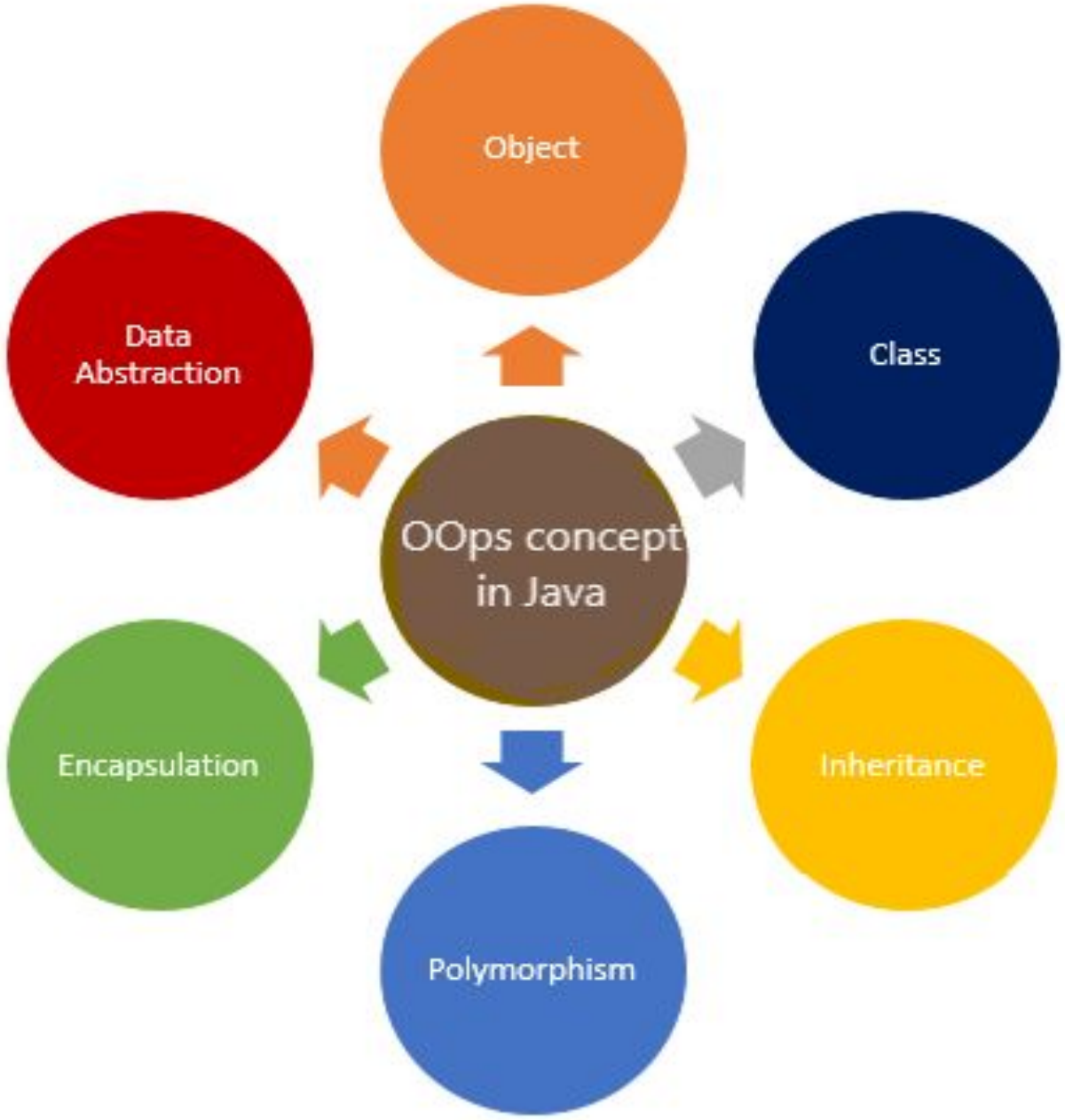


CS20004: Object Oriented Programming using Java

Lec-7

In this Discussion . . .

- Iteration Statements
- Jump Statements
- Arrays
  - Single dimensional Arrays
  - Multi-dimensional Arrays
- Taking inputs from user
- References

Iteration Statements

- Java iteration statements enable repeated execution of part of a program until a certain termination condition becomes true.
- In Java, we have three types of iteration statements that execute similarly:
  - for loop
  - while loop
  - do-while loop
- Despite their execution similarity, the above iteration statements differ in their syntax and condition checking time.

for loop

- In Java, for loop is similar to C and C++.
- It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
- We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Syntax

```
for(initialization;condition;increment/decrement)
{
    //statements
}
```

for loop

-

```
public class Forloopexp
{
	public static void main(String[] args)
	{
		int sum = 0;
		for(int j = 1; j<=10; j++)
		{
			sum = sum + j*j;
		}
		System.out.println("The sum of
the squares of first 10 natural numbers
is " + sum);
	}
}
```

for loop

-

```
public class Forloopexp
{
	public static void main(String[] args)
	{
		int sum = 0;
		for(int j = 1; j<=10; j++)
		{
			sum = sum + j*j;
		}
		System.out.println("The sum of the squares of first 10 natural numbers is " + sum);
	}
}
```

The sum of the squares of first 10 natural numbers is 385

for-each loop

- Java provides an enhanced for loop to traverse the data structures like array or collection.
- In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

Syntax

```
for(data_type var : array_name/collection_name)
{
    //statements
}
```

for-each loop

```
public class Foreachexp
{
	public static void main(String[] args)
	{
		String[] names =
{"Java","C","C++","Python","JavaScript"};
		System.out.println("Printing the
content of the array names:\n");
		for(String name:names)
		{
			System.out.println(name);
		}
	}
}
```

for-each loop

```
public class Foreachexp
{
	public static void main(String[] args)
	{
		String[] names =
{"Java","C","C++","Python","JavaScript"};
		System.out.println("Printing the
content of the array names:\n");
		for(String name:names)
		{
			System.out.println(name);
		}
	}
}
```

```
Printing the content of the array names:

Java
C
C++
Python
JavaScript
```

While loop

- The while loop is also used to iterate over the number of statements multiple times.
- However, if we don't know the number of iterations in advance, it is recommended to use a while loop.

Syntax

```
while(condition)
{
    //looping statements
}
```

While loop

- Unlike for loop, the initialization doesn't take place inside the loop statement in while loop.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

While loop

```
public class Whileexp
{
        public static void main(String[] args)
        {
                int i = 0;
                System.out.println("Printing the list of first 10 even
numbers \n");
                while(i<=10)
                {
                        System.out.println(i);
                        i = i + 2;
                }
        }
}
```

While loop

```
public class Whileexp
{
        public static void main(String[] args)
        {
                int i = 0;
                System.out.println("Printing the list of first 10 even
numbers \n");
                while(i<=10)
                {
                        System.out.println(i);
                        i = i + 2;
                }
        }
}
```

```
Printing the list of first 10 even numbers

0
2
4
6
8
10
```

do-while loop

- The do-while loop checks the condition at the end of the loop after executing the loop statements.
- When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.
- It is also known as the exit-controlled loop since the condition is not checked later.

Syntax

```
do
{
    //looping statements
}while(condition);
```

do-while loop

```
public class Dowhileexp
{
	public static void main(String[] args)
	{
		int i = 10;
		System.out.println("Printing the
list of quotients \n");
		do
		{
			System.out.println(i);
			i = i / 2;
		}while(i>4);
	}
}
```

do-while loop

```
public class Dowhileexp
{
	public static void main(String[] args)
	{
		int i = 10;
		System.out.println("Printing the
list of quotients \n");
		do
		{
			System.out.println(i);
			i = i / 2;
		}while(i>4);
	}
}
```

```
Printing the list of quotients

10
5
```

Jump Statements

- Jump statements are used to transfer the control of the program to the specific statements.
- Specifically, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java:
  - break
  - continue

break statement

- break statements are used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. *However, it breaks only the inner loop in the case of the nested loop.*
- The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.
- **Note:-** Java does not have goto statement

break statement

```
public class Jumpbreakexp
{
	public static void main(String[] args)
	{
		for(int i = 0; i<= 10; i++)
		{
			System.out.println(i);
			if(i==6)
			{
				break;
			}
		}
	}
}
```

break statement

```
public class Jumpbreakexp
{
	public static void main(String[] args)
	{
		for(int i = 0; i<= 10; i++)
		{
			System.out.println(i);
			if(i==6)
			{
				break;
			}
		}
	}
}
```

```
0
1
2
3
4
5
6
```

break statement example with labeled for loop

- break label;
- label: { ... }

```
public class Jumpbreaklabel
{
        public static void main(String[] args)
        {
                a:
                        for(int i = 0; i<= 10; i++)
                        {
                b:
                        for(int j = 0; j<=15;j++)
                        {
                c:
                        for (int k = 0; k<=20; k++)
                        {
                                System.out.println(k);
                                if(k==5)
                                {
                                        break a;
                                }
                        }
                }   }   }}
```

break statement example with labeled for loop

- break label;
- label: { ... }

```
public class Jumpbreaklabel
{
        public static void main(String[] args)
        {
                a:
                        for(int i = 0; i<= 10; i++)
                        {
                b:
                        for(int j = 0; j<=15;j++)
                        {
                c:
                        for (int k = 0; k<=20; k++)
                        {
                                System.out.println(k);
                                if(k==5)
                                {
                                        break a;
                                }
                        }
                }   }   }}
```

```
0
1
2
3
4
5
```

continue statement

- The break statement terminates the block of code, in particular it terminates the execution of an iterative statement
- Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

continue statement

```
public class Jumpcontinueexp
{

	public static void main(String[] args)
	{
		for(int i = 0; i<= 2; i++)
		{
			for (int j = i; j<=5; j++)
			{
				if(j == 4)
				{
					continue;
				}
				System.out.println(j);
			}
		}
	}
}
```

continue statement

```
public class Jumpcontinueexp
{

        public static void main(String[] args)
        {
                for(int i = 0; i<= 2; i++)
                {
                        for (int j = i; j<=5; j++)
                        {
                                if(j == 4)
                                {
                                        continue;
                                }
                                System.out.println(j);
                        }
                }
        }
}
```

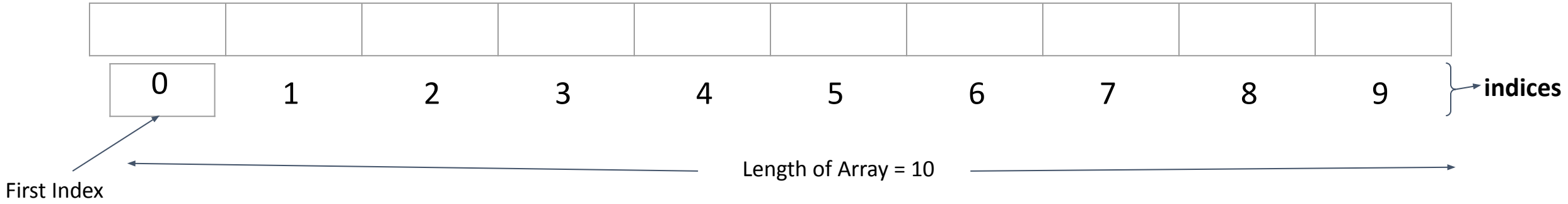
```
0
1
2
3
5
1
2
3
5
2
3
5
```

Math Class

- Math class contains all the floating-point functions that are used for geometry and trigonometry
  - Math.sin()
  - Math.sinh()
  - Math.cbrt(), Math.exp(), Math.log(), Math.log10(),Math.pow(), Math.sqrt()
  - Math.abs(), Math.ceil(), Math.floor()
  - Math.max(), Math.min(), Math.round()
  - Math.random()
  - Math.toDegrees(), Math.toRadians()
  - Math.PI

Java Arrays

- Java array is an object which can store only fixed set of elements of a similar data type in contiguous memory locations.
- Similar to normal arrays, arrays in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



Java Arrays

- Unlike C/C++, the length of the array can be obtained using the length member. In C/C++, we need to use the sizeof operator.

  - In Java, array is an object of a dynamically generated class.
  - Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces.
  - We can store primitive values or objects in an array in Java.
  - Like C/C++, we can also create single dimensional or multidimensional arrays in Java.

Java Arrays

Advantages	Disadvantages

• **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
• **Random access:** We can get any data located at an index position.
	• **Size Limit:** We can store only the fixed size of elements in the array.
	  ○ It doesn't grow its size at runtime.
	  ○ To solve this problem, collection framework is used in Java which grows automatically.

Java Arrays

- Arrays are:
  - declared
  - created
  - initialized
  - used
- Java Array Types:
  - Single Dimensional Array
  - Multidimensional Array

Single dimensional Java Arrays: Array declaration

- declaring an array identifier
- declaring the number of dimensions
- declaring the data type of the array elements

Syntax

```
datatype array_variable_name[ ];
```

or

```
datatype [ ]array_variable_name;
```

or

```
datatype[ ] array_variable_name;
```

Single dimensional Java Arrays: Java Arrays: Array Creation or Array Instantiation

- After declaration, no array actually exists
- In order to create an array, we use the **new** operator:

```
datatype array_variable_name[];

array_variable_name = new datatype[size];
```

- We can refer to the elements of this array through their indexes:

```
array_variable_name[index]
```

Note:- The new keyword is a **Java operator** that creates the object. Initialization: The new operator is followed by a call to a constructor, which initializes the new object.

Single dimensional Java Arrays: Java Arrays: Array Initialization

- Arrays can be initialized when they are declared

```
int monthDays[ ] = {31,10,18,2,11,10,22,17};
```

Declaration, Instantiation and Initialization of Java Array

```
class Arrayexpone
{
        public static void main(String args[])
        {
                int a[]=new int[5];
                a[0]=10;
                a[1]=20;
                a[2]=70;
                a[3]=40;
                a[4]=50;
                //traversing array
                for(int i=0;i<a.length;i++)
                {
                        System.out.println(a[i]);
                }
        }}
```

declaration and instantiation

initialization

length is the property of array

Output:

10
20
70
40
50

Single dimensional Java Arrays

- We can declare, instantiate and initialize the java array together by:

For ex- int a[ ]={33,3,4,5};

Passing Array to a Method in Java

- We can pass the java array to method so that we can reuse the same logic on any array.

```
class Arrayexptwo
{
	//creating a method which receives an array as a parameter
	static void min(int arr[])
	{
		int min=arr[0];
		for(int i=1;i<arr.length;i++)
		{
			if(min>arr[i])
			{
				min=arr[i];
			}
		}
		System.out.println(min);
	}
	public static void main(String args[])
	{
		int a[]={33,3,4,5};//declaring and initializing an array
		min(a);//passing array to method
	}}
```

Passing Array to a Method in Java

- We can pass the java array to method so that we can reuse the same logic on any array.

```
class Arrayexptwo
{
	//creating a method which receives an array as a parameter
	static void min(int arr[])
	{
		int min=arr[0];
		for(int i=1;i<arr.length;i++)
		{
			if(min>arr[i])
			{
				min=arr[i];
			}
		}
		System.out.println(min);
	}
	public static void main(String args[])
	{
		int a[]={33,3,4,5};//declaring and initializing an array
		min(a);//passing array to method
	}}
```

3

Returning Array from the Method

```
`class Arrayexpthree
{
	//creating method which returns an array
	static int[] get()
	{
		return new int[]{10,30,50,90,60};
	}
	public static void main(String args[])
	{
		//calling method which returns an array
		int arr[]=get();
		//printing the values of an array
		for(int i=0;i<arr.length;i++)
		{
			System.out.println(arr[i]);
		}
	}
}
```

```
10
30
50
90
60
```

ArrayIndexOutOfBoundsException

- The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array is negative, equal to the array size or greater than the array size while traversing the array.

```
public class Arrayexcpexp
{
        public static void main(String args[])
        {
                int arr[]={50,60,70,80};
                for(int i=0;i<=arr.length;i++)
                {
                        System.out.println(arr[i]);
                }
        }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java Arrays$ javac Arrayexc
pexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java Arrays$ java Arrayexcp
exp
50
60
70
80
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out
 of bounds for length 4
        at Arrayexcpexp.main(Arrayexcpexp.java:8)
```

Multidimensional Array in Java

- Here data is stored in row and column based index (also known as matrix form).

- **Multidimensional Array in Java declaration Syntax:**

```
datatype[ ][ ] array_variable_name; (or)

datatype [ ][ ]array_variable_name; (or)

datatype array_variable_name[ ][ ]; (or)

datatype [ ]array_variable_name[ ];
```

Multidimensional Array in Java

- Here data is stored in row and column based index (also known as matrix form).

- **Multidimensional Array in Java Instantiation Syntax:**

```
//3 row and 3 column

int[ ][ ] arr=new int[3][3];
```

Multidimensional Array in Java

- Here data is stored in row and column based index (also known as matrix form).

- **Multidimensional Array in Java Initialization Syntax:**

```
arr[0][0]=1;  arr[0][1]=2;  arr[0][2]=3;

arr[1][0]=4;  arr[1][1]=5;  arr[1][2]=6;

arr[2][0]=7;  arr[2][1]=8;  arr[2][2]=9;
```

Multidimensional Array in Java

- Here data is stored in row and column based index (also known as matrix form).

```
class Arrayexpfour
{
public static void main(String args[])
    {
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        //printing 2D array
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }
    }}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java Arrays$ javac Arrayexpfour.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java Arrays$ java Arrayexpfour
1 2 3
2 4 5
4 4 5
```

Procedural Programming

- decomposition around operations
- operation are divided into smaller operations
- Drawbacks:
  - data is given a second- class status when compared with operations
  - difficult to relate to the real world: no functions in real world
  - difficult to create new data types: reduces extensibility
  - programs are difficult to debug: little restriction to data access

Procedural Programming

- decomposition around operations
- operation are divided into smaller operations
- Drawbacks:
  - programs are hard to understand: many variables have global scope
  - programs are hard to reuse: data/functions are mutually dependent
  - little support for developing and comprehending really large programs
  - top- down development approach tends to produce monolithic programs

    [Monolithic:- an application that is made up of one large codebase that includes all the application components, such as the frontend code, backend code, and configuration files.]

How to get input from user in Java

Java Scanner Class

- Java Scanner class allows the user to take input from the console.
- It belongs to java.util package.
- It is used to read the input of primitive types like int, double, long, short, float, and byte.

Syntax

```
Scanner sc=new Scanner(System.in);
```

How to get input from user in Java

Java Scanner Class

Syntax

```
Scanner sc=new Scanner(System.in);
```

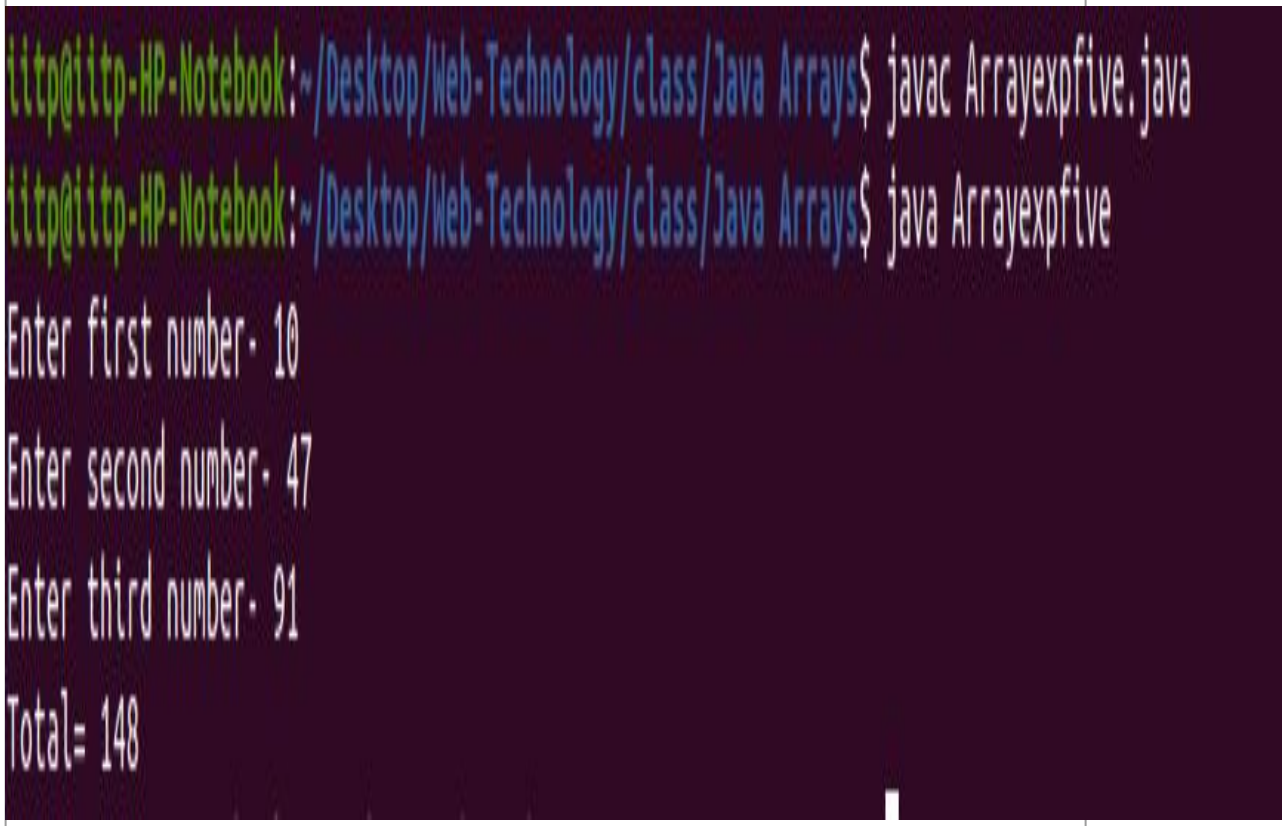
- The above statement creates a constructor of the Scanner class having System.in as an argument.
- It means it is going to read from the standard input stream of the program. The java.util package should be import while using Scanner class.
- It also converts the Bytes (from the input stream) into characters using the platform's default charset.

Methods of Java Scanner

- Java Scanner class provides the following methods to read different primitives types:

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

```
import java.util.*;
class Arrayexpfive
{
public static void main(String[] args)
{
Scanner sc= new Scanner(System.in);    //System.in is
a standard input stream
System.out.print("Enter first number- ");
int a= sc.nextInt();
System.out.print("Enter second number- ");
int b= sc.nextInt();
System.out.print("Enter third number- ");
int c= sc.nextInt();
int d=a+b+c;
System.out.println("Total= " +d);  }  }
```



References

1. https://www.geeksforgeeks.org/type-conversion-java-examples/
2. https://www.javatpoint.com/scope-of-variables-in-java
3. https://i.stack.imgur.com/Ij3vJ.png
4. https://www.javatpoint.com/control-flow-in-java
5. https://www.scaler.com/topics/expression-in-java/
6.