untill 1.4 version we can't declare a method with Variable no. of arguments. if there is a change in no. of arguments compulsory we should go for new method, it increases the length of the code and reduces readibility

To Over Come this problem, some people introduced Var-arg method in 1.5 version

According to this we can declare a method which can take Variable no. of arguments, such type of methods are called Var-arg methods

```
m1(int ... x)
```

We can call this method by passing any no. of int values including "0" number

```
m1();
m1(10);
m1(10,20);
m1(10,20,30,40);
```

Sample

Date

```
public class VarArgsExample {

    public static int Sum (int... numbers) {
        int totalSum = 0;
        for (int num : numbers) {
            totalSum += num;
        }
        return totalSum;
    }

    public static void main (String[] args) {

        int Sum1 = Sum();
        int Sum2 = Sum(1,2,3);
        int Sum3 = Sum(10,20,30,40,50);

        System.Out.println("Sum1:" + Sum1);
        System.Out.println("Sum2:" + Sum2);
        System.Out.println("Sum3:" + Sum3);

    }
}
```

Spiral

class Test

Date

```
ps v m1(int... x)   -> Here we can call this method by passing sometimes 0 int numbers
{                      sometime 1 int number
  Sop.ln("Var-arg method");   sometime 2 int number.
}                             sometime 3 int numbers.
ps v main(String[] args)
{ m1();
  m1(10);
  m1(10,20);
  m1(10,20,30,40);
}
```

Output:
Var-arg method
Var-arg method
Var-arg method
Var-arg method

m1(int... x) → int[] x

by using Index we will differentiate value because var-arg method internally converted to an array.

```
class Test
{ public static void m1(int... x)
  { System.out.println("The number of arguments:" + x.length);
  }
public static void main(String[] args)
  { m1();
    m2(10);
    m3(10,20,30);
    m4(10,20,30,40);
  }
}
```

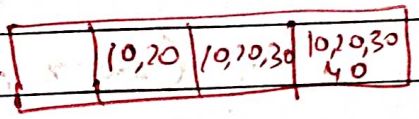
Spiral

Output

Date

The number of arguments: 0
The number of arguments: 1
The number of arguments: 2
The number of arguments: 4

∴ Internally Var-arg parameter Converted into 1-Dimensional array, Hence within the ~~Varga~~ Var-arg method, we Can differentiate Value by using index




```
class Test
{ public static void main (String[] args)
  { Sum ( );
    Sum (10,20)
    Sum (10,20,30)
    Sum (10,20,30,40)
  }
public static void Sum (int... x)
{ int total = 0;
  for (int x1 : x)
  { total = total + x1;
  }
Syso(n ("The sum is :" + total)
}
```

Method with no arguments. Since it's a Varrarg method, it Compe Called with zero or more, when Called with no argument 'x' will bcam Empty array and the Sum will be zero '0'

The Sum 0;
The Sum : 30
The Sum : 60
The Sum : 100

// for(int i=0; i<x.length; i
// { total = total + x[i
// }

Spiral

Which of the following valid var-arg declarations ?

m1 (int ... x) ✓

m1 (int ...x) ✓

m1 (int...x) ✓

m1 (int x...) ✗

m1 (int ...x) ✗

m1 (int ..x) ✗

← after int should be there
← spaces between ... are not allowed

m1 (int x, int ... y) ✓

m1 (String s, double ... y) ✓

m1 (double ...d, String s) ✗

The last parameter should be Var-arg

m1 (char ch, String ... s) ✓

m1 (int ... x, double ... d) ✗

Inside Var-arg method we can take only 1 Var-arg parameter and we can't take more the 1 Var-arg parameter

class Test {

psvm m1 (int ... x) ⇒ m1 (int [])

{

sop ln ("int ...");

}

psvm m1 (int [] x) ⇒ m1 (int [])

{

sop ln ("int []");

}

}

Compile error :- Cannot declare both m1 (int []) and m1 (int ...) in Test

Spiral

Inside a class we can't declare Var-arg method and Corresponding 1 dimensional array method simultaneously otherwise we will get compile time error

```
class Test
{
public static void m1(int... x)
  {
  sysoptln("Var-arg method");
  }
public static void m1(int x)
  {
  soptln("General method");
  }
P.S.V main(String[] args)
  {
  m1();
  }
}
```

Var arg method

Normal method

Output: Var-arg method will Called

normal method is expecting 1 int Value, are we providing Here, NO only m1() with no int parameter

Hence Normal method won't be called

m1(10,20); Var-arg method

m1(10); General method

Var-arg method have least priority in Java.

Spiral

Equivalence between Var-arg parameter and 1 dimensional array

Date

m1(int[] x)	m1(int... x)
m1(new int[]{10});	m1(int[] x) ⟹ m1(int... x)
m1(new int[]{10,20});	
m1(new int[]{10,20,30});	
	m1();
	m1(10);
	m1(10,20,30);

Case 1: Where 1-dimensional array present we can replace with Var-arg parameter.

✓	m1(int[] x) ⟹ m1(int... x)
✓	main(String[] args) ⟹ main(String... args)

Valid replacement

Case-2 Where ever Var-arg parameter present we can't replace with 1 dimensional array

m1(int... x) ⟹ m1(int[] x)

invalid Replacement

m1(int... x)	m1(int[] x)
m1();	→ ✗
m1(10);	→ ✗
m1(10,20,30);	→ ✗
m1(new int[]{10,20});	→ ✓

Sajal

Case 3 Date

m1(int... x) ⇒ int[] x

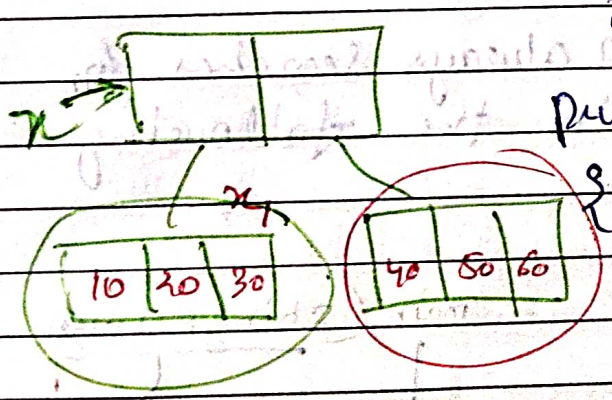
We can call this method by passing a group of int value and x will become 1-dimensional array

m1(int[]... x) ⇒ int[][] x

We can call this method by passing a group of 1-dimensional int arrays and x will become 2-dimensional int array

Example

```
class Test
{
    public static void (String[] args)
    { int[] a = {10,20,30};
      int[] b = {40,50,60};
      m1(a,b);
    }
    public static void m1(int[]... x)
    {
      for(int[] x1 : x)
      { System.out.println(x1[0]);
      }
    }
}
```



Output

10
40

Spiral

Command line argument

main() method

Date

Part - 1

Conclusion: 1

Whether Class contains main() method or not and whether main() method is declared according to requirement or not, these things won't be checked by compiler at runtime J.V.M is responsible to check these things.

if J.V.M Unable to find main() method then we will get run time exception saying : No such method Error : main.

```
Class Test
{
}
```

Compiler → NO ISSUE

R.E : No such method Error : main

Conclusion: 2 : At run time JVM always searches for the main() method with the following prototype

```
public static void main(String[] args)
```

- public → To Call by JVM from anywhere
- static → without existing object also JVM has to call this method
- main → This is the name which is configured inside JVM
- void → main() method won't return anything to JAVA
- String[] args → command line arguments

Spira

Date

public static void main (String[] args)

The above syntax is very strict and if we perform any change, then we will get runtime Exception saying. No such method Error: main

→ Various allowed changes are acceptable

① ~~Even though~~ instead of public static we can take static public that is order of modifiers is not important

② We can declare String array in (String[] args) in any acceptable form:

main (String[] args)
main (String [] args)
main (String args[])

③ instead of args we can take any valid java identifier

main (String[] durga)

④ We can replace String array with Var-arg parameter

main (String ... args)

Date

5. main() method can be declared with the following modifiers
   - final, synchronised, strictfp

```
Class Test
{
    static final synchronised strictfp void
    main (String... dog)
    {
        Sopln ("Valid main method")
    }
}
```

Output:
Valid main method

1. 'Static': the main method is marked as static because it needs to be called without creating an instance of the class. The main() is the entry point of the Java program, and it is called by the JVM before any objects are created

2. 'Synchronised': used to make main() method synchronised This means that only one thread at a time can execute the main() method. In most cases this is unnecessary for the 'main' method since it is typically executed in single thread, but adding 'synchronised' ensures thread safety. If the method is called from multiple threads.

3. final: The 'final' keyword means that the 'main' method cannot be overridden by any subclass, since the main method is the entry point of the program and needs to be fixed

and standard across all classes, Date making it 'final' ensures that its behaviour remains consistent.

5. Strictfp: Used to ensure that all floating point calculation with in the method are perform in strict adherence to the IEEE-754 standard for floating point airthmetic; this ensures consistent results across different platforms and JVM implementation when working with floating-point numbers.

6. Void: the 'main'() method does not return any value. it simply serve as the starting point of the program.

7. 'String... args': This uses var-arg ~~method~~ Syntax allowing you to pass a variable number of command line argument passed to this program through this Array.

Q:/ Which of the following main() declaration are valid?

Public static Void(String args) ×
public Static Void main(String args) ×
Public Static Void Main(String[] args) ×
Public Static int main(String[] args) ×
~~Public~~ static misstic ×
final synchronized Strictfp Public Void main(String[] args
final synchronised Strictfp Public static void main(String[] args)
Public Static Void main(String... args)

Spiral

Which of the above case we get Compile time error

Date

We won't get Compile time error anywhere but except last two cases in remaining we will run time exception.

NoSuchMethodError : main

Case 8:

Overloading of the Main method is possible but J.V.M will always call (String[] args) the overloaded method we have to call explicitly like normal method call

```
class Test
{
    P S V main (String[] args)
    {
        System.out.println("String[]");
    }
    P S V main (int[] args)
    {
        System.out.println("int[]");
    }
}
```

Overloaded methods

Case II

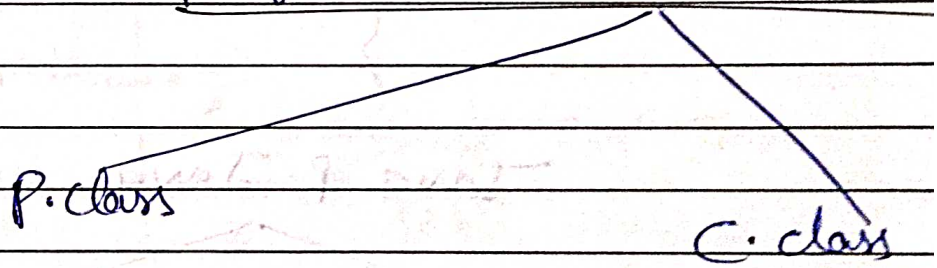
Date

```
class P
{
    public static void main (String[] args)
    {
        System.Out.println ("Parent main");
    }
}
class C extends P
{

}
```

Inheritance

Program Saved: P.Java



Java P ↵

O/P: Parent main

Java C ↵

O/P: Parent main

Inheritance concept applicable for main() method, Hence if a executing child class, if a child doesn't contain main() method, then parent class main() method will be executed.

Spiral

Date

```
class P
{
    psv main (String[] args)
    {
        sopln ("parent main");
    }
}
class C extends P
{
    psv main (String... args)
    {
        sopln (child name
    }
}
```

It is method hiding not overriding

Java P.Java

P.class

output// parent main

C.class

child main //output

it Seems overriding concept are applicable for main method beet it is not overriding and of it is method hiding

Note: in main() method overloading & Inheritance applicable but not overriding instead of it method hiding applicable.

Spiral

Date

Command line arguments
main() method
Part II

class Test
{
}

1.7 version Enhancement w.r.t main()

1.6 V	1.7V
Javac Test.java ↵	Javac Test.java ↵
	Java test ↵
R.E: NoSuchMethodError: main	Error: Main method not found in Class Test, please declared main method as public static void main (String[]

class Test
{
static
{
sop ln ("static block")
}
}

Same

Javac Test.java ↵

Java Test ↵

Error: Main Method not found in Class Test.

Java Test.java ✓

Java Test ↵

o/p Static Block:

R.E NoSuchMethodError: main

Spiral

Date

1.6 V

```
Class Test
{ static
  { sopln("static Block");
    System.exit(0);
  }
}
```

Javac Test.Java

Java Test ↵

o/p: static block

As the System.exit(0) Terminate the J.V.M Hence J.V.M won't search main method further.

1.7 V

```
Class Test
{ static
  { sopln("static Block");
    System.exit(0);
  }
}
```

Javac Test.Java

Java Test ↵

Whether the Class Contains static method/block or not, if it doesn't contain main method execution won't happen J.V.M rise

Error: main method not found in class Test

```
Class Test
{
  Static
  { Soplns ("static block");
  }
  Pss V main (String[] args)
  { Sopln ("main method");
  }
}
```

Javac Test.Java

Java Test ↵

o/p: static block
main method

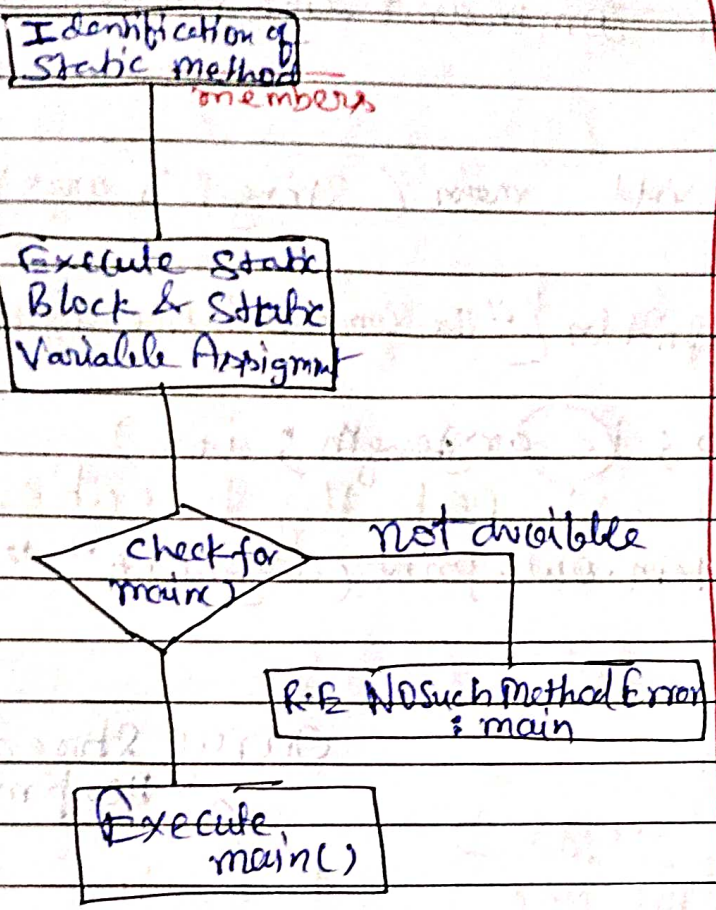
Javac Test.Java

Java Test ↵

o/p: static block
main method

Spiral

1.6 V



1.7 V

Command line argument

String[] args = {"A", "B", "C"}

Public static Void main(String[] args)

The arguments which are passing from Command prompt are called Command line arguments with these Command line argument J.V.M will create an array and by passing an array as argument, JVM will call main method

Java Test A B C

args[0] args[1] args[2]

args.length = (3)

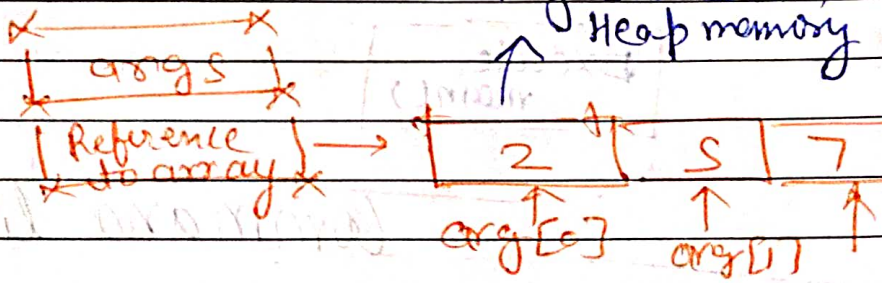
Spiral

program for getting the same output what we Enter in command line prompt arguments

Date

```
class Test
{
    public static void main (String[] args)
    {
        System.out.println("The command-line arguments are:");
        for(int i=0; i<args.length; i++)
        {
            System.out.print(args[i]+" ");
        }
    }
}
```

Perfectly Execution



arg.length = 3

program to Square via CmdA

```
class Test
{
    P S V main (String [] args)
    int n= Integer.parseInt(args[0])
    Sop ln ("the Square of "+n+" is "+(n*n))
}
}
```

Javac Test.d

↓

Java Test

16

Spira

The main objective of Command line argument is we can customize behaviour of the main() method.

// Command line argument is always of String type (String[] args)

Example-1

```
class Test
{
    psvm(String[] args)
    {
        for(int i=0; i<=args.length; i++)
        {
            sopln(args[i]);
        }
    }
}
```

R.E

Java Test A B C ↵

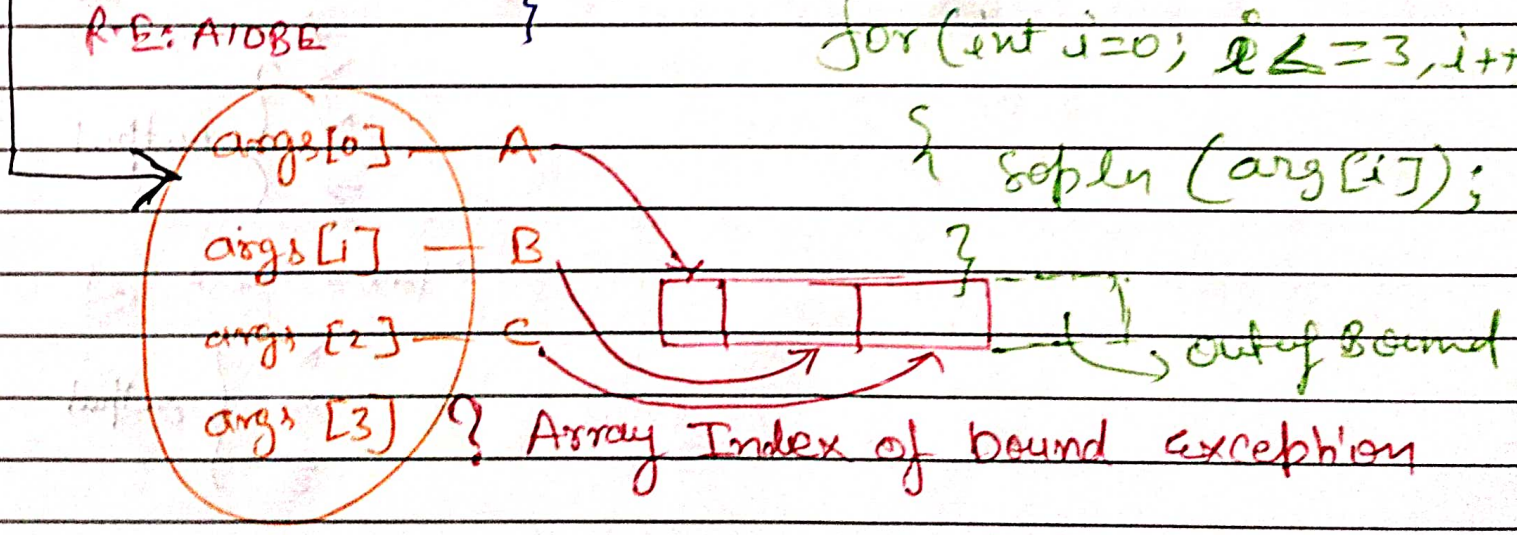
R.E : A1OBE

Java Test A B ↵

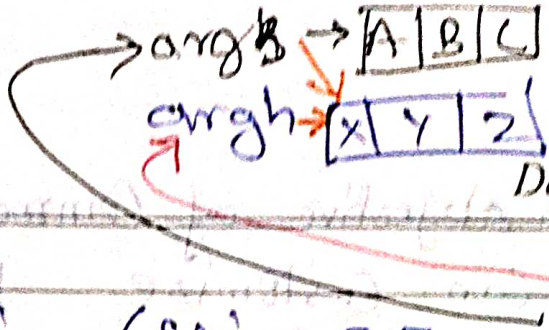
R.E : ArrayIndexOutBoundException

Java Test ↵

R.E : A1OBE



Example 2



Date

```
class Test
{
  p s v main (String [] args)
  {
  String [] argh = {"x", "y", "z",};
  args = argh;
  for (String s : args)
  {
    sopln(s);
  }
  }
}
```

Java Test ABC ↵

x
y
z
} output

Java Test AB ↵

x
y
z
} output

Java Test ↵

x
y
z
} output

Example 3: within main() method, command line argument are available in string form

Date

```
Class Test
{
  P s v main (String[] args)
    {
      Sopln (args[0] + args[1]); //1020
    }
}
```

Java 10 20 ↵

Example 4: Usually space itself is the seperator between command line argument. If our command line argument itself contains space, then we have to enclose that command in double Quotes " "

```
Class Test
{
  P static void main (String[] args)
  {
    Sopln (args[0]); //output:
  }
}
```

Note book

Java Test "Note book" ↵

if Java Test Note book ↵ //output: Note

args[0] args[1]