Notes / Reference material:

- Video for set up and installation: Click here
- Interesting links

- https://en.wikipedia.org/wiki/The_Imitation_Game
- Coding is not difficult by Bill Gates, Mark Zuckerberg
- Real life examples of algorithms used

Note before reading:
Following notes have terminologies and concepts that might not be discussed in classes. You can dive deep into reading about them on the internet. Feel free to raise doubts about them in classes.

- **What and why?**

How humans interact using languages, we can interact with computers using its language.

- **About Java**

Click here

- **What is Java?**

  Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centres, game consoles, scientific supercomputers, cell phones, etc.

- **What is Java Platform?**

Java Platform is a collection of programs that help programmers to develop and run Java programming applications efficiently. It includes an execution engine, a compiler, and a set of libraries in it. It is a set of computer software and specifications. James Gosling developed the Java platform at Sun Microsystems, and the Oracle Corporation later acquired it.

- **What is Java used for?**

Here are some important Java applications:

- It is used for developing Android Apps (Example of calculator app)
- Helps you to create Enterprise Software
- Wide range of Mobile java Applications
- Scientific Computing Applications
- Use for Big Data Analytics
- Java Programming of Hardware devices
- Used for Server-Side Technologies like Apache, JBoss, GlassFish, etc.

- **History of Java Programming Language**

Here are important landmarks from the history of the Java language:

- The Java language was initially called OAK.
- Originally, it was developed for handling portable devices and set-top boxes. Oak was a massive failure.
- In 1995, Sun changed the name to “Java” and modified the language to take advantage of the burgeoning www (World Wide Web) development business.
- Later, in 2009, Oracle Corporation acquired Sun Microsystems and took ownership of three key Sun software assets: Java, MySQL, and Solaris.

- **Java Features**

Here are some important Java features:

- It is one of the easy-to-use programming languages to learn.
- Write code once and run it on almost any computing platform.
- Java is platform-independent. Some programs developed in one machine can be executed in another machine.
- It is designed for building object-oriented applications.
- It is a multithreaded language with automatic memory management.

Source: https://www.guru99.com/java-platform.html

- **Why Java is so popular?**

https://www.frgconsulting.com/insights/why-is-java-so-popular-developers

- **What is data?**

Data can be defined as a representation of facts, concepts, or instructions in a formalized manner, which should be suitable for communication, interpretation, or processing by human or electronic machine.

- **Why do we need data structures?**
    - Data Structure is used for organizing the data in memory. There are various ways of organizing the data in the memory for eg. array, list, stack, queue and many more.
    - The data structure isn't a programming language like C, C++, Java, etc. It is a set of algorithms that can be used in any programming language to organize the data in the memory.
    - As applications are becoming more complex and the amount of data is increasing day by day, which may cause problems with processing speed, searching data, handling multiple requests etc. Data structure provides a way of organizing, managing, and storing data efficiently. With the help of data structure, the data items can be traversed easily. Data structure provides efficiency, reusability and abstraction. It plays an important role in enhancing the performance of a program because the main function of the program is to store and retrieve the user's data as fast as possible.

What is an algorithm?

In computer programming terms, an algorithm is a set of well-defined instructions to solve a particular problem. It takes a set of input and produces a desired output. For example,

An algorithm to add two numbers:

1. Take two number inputs

2. Add numbers using the + operator
3. Display the result.

Source: https://www.mygreatlearning.com/blog/data-structure-tutorial-for-beginners/

Real life algo examples:

https://www.invisibly.com/learn-blog/algorithm-examples-everyday-life

• Variables

Variables are containers for storing data values.

In Java, there are different types of variables, for example:

- `String` - stores text, such as "Hello". String values are surrounded by double quotes
- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `float` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `boolean` - stores values with two states: true or false

```
int geek= 10;
geek= 20;  // myNum is now 20
System.out.println(geek);


String name = "Geekster";
System.out.println(name);
```

- Taking input in java

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner scn= new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = scn.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user
input
  }
}
```

Operators

+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y

%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Other operators

=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Source: https://www.w3schools.com/java/

HOMEWORK DAY 1

1. **Install Visual studio code and JAVA. Click here**
2. **Read about JAVA and it's applications.**
3. **Try codes done in class on your own IDE.**
4. **Try to think of some data and identify which data type will you use to store it. Ex: Percentage: Float, Speed: Integer etc.**

- **Comparison Operators**

Here's a sample program that demonstrates the use of operators for comparison in Java:

```
public class ComparisonOperators
{
     public static void main(String args[])
{
     int geek = 100, ster = 50;
     System.out.println("geek > ster : " + (geek > ster)); // true
     System.out.println("geek < ster : " + (geek < ster)); // false
     System.out.println("geek >= ster : " + (geek >= ster)); // true
     System.out.println("geek <= ster : " + (geek <= ster)); // false
     System.out.println("geek == ster : " + (geek == ster)); // false
     System.out.println("geek != ster : " + (geek != ster)); // true
}
}
```

- **Equal to**

This operator checks whether the value on the operator's left side is equal to the value on the right side.

Example:

```
import java.util.Scanner;
public class ComparisonExample {
public static void main(String[] args) {
int g, s;
Scanner sc= new Scanner(System.in);
//take the value of x as input from user and store it in
variable g
```

```
System.out.print("Enter the value of g : ");
g = sc.nextInt();
//take the value of s as input from user
System.out.print("Enter the value of s : ");
//store the value in variable s
s = sc.nextInt();

System.out.println(g == s);
}
}
```

Output:

Case 1: g = 3; s =5; Returns false as they are not equal

Case 2: g = 4; s =4; Returns true as they are equal

- **Not Equals to**

Example:

```
import java.util.Scanner;
public class ComparisonExample {
public static void main(String[] args) {
int g, s;
Scanner sc= new Scanner(System.in);
//take the value of x as input from user and store it in
variable g
System.out.print("Enter the value of g : ");
g = sc.nextInt();
//take the value of s as input from user
System.out.print("Enter the value of s : ");
//store the value in variable s
s = sc.nextInt();

System.out.println(g != s);
```

```
}
}
```

Output:

Case 1: g = 3; s =4; Returns true as they are not equal

Case 2: g = 3; s =3; Returns false as they are equal

- **Less than**

This operator checks whether the value on the operator's left side is less than the value on the right side.

Example:

```
import java.util.Scanner;
public class ComparisonExample {
public static void main(String[] args) {
int g, s;
Scanner sc= new Scanner(System.in);
//take the value of x as input from user and store it in
variable g
System.out.print("Enter the value of g : ");
g = sc.nextInt();
//take the value of s as input from user
System.out.print("Enter the value of s : ");
//store the value in variable s
s = sc.nextInt();

System.out.println(g < s);
}
}
```

Output:

Case 1: g = 4; s =6; Returns true as g is less than s

Case 2: g = 44; s =32; Returns false as g is not less than s

- **Greater than**

This operator checks whether the value on the operator's left side is greater than the value on the right side.

Example:

```
import java.util.Scanner;
public class ComparisonExample {
public static void main(String[] args) {
int g, s;
Scanner sc= new Scanner(System.in);
//take the value of x as input from user and store it in
variable g
System.out.print("Enter the value of g : ");
g = sc.nextInt();
//take the value of s as input from user
System.out.print("Enter the value of s : ");
//store the value in variable s
s = sc.nextInt();

System.out.println(g > s);
}
}
```

Output:

Case 1: x = 67; y =66; Returns true as x is greater than y

Case 2: x = 43; y =57; Returns false as x is less than y

- **Logical operators**

These operators are used to perform logical "AND", "OR" and "NOT" operation, i.e. the function similar to AND gate and OR gate in digital electronics. They are used to combine two or more conditions/constraints or to complement the

evaluation of the original condition under particular consideration. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e. it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Let's look at each of the logical operators in a detailed manner:

1. **'Logical AND' Operator(&&):** This operator returns true when both the conditions under consideration are satisfied or are true. If even one of the two yields false, the operator results false. For example, cond1 && cond2 returns true when both cond1 and cond2 are true (i.e. non-zero).

   Syntax:

   ```
   condition1 && condition2
   ```

Example:

```
a = 10, b = 20, c = 20

condition1: a < b
condition2: b == c

if(condition1 && condition2)
d = a+b+c

// Since both the conditions are true
      d = 50.
```

- **'Logical OR' Operator(||):** This operator returns true when one of the two conditions under consideration are satisfied or are true. If even one of the two yields true, the operator results true. To make the result false, both the constraints need to return false.

Syntax:

```
condition1 || condition2
```

Example:

```
a = 10, b = 20, c =20

condition1: a < b

condition2: b > c

if(condition1 || condition2)

d = a+b+c

// Since one of the condition is true

d = 50.
```

- **'Logical NOT' Operator(!):** Unlike the previous two, this is a unary operator and returns true when the condition under consideration is not satisfied or is a false condition. Basically, if the condition is false, the operation returns true and when the condition is true, the operation returns false.

Syntax:

```
!(condition)
```

Example:

```
a = 10, b = 20
```

```
!(a<b) // returns false
```

```
!(a>b) // returns true
```

Let us see the truth tables behind the logical operators in Java for better understanding

- **LOGICAL AND (&&)**

Truth table

Condition 1	Condition 2	Condition 1 && Condition 2
True	True	True
True	False	False
False	True	False
False	False	False

- **LOGICAL OR (||)**

Condition 1	Condition 2	Condition 1 || Condition 2
True	True	True
True	False	True
False	True	True
False	False	False

Source: https://www.w3schools.com/java/

• Conditions

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement

2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

- Decision-Making statements

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement

3. if-else-if ladder
4. Nested if-statement

Let's understand the if-statements one by one.

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

Consider the following example in which we have used the **if** statement in the java code.

Student.java

Student.java

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

x + y is greater than 20

2) if-else statement

The if-else statement

is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

Consider the following example.

Student.java

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than      10");
}   else {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

x + y is greater than 20

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Consider the following example.

```
Student.java
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
```

```
}
}
}
```

Output:

Delhi

4. Nested if-statement

In nested if-statements, the if statement can contain an if or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
   statement 1; //executes when condition 1 is true
   if(condition 2) {
     statement 2; //executes when condition 2 is true
    }
else{
   statement 2; //executes when condition 2 is false
   }
}
```

Source: https://www.w3schools.com/java/

- Loops -

1. What is Loop and why do we need it?

Loops in Java come into use when we need to repeatedly execute a block of statements.
Java for loop provides a concise way of writing the loop structure. The for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization expr; test expr; update exp)
{
     // body of the loop
     // statements we want to execute
}
```

2. For loop

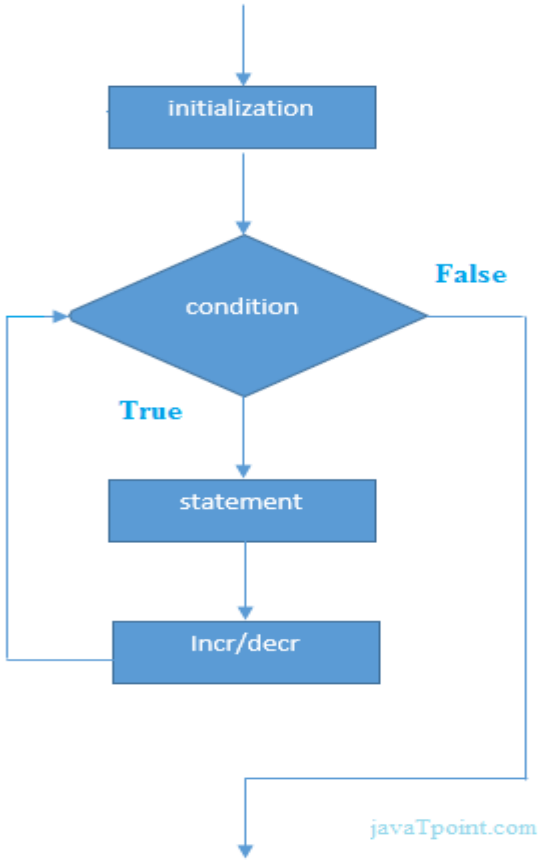
A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

Syntax:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

Flowchart:



Example:

ForExample.java

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
```

1. }

Geekster

In following each question, Show how 'i' is incremented in loop and what gets printed and explain what is dry run.

- WAP to print 1-n

```java
import java.util.Scanner;

public class Print_1_To_N_UsingFor
{
	public static void main(String[] args)
	{
		//create object of scanner class
		Scanner scanner = new Scanner(System.in);

		// enter the value of " n ".
		System.out.print("Enter the value n : ");

		// read the value.
		int n = scanner.nextInt();

		System.out.println("Numbers are : " );
		for(int i=1; i<=n; i++)
		{
			System.out.println(i);
		}
	}
}
```

Output

```
Enter the value n : 15
Numbers are :
1
2
3
4
5
6
7
8
9
10
11
```

12
13
14
15

- WAP to print a-z

```
class Alphabets
{
 public static void main(String[]args)
 {
// we use the escape sequence \n to print the characters in a
new line.
   for(char i = 'A' ; i <= 'Z'; i++)
// initialise the variable with the starting letter and iterate
the loop till the desired character occurs.
   {
     System.out.println(i);
// A space should be given beside %c to maintain the spacing
between letters.
   }
 }
}
```

- WAP to print even numbers till n.

```java
public class DisplayEvenNumbers {

    public static void main(String[] args) {
        //number
        int n = 20;

        //print all even numbers <=n
        for (int i=2; i<=n; i+=2) {
            System.out.print(i+"  ");
        }
    }
}
```

Homework exercise:

Q. Guess the output

1.

```
class ForSample
{
   public static void main(String s[])
    {
        for(int i = 0; i <= 5; i++ )
{
           System.out.println("i = " + i );
        }
    }
}
```

2.

```
class ForSample
{
   public static void main(String s[])
    {
        for(int i = 1; i <= 0; i++ )
{
           System.out.println("i = " + i );
        }
    }
}
```

3.

```
class ForSample
{
   public static void main(String s[])
    {
```

```
        for(int i = 10; i >= 5; i- )
{
            System.out.println(“Hey”);
        }
    }
}
```

4.

```
class Test {
public
    static void main(String[] args)
    {
        for (int i = 0;; i++)
            System.out.println("HELLO GEEKS");
    }
}
```

5.

```
class Test {
public
    static void main(String[] args)
    {
        for (int i = 0;i<=25; i++)
          {
```

```
            System.out.println(i+ " ");
            if(i%5 == 0)
                {
                  System.out.println("\n");
                }
          }
    }
}
```

6.

```
public class Test{
public static void main(String[] args){
 for(int i = 0; i<5; i+=2)
 {
 System.out.println("Hello");
 i+=2;
 }
}
}
```

METHODS/ FUNCTIONS

A method is a way to perform some task. Similarly, the method in Java is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using methods. In this section, we will

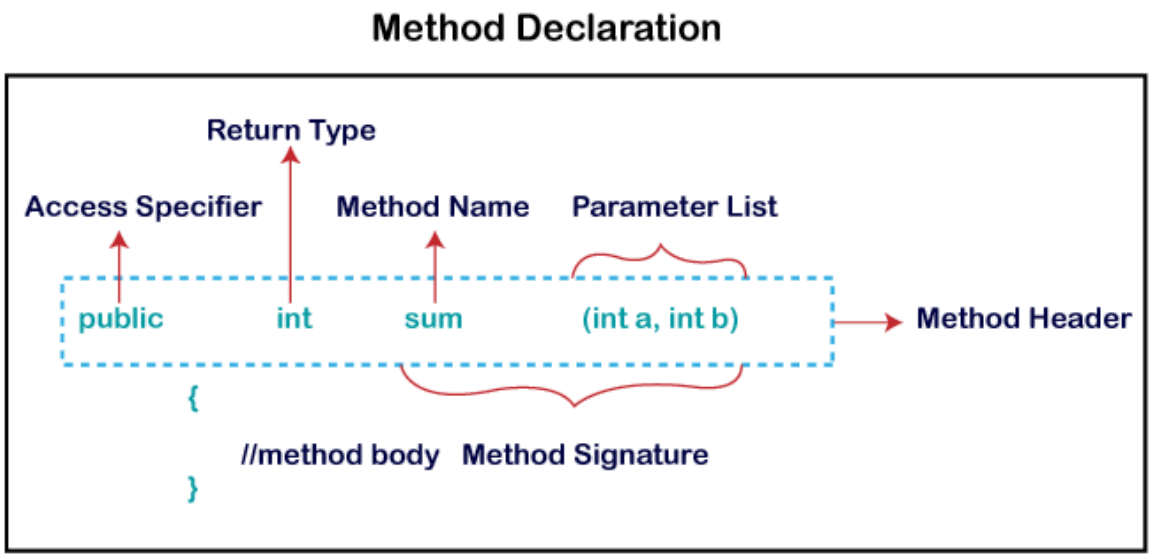
learn what a method is in Java, types of methods, method declaration, and how to call a method in Java.

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We are not required to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the main() method, from where execution of code begins.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use the void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by an adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

Single-word method name: sum(), area()

Multi-word method name: area_Of_Circle(), string_Comparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries and is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in

the program at any point. Some predefined methods are **length(), equals(), compareTo(), sqrt(),** etc.

User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

WAP to print the sum of two numbers, using method.

```
import java.util.Scanner;
public class IntegerSumExample2 {
    public static void main(String[] args) {
      System.out.println("Enter the Two numbers for addition: ");
      Scanner readInput = new Scanner(System.in);
      int a = readInput.nextInt();
        int b = readInput.nextInt();
        readInput.close();
        // It will return the sum of a and b.
        System.out.println("The sum of a and b is = " +
Integer.sum(a, b));
    }
}
```

Output:

Enter the Two numbers for addition:

3435

2865

The sum of a and b is = 6300

Meaning of pass by value and reference by example

Pass by Value: In the pass by value concept, the method is called by passing a value. So, it is called pass by value. It does not affect the original parameter.

Pass by Reference: In the pass by reference concept, the method is called using an alias or reference of the actual parameter. So, it is called pass by reference. It forwards the unique identifier of the object to the method. If we made changes to the parameter's instance member, it would affect the original value.

Java does not supports pass by reference concept.

About the Parameters Passed in Java

The fundamental concept for passing the parameters in modern programming languages is passing by value and passing by reference. But, in Java, the pass by reference concept is degraded. It supports only the pass by value concept.

The primitive variables hold the actual values, whereas the non-primitive variables hold the reference variable. However, both variables use stack memory to store the values.

In Java, during the method invocation, a copy of each argument is created then passed to the method.

In the case of primitive data types, it copies the value inside stack memory then passes it to the method. In the case of non-primitive data types, it points a reference in stack memory to actual data, which occurs in a heap. When we pass an object, it will copy a reference from the stack memory and pass it to the callee method.

Source: https://www.javatpoint.com/method-in-java

Homework exercise:

Q. Guess the output

1.

```
public class Main{
        static void myMethod(){
            System.out.println("I just got executed!");
        }
        public static void main(String[] args) {
            myMethod();
        }
    }
```

2.

```
public class Main{
        static void myMethod(){
            System.out.println("I just got executed!");
        }
        public static void main(String[] args) {
            myMethod();
            myMethod();
            myMethod();
        }
    }
```

3.

```
class Main {
        // create a method
        public int addNumbers(int a, int b){
```

```
            int sum = a + b;
            // return value
            return sum;
        }
        public static void main(String[] args) {

            int num1 = 25;
            int num2 = 15;
            // create an object of Main
            Main obj = new Main();
            // calling method
            int result = obj.addNumbers(num1, num2);
            System.out.println("Sum is: " + result);
        }
    }
```

4.

```
class Main{
        // create a method
        public static int square(int num) {
            // return statement
            return num * num;
        }
        public static void main(String[] args) {
            int result;
            // call the method
            // store returned value to result
            result = square(10);
            System.out.println("Squared value of 10 is: " + result);
        }
    }
```

5.

```
public class Main{

        // method defined
        private static int getSquare(int x){
            return x * x;
        }
        public static void main(String[] args){
```

```
            for (int i = 1; i <= 5; i++) {
                // method call
                int result = getSquare(i);
                System.out.println("Square of " + i + " is: " + result);
            }
        }
    }
```

6.

```
import java.util.Scanner;

public class pattern
{
        public static void main(String[] args) {
            int i, j, k = 1;
            for (i = 1; i <= 5; i++) {
                for (j = 1; j< i + 1; j++) {
                    System.out.print(k++ + " ");
                }

                System.out.println();
            }
        }

    }
```

8.

```
 import java.util.Scanner;

public class pattern
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of rows: ");

        int rows = sc.nextInt();
        for (int i = 1; i <= rows; i++)
        {
            for (int j = 1; j <= i; j++)
```

```
            {
                if(j%2 == 0)
                {
                    System.out.print(0);
                }
                else
                {
                    System.out.print(1);
                }
            }

            System.out.println();
        }
    }
}
```

9.

```
import java.util.Scanner;

public class pattern
{
    public static void main(String[] args)
    {
        int alphabet = 65;
                for (int i = 0; i <= 5; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                System.out.print((char) (alphabet + j) + " ");
            }
            System.out.println();
        }
    }
}
```

10.

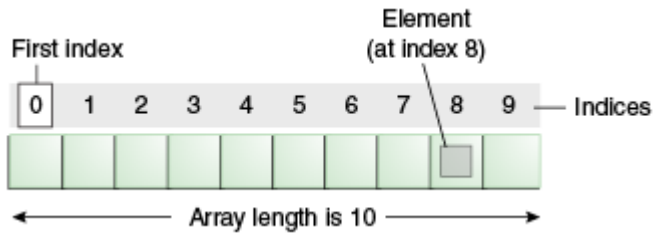
```
public class pattern
{
    public static void main(String[] args)
{
```

```
        for (int i = 0; i <= 5; i++) { int alphabet = 65; for (int j = 5; j
> i; j--)
        {
            System.out.print(" ");
        }
        for (int k = 0; k <= i; k++)
        {
            System.out.print((char) (alphabet + k) + " ");
        }
        System.out.println();
    }
}
}
```

• Arrays

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Arrays in Java are index-based, the first element of the array is stored at the 0th index, the 2nd element is stored on the 1st index and so on.



Advantages

- **Code Optimization:** It makes the code optimised, we can retrieve or sort the data efficiently.

- **Random access:** We can get any data located at an index position.

Disadvantages

- **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.

Can store only one type of data i.e int,float,char etc.

The value held in the box can change, or vary. But **each variable can only hold one item of data** . An array is a series of memory locations – or 'boxes' – each of which holds a single item of data, but with each box sharing the same name. All data in an array must be of the same data type .

Can't delete elements, can't append in back

Elements are sequentially stored one after another. When someone wants to access an element at a certain index, pointer arithmetic (which is the mechanism under the hood) allows one to quickly and efficiently obtain any particular element.

If the index of a requested element is `3`, the underlying mechanism simply needs to take the memory address of the *zero*-th element and add three times the size of each element. Since all array elements have the same size, this kind of computation leads directly to the element with index `3`. Furthermore, this happens in `O(1)` complexity which means it's as fast as it can be.

What makes removing an array element difficult is the fact that all elements are stored sequentially in a single memory block. Due to the nature of array's memory placement, it is simply impossible to remove the element directly.

Instead, to "remove" any element, all subsequent elements need to be shifted backward by one place. This will create an illusion that a specific element was removed.

Example of array:

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
 }
}
```

Output:

10

20

70

40

50

Types of Array in java

There are two types of arrays.

- Single Dimensional Array
- Multidimensional Array

- WAP printing an array

```
import java.util.Scanner;
```

```
public class arrayinput {

    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.print("Enter size of array");
        // INPUT SIZE
        int size = scn.nextInt();

        // ARRAY INITIALIZE
        int[] arr = new int[size];

        System.out.print("Enter values in array");
        // FILL ARRAY USING INPUT
        for (int i = 0; i < size; i++) {
            arr[i] = scn.nextInt();
        }

        // PRINT ARRAY
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + " ");
        }
    }

}
```

- WAP reverse printing an array

```
import java.util.*;

public class reversePrint {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
```

```
        // input
        input(arr);
        // REVERSE ARRAY PRINTING
        printReverseArray(arr);
    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void printReverseArray(int[] arr) {
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

}
```

Arrays

- WAP print alternate elements of array

```
import java.util.*;

public class PrintEvenValues {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
```

```
        int[] arr = new int[size];
        // input
        input(arr);
        printEvenValues(arr);
    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void printEvenValues(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            if ((arr[i] % 2) == 0) {
                System.out.print(arr[i] + " ");
            }
        }
        System.out.println();
    }

    public static void printOddValues(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            if ((arr[i] % 2) == 1) {
                System.out.print(arr[i] + " ");
            }
        }
        System.out.println();
    }

}
```

- WAP to print alternate elements in reverse

```
import java.util.*;

public class reversePrint {
    public static Scanner scn = new Scanner(System.in);
```

```
    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
        // input
        input(arr);
        // REVERSE ARRAY PRINTING
        printReverseArray(arr);
    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void printReverseArray(int[] arr) {
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

}
```

- WAP to take input of an array and find the sum.

```
import java.util.*;

public class SumOfArray {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
        // input
```

```
        input(arr);

        sumOfArray(arr);

    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void sumOfArray(int[] arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            // sum = sum + arr[i];
            sum += arr[i];
        }
        System.out.println("Sum Of Array" + sum);
    }

}
```

- WAP to count even in an array.

```
import java.util.*;

public class CountEvenValues {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
        // input
        input(arr);
        countEvenValues(arr);
```

```
        countOddValues(arr);
    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void countEvenValues(int[] arr) {
        int countOfEven = 0;// count of even
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] % 2 == 0) {
                countOfEven++;
            }
        }
        System.out.println("Even element count is: " + countOfEven);
    }

    public static void countOddValues(int[] arr) {
        int countOfOdd = 0;// count of even
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] % 2 != 0) {
                countOfOdd++;
            }
        }
        System.out.println("Even element count is: " + countOfOdd);
    }

}
```

- WAP to count elements with index=value

```
class Geek {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int arr[], int n)
    {
        int ans = 0;
```

```
        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;
        return ans;
    }

    //driver code
    public static void main (String[] args)
    {
        int arr[] = { 1, 1, 2 };
        int n = arr.length;

        System.out.println(countPairs(arr, n));
    }
}
```

Arrays

- WAP to find first occurrence of an element (Linear search)

```
import java.util.*;

                    public class FirstLastOcc {
                      public static Scanner scn = new Scanner(System.in);


                      public static void main(String[] args) {
                        // size input
                        int size = scn.nextInt();
                        // array initialize
                        int[] arr = new int[size];
```

```
        // input
        input(arr);
        int value = scn.nextInt();
        // firstLastOcc(arr, value);
        firstLastOcc_(arr, value);
    }

    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void firstLastOcc(int[] arr, int value) {
        int firstocc = -1;
        // FIRST OCCURENCE
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == value) {
                firstocc = i;
                break;
            }
        }
        System.out.println(firstocc);
        // LAST OCCURENCE
        for (int i = arr.length - 1; i >= 0; i--) {
            if (arr[i] == value) {
                System.out.println(i);
                break;
            }
        }
    }

    public static void firstLastOcc_(int[] arr, int value) {
        int i = 0;
        // FIRST OCCURENCE
        for (i = 0; i < arr.length; i++) {
            if (arr[i] == value) {
                break;
            }
        }
        System.out.println(i);
        // LAST OCCURENCE
        for (i = arr.length - 1; i >= 0; i--) {
            if (arr[i] == value) {
```

```
                    break;
                }
            }
            System.out.println(i);
        }


    }
```

- WAP to find the largest/smallest in an array . (Introduce Math.max,min)

```
import java.util.*;


                public class minmax {
                    public static Scanner scn = new Scanner(System.in);


                    public static void main(String[] args) {
                        // size input
                        int size = scn.nextInt();
                        // array initialize
                        int[] arr = new int[size];
                        // input
                        input(arr);
                        minElement(arr);
                        maxElement(arr);
                    }


                    public static void input(int[] arr) {
                        for (int i = 0; i < arr.length; i++) {
                            arr[i] = scn.nextInt();
                        }
                    }


                    public static void minElement(int[] arr) {
                        int min = Integer.MAX_VALUE;
                        for (int i = 0; i < arr.length; i++) {
                            if (arr[i] < min) {
                                min = arr[i];
                            }
                        }
```

```
            System.out.println(min);


        }


        public static void maxElement(int[] arr) {
            int max = Integer.MIN_VALUE;
            for (int i = 0; i < arr.length; i++) {
                if (arr[i] > max) {
                    max = arr[i];
                }
            }
            System.out.println(max);


        }


    }
```

- WAP to check if the sorted array has any duplicates.

```
import java.util.*;


public class CountOfOccurence {
    public static Scanner scn = new Scanner(System.in);


    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
        // input
        input(arr);
        int value = scn.nextInt();// value to search
        countOccurence(arr, value);
    }


    public static void input(int[] arr) {
```

```
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }

    public static void countOccurence(int[] arr, int value) {
        int count = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == value) {
                count++;
            }
        }
        System.out.println(count);
    }

}
```

- Introduce swap

```
import java.util.*;

public class swap {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        int a = scn.nextInt();
        int b = scn.nextInt();
        // SWAP VALUES OF A AND B
        // System.out.println(a + " " + b);
        // int temp = a;
        // a = b;
        // b = temp;
        // System.out.println(a + " " + b);

        // SWAP VALUES OF A AND B without 3rd
        // a = a + b;
        // b = a - b;
        // a = a - b;
        // System.out.println(a + " " + b);
```

```
        // SINGLE LINE SWAP
        b = ((a + b) - (a = b));
        System.out.println(a + " " + b);


    }


}
```

- WAP reverse an array

```
import java.util.*;

public class ReverseArray {
    public static Scanner scn = new Scanner(System.in);


    public static void main(String[] args) {
        // size input
        int size = scn.nextInt();
        // array initialize
        int[] arr = new int[size];
        // input
        input(arr);
        printArray(arr);
        reverseArray(arr);
        printArray(arr);
    }


    public static void input(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scn.nextInt();
        }
    }


    public static void reverseArray(int[] arr) {
        for (int i = 0; i < arr.length / 2; i++) {
            // swap ->(arr[i],arr[arr.length-i-1])
            // b=((a+b)-(a=b))
            // a=arr[i]
            // b=arr[j]
```

```
                int j = arr.length - i - 1;


                // swap only aand b
                // int a = arr[i];
                // int b = arr[j];
                // b = ((a + b) - (a = b));


                // arr[i] = b;
                // arr[j] = a;


                // actual swap in array
                arr[j] = ((arr[i] + arr[j]) - (arr[i] = arr[j]));
            }
        }


        public static void printArray(int[] arr) {
            for (int ele : arr) {
                System.out.print(ele + " ");
            }
            System.out.println();
        }


    }
```

Array contd

- WAP bubble sort

```
import java.util.Scanner;


                                                public class BubbleSort {
                                                   public static Scanner scn = new
                                                Scanner(System.in);
```

```
    public static void main(String[] args) {
        int[] arr = { 10, 8, 7, 3, 2, 4, 1, 5, 6 };
        printArray(arr);
        bubbleSort(arr);
        printArray(arr);
    }

    public static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    swap(arr, j, j + 1);
                }
            }
        }
    }

    public static void swap(int[] arr, int i, int j) {
        arr[j] = ((arr[i] + arr[j]) - (arr[i] = arr[j]));
    }

    public static void printArray(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

- WAP selection sort

```
import java.util.Scanner;

public class SelectionSort {
    public static Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        int[] arr = { 7, 5, 4, 3, 2, 1, 6 };
        printArray(arr);
        selectionSort(arr);
        printArray(arr);
    }

    public static void selectionSort(int[] arr) {
```

```
    int len = arr.length;
    for (int i = len - 1; i >= 0; i--) {
        int maxidx = 0;// index of maximum value
        for (int j = 0; j <= i; j++) {
            if (arr[j] > arr[maxidx]) {
                maxidx = j;
            }
        }
        swap(arr, i, maxidx);
    }
}

public static void swap(int[] arr, int i, int j) {
    arr[j] = ((arr[i] + arr[j]) - (arr[i] = arr[j]));
}

public static void printArray(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
```

- WAP insertion sort

```
import java.util.Scanner;

public class InsertionSort {
    public static Scanner scn = new
Scanner(System.in);

    public static void main(String[] args) {
        int[] arr = { 10, 8, 7, 3, 2, 4, 1, 5, 6 };
        printArray(arr);
        insertionSort(arr);
        printArray(arr);
    }

    public static void insertionSort(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
```

```
                for (int j = i - 1; j >= 0; j--) {
                    if (arr[j] > arr[j + 1]) {
                        swap(arr, j, j + 1);
                    }
                }
            }
        }

        public static void swap(int[] arr, int i, int j) {
            arr[j] = ((arr[i] + arr[j]) - (arr[i] = arr[j]));
        }

        public static void printArray(int[] arr) {
            for (int i = 0; i < arr.length; i++) {
                System.out.print(arr[i] + " ");
            }
            System.out.println();
        }
    }
```

Overview:

- WAP to create an array till n (input) where i'th element is i'th fibonacci number.

```
import java.util.Scanner;

public class Fibonacci {

    public static java.util.Scanner scn = new Scanner(System.in);

    public static void main(String[] args) {
        int n = scn.nextInt();
        int ans = fibonacci(n);
        System.out.println(ans);
    }

    public static int fibonacci(int n) {
        int a = 0;
```

```
        int b = 1;
        int c = 0;
        for (int i = 1; i <= n - 2; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return c;
    }


}
```

- WAP to create an array till n (input) where i’th element is i factorial.

```
Import
java.util.HashMap;
                    import java.util.Scanner;


                    public class Factorial {


                      public static java.util.Scanner scn = new Scanner(System.in);


                      public static void main(String[] args) {
                        int n = scn.nextInt();
                        int ans = factorial(n);
                        System.out.println(ans);
                      }


                      public static int factorial(int n) {
                        int product = 1;
                        for (int i = 1; i <= n; i++) {
                            product *= i;
                        }
                        return product;
                      }
                    }
```

- WAP to count the number of prime numbers in an array.

```java
import java.util.Arrays;
import java.util.Vector;

// Java program to find count of
// primes in given array.
class Geekster
{

    // Function to find count of prime
    static int primeCount(int arr[], int n)
    {
        // Find maximum value in the array
        //.*max_element(arr, arr+n);
        int max_val = Arrays.stream(arr).max().getAsInt();

        // USE SIEVE TO FIND ALL PRIME NUMBERS LESS
        // THAN OR EQUAL TO max_val
        // Create a boolean array "prime[0..n]". A
        // value in prime[i] will finally be false
        // if i is Not a prime, else true.
        Boolean[] prime = new Boolean[max_val + 1];
        for (int i = 0; i < max_val + 1; i++)
        {
            prime[i] = true;
        }

        // Remaining part of SIEVE
        prime[0] = false;
        prime[1] = false;
        for (int p = 2; p * p <= max_val; p++)
        {

            // If prime[p] is not changed, then
            // it is a prime
            if (prime[p] == true)
            {
```

```
                // Update all multiples of p
                for (int i = p * 2; i <= max_val; i += p)
                {
                    prime[i] = false;
                }
            }
        }

        // Find all primes in arr[]
        int count = 0;
        for (int i = 0; i < n; i++)
        {
            if (prime[arr[i]])
            {
                count++;
            }
        }

        return count;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = {1, 2, 3, 4, 5, 6, 7};
        int n = arr.length;
        System.out.println(primeCount(arr, n));
    }
}
```