Parul® University

Object Oriented Programming with JAVA

Ms. Vaishalee Joishar, Cyber Security Trainer

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-4

Arrays

Parul®
University

DIGITAL LEARNING CONTENT

What we will learn

- ✔ Single Dimensional arrays
- ✔ Copying arrays
- ✔ Passing and returning array from method
- ✔ Searching and sorting arrays and the Array class
- ✔ Two-Dimensional array and its processing
- ✔ Passing Two-dimensional Array to methods
- ✔ Multidimensional Arrays

Array

Why we need Array?

- Very often we need to deal with relatively large set of data.
- E.g.-
  - Percentage of all the students of the college. (May be in thousands)
  - Age of all the citizens of the city. (May be lakhs)
- We need to declare thousands or lakhs of the variable to store the data which is practically not possible.
- We need a solution to store more data in a single variable.
- Array is the most appropriate way to handle such data.
- As per English Dictionary, “Array means collection or group or arrangement in a specific order.”

Cont...

- An array is a fixed size sequential collection of elements of same data type grouped under single variable name.

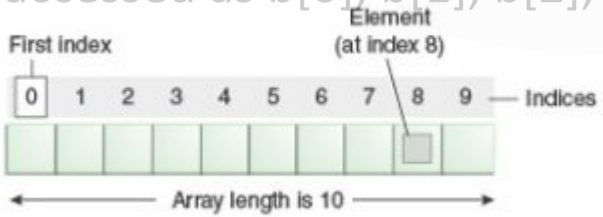
```
int percentage[10];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fixed Size	Sequential	Same Data type	Single Variable Name
The size of an array is fixed at the time of declaration which cannot be changed later on. Here **array size** is **10**.	All the elements of an array are stored in a consecutive blocks in a memory. **10** (0 to 9)	Data type of all the elements of an array is same which is defined at the time of declaration. Here **data type** is **int**	All the elements of an array will be referred through common name. Here **array name** is **percentage**

Array declaration

- Normal Variable Declaration: **int a;**
- Array Variable Declaration: **int b[10];**
- **Individual value** or data stored in an array is known as an e**lement of an array.**
- Positioning / **indexing** of an elements in an array always **starts with 0 not 1**.
  - If 10 elements in an array then index is 0 to 9
  - If 100 elements in an array then index is 0 to 99
  - If 35 elements in an array then index is 0 to 34
- Variable a stores 1 integer number where as variable b stores 10 integer numbers which can be accessed as b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7], b[8] and b[9]



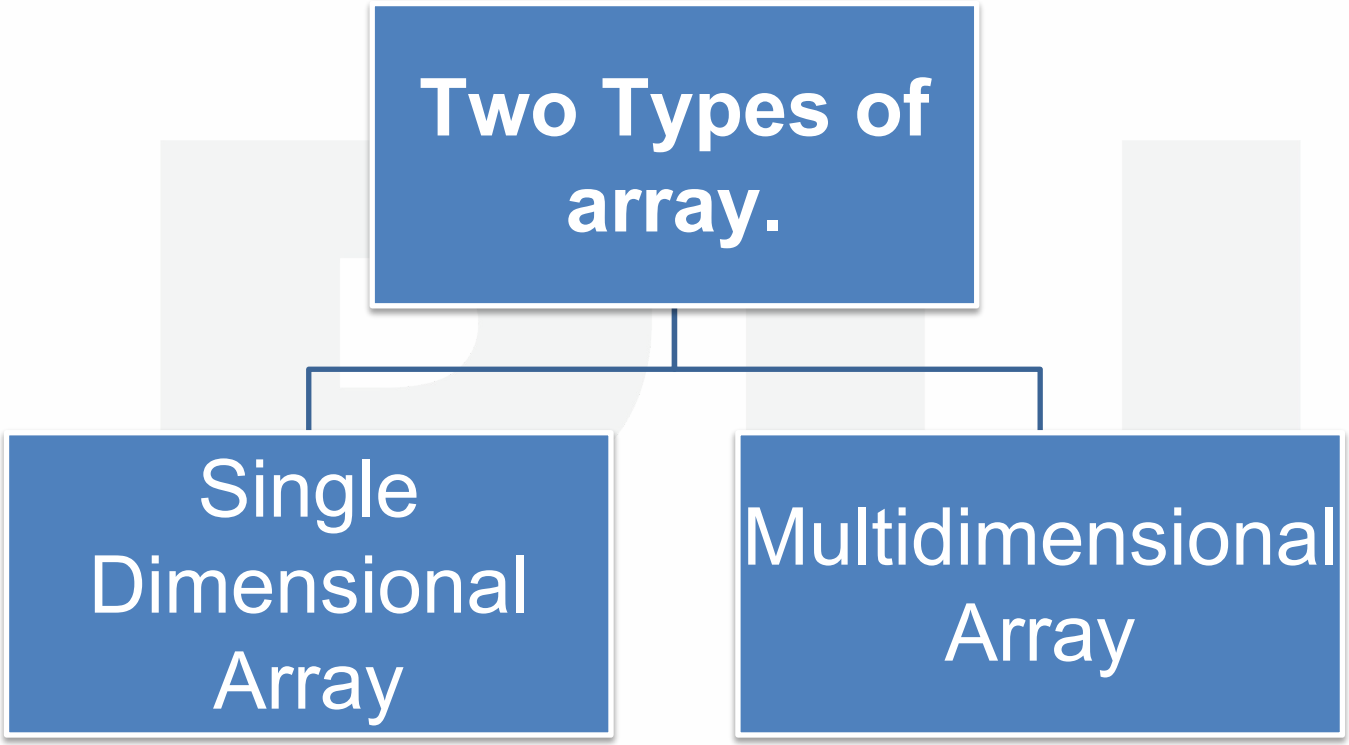
Cont...

Important point about Java array.

- An array is derived datatype.
- An array is dynamically allocated.
- The individual elements of an array is refereed by their index/subscript value.
- The subscript for an array always begins with 0.

35	13	28	106	35	42	5	83	97	14
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Types of Array



One-Dimensional Array

- An array using **one subscript** to represent the **list of elements** is called **one dimensional array.**
- A One-dimensional array is essentially a **list** of **like-typed variables.**
- **Array declaration:**

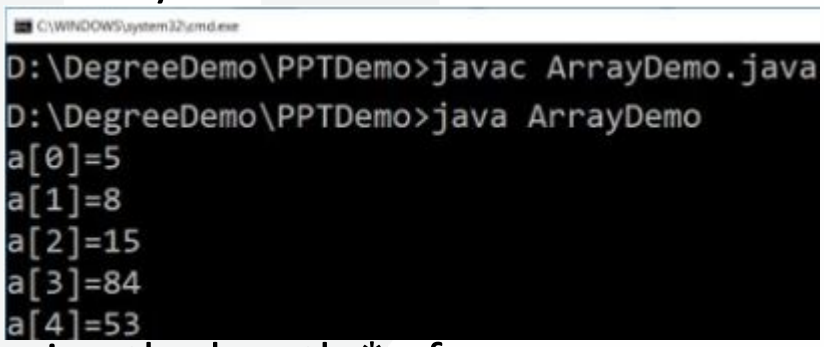
  type var-name[];
- **Example**:

  int student_marks[];
- Above example will represent array with no value (null).
- To link student_marks with actual array of integers, we must allocate one using new keyword.
- Example:

  int student_marks[] = new int[20];

Example (Array)

```
1.  public class ArrayDemo{
2.  public static void main(String[] args) {
3.  int a[]; // or int[] a
4.  // till now it is null as it does not assigned any memory
5.  a = new int[5]; // here we actually create an array
6.  a[0] = 5;
7.  a[1] = 8;
8.  a[2] = 15;
9.  a[3] = 84;
10. a[4] = 53;
11. /* in java we use length property to determine the length * of an array,
    unlike c where we used sizeof function */
12. for (int i = 0; i < a.length; i++) {
13.     System.out.println("a["+i+"]="+a[i]); }  } }
```



WAP to store 5 numbers in an array and print them

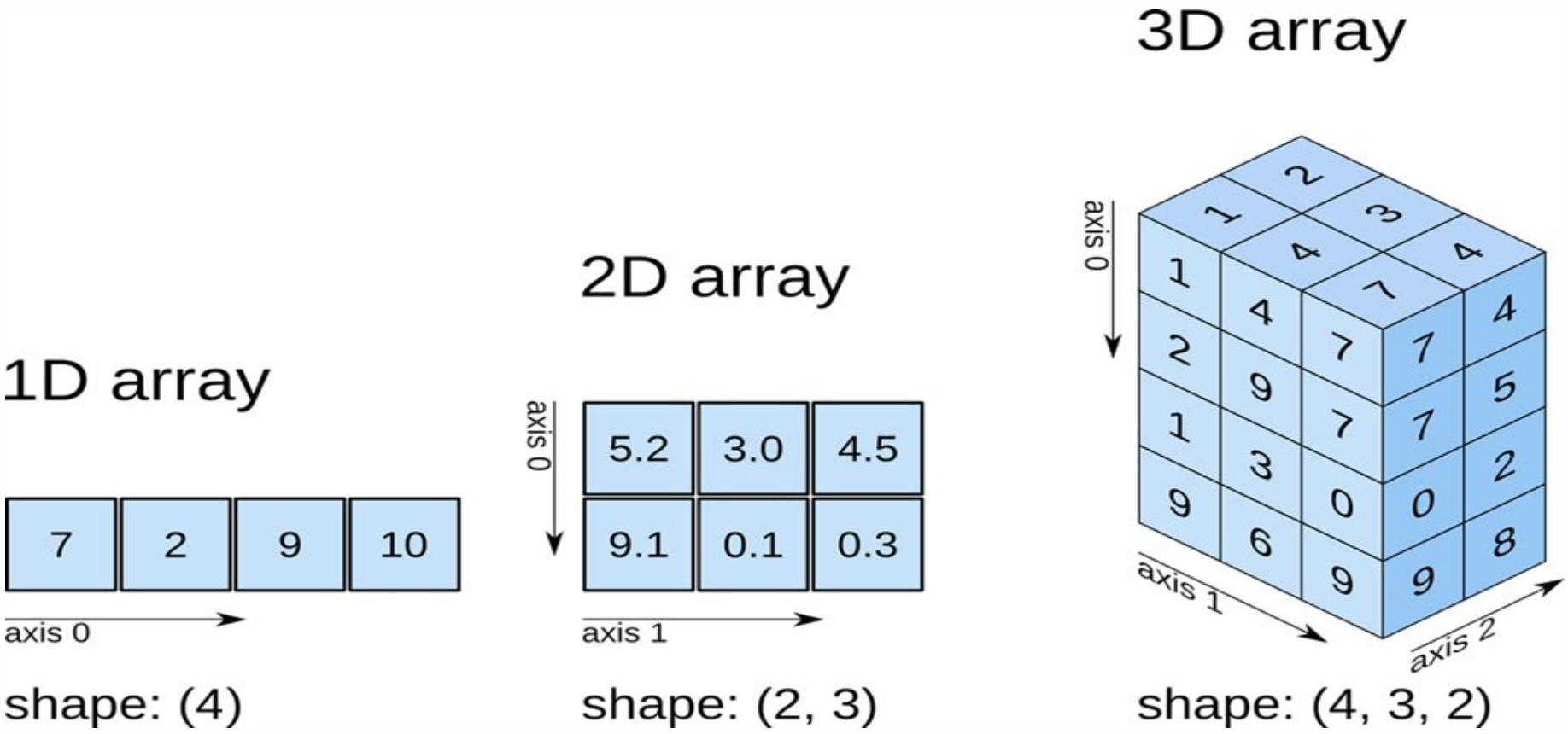
```
1.  import java.util.*;
2.  class ArrayDemo1{
3.  public static void main (String[] args){
4.  int i, n;
5.  int[] a=new int[5];
6.  Scanner sc = new Scanner(System.in);
7.  System.out.print("enter Array Length:");
8.  n = sc.nextInt();
9.  for(i=0; i<n; i++) {
10.     System.out.print("enter a["+i+"]:");
11.     a[i] = sc.nextInt();     }
12. for(i=0; i<n; i++)
13.     System.out.println(a[i]);
14.   }
15. }
```

```
enter Array
Length:5
enter a[0]:1
enter a[1]:2
enter a[2]:4
enter a[3]:5
enter a[4]:6
1
2
4
5
6
```

Exercise

1. WAP to print elements of an array in reverse order
2. WAP to count positive number, negative number and zero from an array of n size
3. WAP to count odd and even elements of an array.
4. WAP to calculate sum and average of n numbers from an array.
5. WAP to find largest and smallest from an array.

Multidimensional Array



WAP to read 3 x 3 elements in 2d array

- In java, multidimensional array is actually array of arrays.
- Example:

First Over (a[0])	4 a[0][0]	0 a[0][1]	1 a[0][2]	3 a[0][3]	6 a[0][4]	1 a[0][5]
Second Over (a[1])	1 a[1][0]	1 a[1][1]	0 a[1][2]	6 a[1][3]	0 a[1][4]	4 a[1][5]
Third Over (a[2])	2 a[2][0]	1 a[2][1]	1 a[2][2]	0 a[2][3]	1 a[2][4]	1 a[2][5]

- **length field:**
- If we use length field with multidimensional array, it will return length of first dimension.
- Here, if runPerOver.length is accessed it will return 3
- Also if runPerOver[0].length is accessed it will be 6

WAP to read 3 x 3 elements in 2d array

```
1.  import java.util.*;
2.  class Array2Demo{
3.  public static void main(String[] args) {
4.  int size;
5.  Scanner sc=new Scanner(System.in);
6.  System.out.print("Enter size of an array");
7.  size=sc.nextInt();
8.  int a[][]=new int[size][size];
9.  for(int i=0;i<a.length;i++){
10. for(int j=0;j<a.length;j++){
11.             a[i][j]=sc.nextInt();    }      }
12. for(int i=0;i<a.length;i++){
13. for(int j=0;j<a.length;j++){
14. System.out.print("a["+i+"]["+j+"]:"+a[i][j]+"\t"
15. System.out.println(); } } }
```

	Column-0	Column-1	Column-2
Row-0	11	18	-7
Row-1	25	100	0
Row-2	-4	50	88

Output:

```
11
12
13
14
15
16
17
18
19
a[0][0]:11      a[0][1]:12      a[0]
[2]:13
a[1][0]:14      a[1][1]:15      a[1]
```

Exercise

- WAP to perform addition of two 3 x 3 matrices
- WAP to perform multiplication of two 3 x 3 matrices

Initialization of an array elements

One dimensional Array

1. int a[5] = { 7, 3, -5, 0, 11 }; // a[0]=7, a[1] = 3, a[2] = -5, a[3] = 0, a[4] = 11
2. int a[5] = { 7, 3 }; // a[0] = 7, a[1] = 3, a[2], a[3] and a[4] are 0
3. int a[5] = { 0 }; // all elements of an array are initialized to 0

Two dimensional Array

1. int a[2][4] = { { 7, 3, -5, 10 }, { 11, 13, -15, 2} }; // 1st row is 7, 3, -5, 10 & 2nd row is 11, 13, -15, 2
2. int a[2][4] = { 7, 3, -5, 10, 11, 13, -15, 2 }; // 1st row is 7, 3, -5, 10 & 2nd row is 11, 13, -15, 2
3. int a[2][4] = { { 7, 3 }, { 11} }; // 1st row is 7, 3, 0, 0 & 2nd row is 11, 0, 0, 0
4. int a[2][4] = { 7, 3 }; // 1st row is 7, 3, 0, 0 & 2nd row is 0, 0, 0, 0
5. int a[2][4] = { 0 }; // 1st row is 0, 0, 0, 0 & 2nd row is 0, 0, 0, 0

Multi-Dimensional Array (Example)

```
1.  Scanner s = new Scanner(System.in);
2.  int runPerOver[][] = new int[3][6];
3.  for (int i = 0; i < 3; i++) {
4.  for (int j = 0; j < 6; j++) {
5.  System.out.print("Enter Run taken" + " in Over numner " + (i + 1) + " and Ball
    number " + (j + 1) + " = ");
6.  runPerOver[i][j] = s.nextInt();      }    }
7.  int totalRun = 0;
8.  for (int i = 0; i < 3; i++) {
9.  for (int j = 0; j < 6; j++) {
10. totalRun += runPerOver[i][j];   }    }
11. double average = totalRun / (double) runPerOver.length;
12. System.out.println("Total Run = " + totalRun);
13. System.out.println("Average per over = " + average);
```

Multi-Dimensional Array (Cont.)

```
14.  manually allocate different size:
15.  int runPerOver[][] = new int[3][];
16.  runPerOver[0] =  new int[6];
17.  runPerOver[1]  =  new int[7];
18.  runPerOver[2]  =  new int[6];

19.  initialization:
20.  int runPerOver[][] = {
21.                       {0,4,2,1,0,6},
22.                       {1,-1,4,1,2,4,0},
23.                       {6,4,1,0,2,2},
24.                 }
```

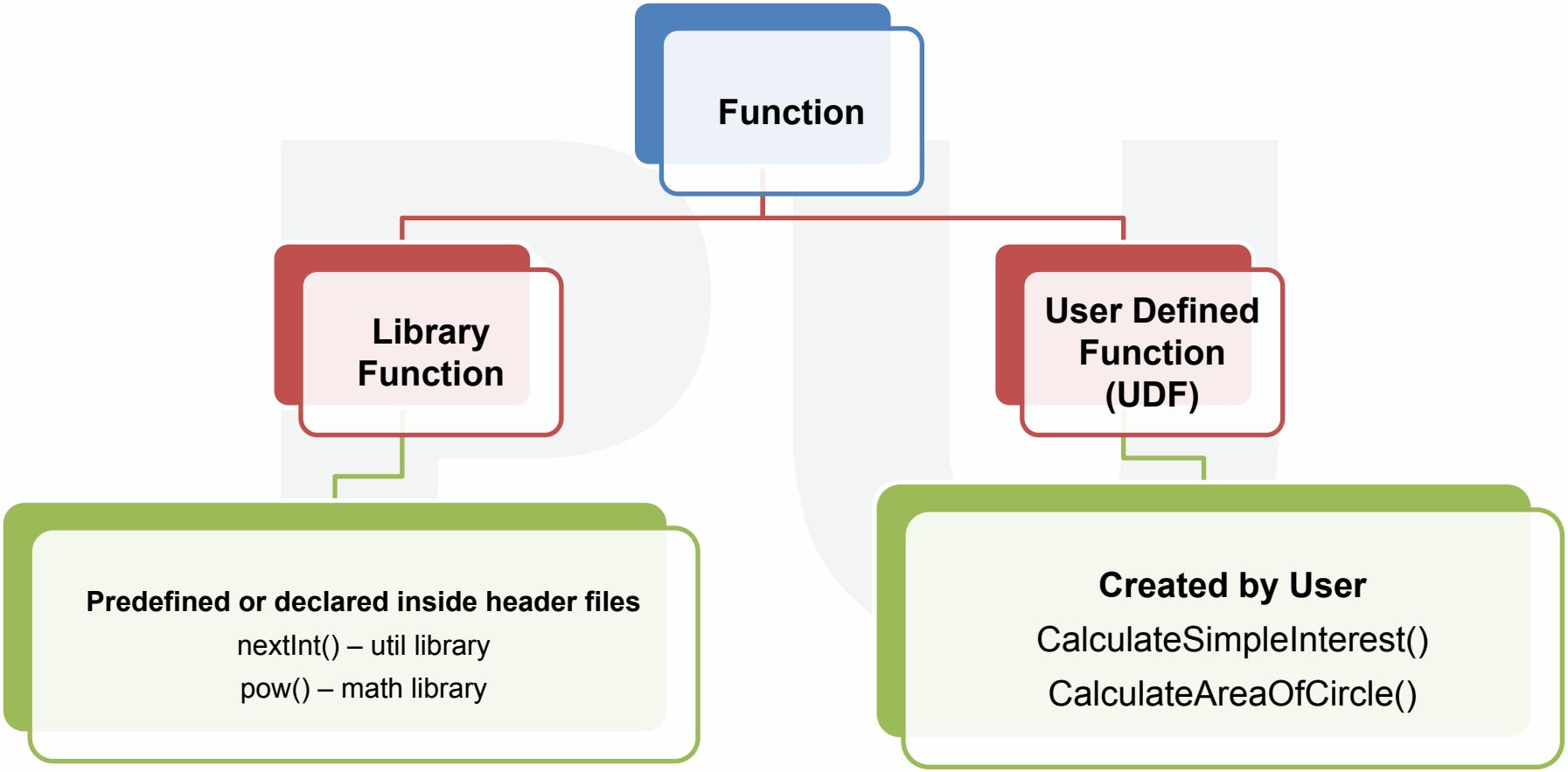
Note : here to specify extra runs (Wide, No Ball etc..) negative values are used

Methods

- A method is a group of statements that performs a specific task.
- A large program can be divided into the basic building blocks known as method/function.
- The function contains the set of programming statements enclosed by { }.
- Program execution in many programming language starts from the main function.
- main is also a method/function

```
void main()
{
    // body part
}
```

Types of Function



Program Structure of Function

- User-defined function's program structure is divided into three parts as follows:

C:Function Structure

```
void func1();


void main()
{
   ....
   func1();
}

void func1()
{
   ....
   //function body
   ....
}
```

Function Prototype

Function call

Function definition

Java:Function for addition of 2 numbers

```
class MethodDemo{
 public static void main(String[] ags){
        int a=10,b=20,c;
        c=add(a,b);
        System.out.println("a+b="+c);
 }
static int add(int i, int j){
        return i+j;
 }
}
```

Method Definition

- A method definition defines the method header and body.
- A method body part defines method logic.
- Method statements

Syntax

```
return-type method_name(datatyp1 arg1, datatype2 arg2, …)
{
        functions statements
}
```

Example

```
int addition(int a, int b);
{
        return a+b;
}
```

WAP to add two number using add(int, int) Function

```
1.  public class Main {
2.      public static void main(String[] args) {
3.          int num1 = 10;
4.          int num2 = 20;
5.          int sum = add(num1, num2);
6.          System.out.println("Sum: " + sum);
7.      }
8.      public static int add(int a, int b) {
9.          return a + b;
10.     }
11. }
12.
```

Output:

```
a+b=30
```

Actual parameters v/s Formal parameters

- Values that are passed from the calling functions are known **actual parameters**.
- The variables declared in the function prototype or definition are known as **formal parameters**.
- Name of formal parameters can be same or different from actual parameters.
- Sequence of parameter is **important**, not name.

Actual Parameters

```
int num1 = 10;
        int num2 = 20;
        int sum = add(num1, num2);
System.out.println("Sum: " +sum);

// num1 and num2 are the actual
parameters.
```

Formal Parameters

```
public static int add(int a, int b)
{
        return a + b;
}
        // a and b are the formal
parameters.
```

Return Statement

- The function can **return only one value**.
- Function cannot return more than one value.
- If function is not returning any value then return type should be **void**.

Actual Parameters

```
int a=10,b=20,c;
MethodDemo md=new
MethodDemo();
c=md.sub(a,b);
```

// a and b are the actual parameters.

Formal Parameters

```
int sub(int i, int j)
{
      return i – j;
}
```

// i and j are the formal parameters.

WAP to calculate the Power of a Number using method

```
1.  import java.util.*;
2.  public class PowerMethDemo1{
3.  public static void main(String[] args){
4.    int num, pow, res;
5.    Scanner sc=new Scanner(System.in);
6.    System.out.print("enter num:");
7.    num=sc.nextInt();
8.    System.out.print("enter pow:");
9.    pow=sc.nextInt();
10.   PowerMethDemo1 pmd=new
      PowerMethDemo1();
11.   res = pmd.power(num, pow);
12.   System.out.print("ans="+res);
13.   } //main()
14.   int power(int a, int b){
15.     int i, r = 1;
16.     for(i=1; i<=b; i++)
17.     {
18.       r = r * a;
19.     }
20.     return r;
21.   }//power()
22.  }//class
```

Output:

```
enter num:5
enter pow:3
ans=125
```

Types of Methods(Method Categories)

Function can be divided in 4 categories based on arguments and return value

1. Method without arguments and without return value

Ex: -void add();

2. Method without arguments and with return value

Ex:-int add();

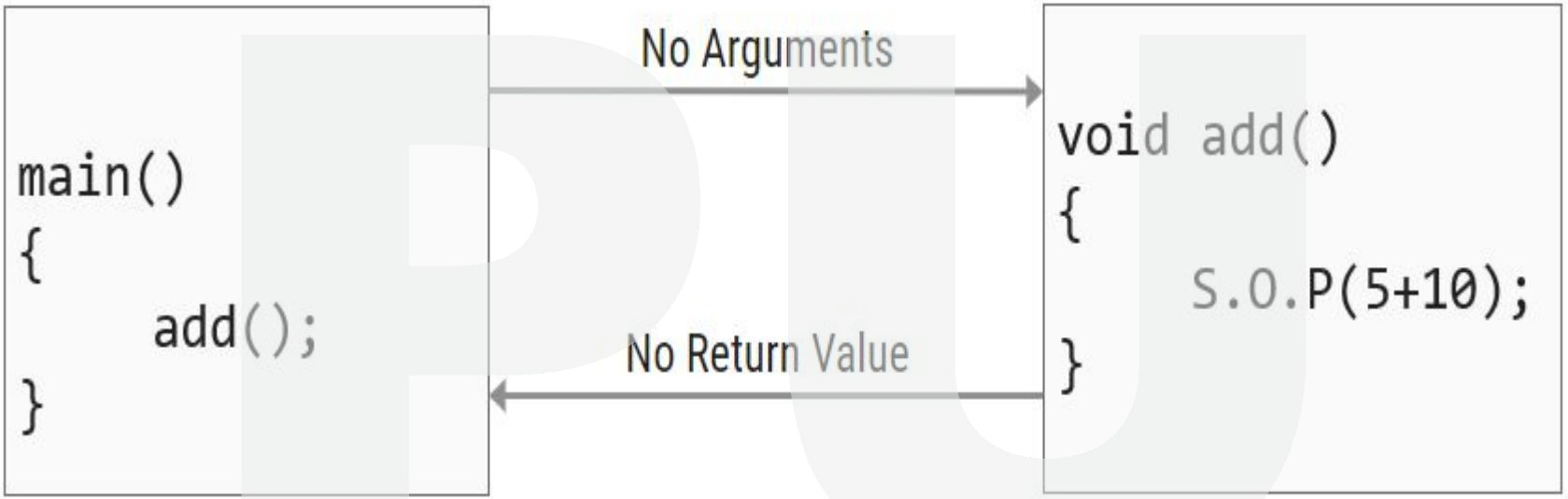
3. Method with arguments and without return value

Ex:-void add(int, int);

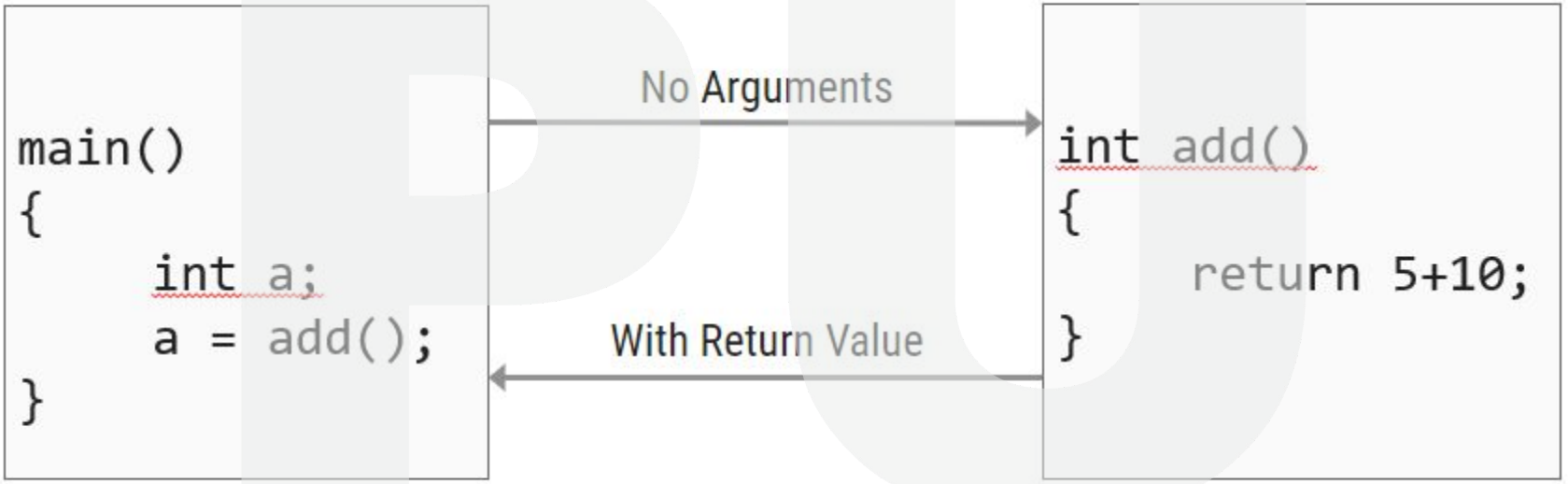
4. Method with arguments and with return value

Ex:-int add(int, int);

Method without arguments and without return value



Method without arguments and with return value



Method with arguments and without return value

```
main()
{
      int a=5,b=10;
      add(a,b);
}
```

With Arguments →

No Return Value ←

```
void add(int a, int b)
{
      S.O.P(a+b);
}
```

Method with arguments and with return value

```
main()
{
      int a=5,b=10,c;
      c=add(a,b);
}
```

With Arguments

With Return Value

```
int add(int a, int b)
{
      return a + b;
}
```

Method Overloading: Compile-time Polymorphism

- **Definition**: When two or more methods are implemented that share same name but different parameter(s), the methods are said to be **overloaded**, and the process is referred to as **method overloading**
- Method overloading is one of the ways that Java implements **polymorphism.**
- When an overloaded method is invoked, Java uses the **type and/or number of arguments** as its guide to determine which version of the overloaded method to actually call.

  E.g. public void **draw**()
  public void **draw**(int height, int width)
  public void **draw**(int radius)
- Thus, overloaded methods must differ in the type and/or number of their parameters.
- While in overloaded methods with different return types and same name & parameter are not allowed ,as the return type alone is insufficient for the compiler to distinguish two versions of a method.

Method Overloading: Compile-time Polymorphism

```
1.  class Addition{
2.  int i,j,k;
3.    void add(int a){
4.    i=a;
5.    System.out.println("add i="+i);
6.    }
7.    void add(int a,int b){\\overloaded add()
8.    i=a;
9.    j=b;
10.   System.out.println("add i+j="+(i+j));
11.   }
12.   void add(int a,int b,int c){\\overloaded add()
13.   i=a;
14.   j=b;
15.   k=c;
16.   System.out.println("add i+j+k="+(i+j+k));
18.   }
19.   }
20.   class OverloadDemo{
21.   public static void
      main(String[] args){
22.     Addition a1= new Addition();
        //call all versions of add()
23.     a1.add(20);
24.     a1.add(30,50);
25.     a1.add(10,30,60);
26.     }
27.   }
```

Output

```
add i=20
add i+j=80
add i+j+k=100
```

Method Overloading: Compile-time Polymorphism

```
1.  class Addition{
2.  int i,j,k;
3.    void add(int a){
4.    i=a;
5.    System.out.println("add i="+i);   }
6.    void add(int a,int b){\\overloaded add()
7.    i=a;
8.    j=b;
9.    System.out.println("add i+j="+(i+j));     }
10.   void add(double a, double b){\\overloaded
      add()
11.   System.out.println("add a+b="+(a+b));  }
12.   void add(int a,int b,int c){\\overloaded add()
13.   i=a;
14.   j=b;
15.   k=c;
16.   System.out.println("add
      i+j+k="+(i+j+k));
17.    }  }
18.  class OverloadDemo{
19.  public static void
      main(String[] args){
20.    Addition a1= new Addition();
21.    //call all versions of add()
22.    a1.add(20);
23.    a1.add(30,50);
24.    a1.add(10,30,60);
25.    a1.add(30.5,50.67);
26.    }
27.  }
```

Output

```
add i=20
add i+j=80
add i+j+k=100
```

Method Overloading: Points to remember

- Method overloading supports polymorphism because it is one way that Java implements the “one interface, multiple methods” paradigm.
- Overloading increases the readability of the program.
- There are two ways to overload the method in java
    1. By changing number of arguments
    2. By changing the data type
- In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

Method Overloading: Points to remember

- **Can we overload java main() method?**
  - Yes, by method overloading. We can have any number of main methods in a class by method overloading
  - But JVM calls main() method which receives string array as arguments only.

Advantages of Method

- Rewriting the same logic or code again and again in a program can be avoided.
- Same function can be call from multiple times without rewriting code.
- Instead of writing many lines, just function need to be called.
- Instead of changing code multiple times, code in a function need to be changed.
- Large program can be easily understood or traced when it is divide into functions.
- Testing and debugging of code for errors can be done easily in individual function.

Scope of a Variable

- **Whenever we declare a variable, we also determine its scope, lifetime and visibility.**

Scope	Scope is defined as the area in which the declared variable is 'accessible'. **There are five scopes**: program, file, function, block, and class. ***Scope*** is the region or section of code where a variable can be accessed. Scoping has to do with **when** a variable is accessible and used.
Lifetime	The lifetime of a variable is the period of time in which the variable is allocated a space (i.e., the period of time for which it "lives"). There are three lifetimes in C: static, automatic and dynamic. ***Lifetime* is the time duration where an object/variable is in a valid state.** Lifetime has to do with when a variable is created and destroyed
Visibility	Visibility is the "accessibility" of the variable declared. It is the result of hiding a variable in outer scopes.

Scope of a Variable

Function Structure

```
class FunctionDemo{
float f;
static int a;

public static void main()
{
  int i;
  static int j;
  func1(i);
}

void func1(int value)
{
  int x;
  //function body
  ....
}
}
```

Global Variable

Static Global Variable

Local Variables

Static Local Variable

Parameter Variable

Scope	Description
Local (block/ function)	"visible" within function or statement block from point of declaration until the end of the block.
Class	"seen" by class members.
File (program)	visible within current file.
Global	visible everywhere unless "hidden".

Lifetime of a variable

- The lifetime of a variable or object is the time period in which the variable/object has valid memory.
- Lifetime is also called "allocation method" or "storage duration“.

	Lifetime	Stored
Static	Entire duration of the program's execution.	data segment
Automatic	Begins when program execution enters the function or statement block and ends when execution leaves the block.	function call stack
Dynamic	Begins when memory is allocated for the object (e.g., by a call to malloc() or using new) and ends when memory is deallocated (e.g., by a call to free() or using delete).	heap

Scope vs Lifetime of a variable

Variable Type	Scope of a Variable	Lifetime of a Variable
Instance Variable	Throughout the class except in static methods	Until object is available in the memory
Class Variable	Throughout the class	Until end of the Class
Local Variable	Throughout the block/function in which it is declared	Until control leaves the block

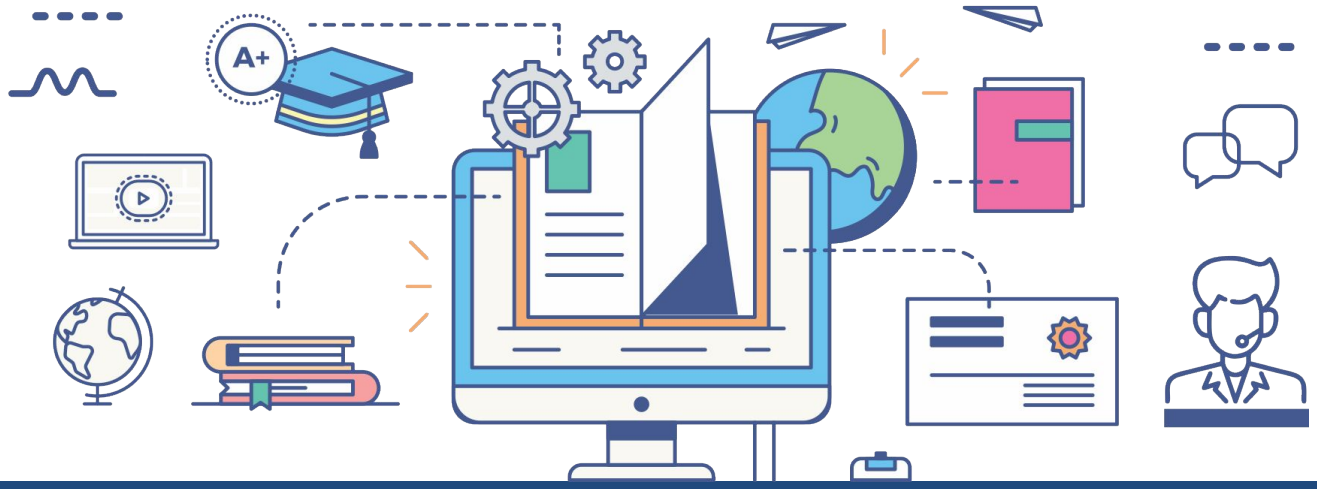
Exercise

- Write a function to check whether given number is prime or not.
- Write a function to search a given number from an array.

References

1. https://www.w3schools.com/java/
2. https://www.javatpoint.com/java-tutorial
3. https://www.geeksforgeeks.org/java/
4. Black Book Java
5. https://youtu.be/bm0OyhwFDuY?si=7WdQ3LOOLQi5ll8V

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in