ASSIGNMENT 7.1

1)

i)For loop:-

Java for loop consists of three factors which are an initialization statement, a testing condition, and an increment or decrement part for incrementing/decrementing the control variable.
The basic syntax of java for loop goes like this:

```
for(initializing statement; testing condition; increment/decrement)
{
    //code to be iterated
}
```

The initializing statement marks the beginning of the loop structure. It contains a variable with some initial value that is pre-defined. This statement is executed only once.

For example:-

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

OUTPUT:-

```
Hello World 0
Hello World 1
Hello World 2
Hello World 3
Hello World 4
Hello World 5
```

The value of i increases from 0 to 5 and then stops because the value of i becomes 6 and the condition becomes false. Hence it goes out of the loop.

ii) **While loop**:-

In a while loop, the condition is evaluated first and if it returns true then the statements inside the while loop will execute. When the condition returns false, the control comes out of the loop and jumps to the next statement after the while loop.

For example:

```
class Solution {
   public static void main(String args[]){
       int i=5;
       while(i>0){
           System.out.println(i);
           i--;
       }
   }
}
```

OUTPUT:

```
5
4
3
2
1
```

Here, Inside the body of the while loop, the value of i is decremented by using i--. So on every next iteration of the while loop, the loop checks whether the value of i >0, if yes it executes the Rest of the statements or else the loop ends. At the very last iteration of the while loop, the value of i is 0, loop checks the condition and it returns false as i<0 so the statements won't execute for i=0 and the loop ends.

iii) Continue:

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop. In a for loop, the continue keyword causes control to immediately jump to the update statement. In a while loop or do/while loop, control immediately jumps to the Boolean expression.

For example:-

```
public class Solution{

   public static void main(String args[]) {
      int [] nums = {1, 2, 3, 4, 5};

      for(int n : nums) {
         if( n == 3 ) {
            continue;
         }
         System.out.println( n );
      }
   }
}
```

OUTPUT:-

```
1
2
4
5
```

iv) IF:-

The Java if statement is a decision-making statement which is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

Syntax:-

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

For example:-

```
Class Solution {

int i = 1;
int j = 1;

if(i == j){

System.out.print(“True”);
}
}
```

OUTPUT:-
True

v) ELSE:-

The Java else statement is used to test a condition. It executes the *if block* if the condition is true otherwise else block, is executed. Else block must have a preceding If block.

Syntax:-

```
if(condition){
 //code if condition is true
  }else{
    //code if condition is false
   }
```

vi) SWITCH:-

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax

The syntax of enhanced for loop is –

```
switch(expression) {
   case value :
      // Statements
      break;
   case value :
      // Statements
      break;
   // You can have any number of case statements.
   default :      // Statements
}
```

For example:-

```
public class Test {
   public static void main(String args[]) {
      // char grade = args[0].charAt(0);
      char grade = 'C';
      switch(grade) {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println(" Passed");
```

```
        case 'F' :
           System.out.println("Better try again");
           break;
        default :
           System.out.println("Invalid grade");
     }
     System.out.println("Your grade is " + grade);
  }
}
```

vii) **Recursion**

A method that calls itself is known as a recursive method. And, this process is known as recursion. A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

For example:-

```
class Factorial {
   static int factorial( int n ) {
      if (n != 0)  // base condition
         return n * factorial(n-1); // recursive call
      else
         return 1;
   }
   public static void main(String[] args) {
      int number = 4, result;
      result = factorial(number);
      System.out.println(number + " factorial = " + result);
   } }
```

OUTPUT:-

4 factorial = 24

viii) **Binary search tree**:-

A binary Search Tree is a node-based binary tree data structure which has the following properties:

- *The left subtree of a node contains only nodes with keys lesser than the node's key.*
- *The right subtree of a node contains only nodes with keys greater than the node's key.*
- *The left and right subtree each must also be a binary search tree.*
  There must be no duplicate nodes.

To check whether a tree is a BST:-

```
public class BinaryTree {


   static class Node {


      //instance variable of Node class


      public int data;


      public Node left;


      public Node right;


  //constructor
```

```
        public Node(int data) {
        this.data = data;
         this.left = null;
          this.right = null;
      }
  }


   // instance variable of binary tree class


    public Node root;


   // constructor for initialise the root to null BYDEFAULT


    public BinaryTree() {


        this.root = null;


    }


    // method to check the given tree is Binary search tree or not

      public boolean isBSTOrNot() {
       return isBSTOrNot(this.root, Integer.MIN_VALUE, Integer.MAX_VALUE);
}


    private boolean isBSTOrNot(Node root, int minValue, int maxValue) {
```

```
        // check for root is not null or not

        if (root == null) {

            return true;

        }

        // check for current node value with left node value and right node value and recursively check for left sub tree and right sub tree

        if(root.data >= minValue && root.data <= maxValue && isBSTOrNot(root.left, minValue, root.data) && isBSTOrNot(root.right, root.data, maxValue)){

            return true;

        }

        return false;

}    public static void main(String[] args) {

        // Creating the object of BinaryTree class

        BinaryTree bt = new BinaryTree();
```

```
        bt.root= new Node(100);

        bt.root.left= new Node(90);

        bt.root.right= new Node(110);

        bt.root.left.left= new Node(80);

        bt.root.left.right= new Node(95);

        System.out.println(bt.isBSTOrNot());

} }
```

ix) **Closure** :-

A closure is a function (or method) that refers to free variables in their lexical context. The function is a block of code with parameters. It may produce a result value (return type). A free variable is an identifier used but not defined by the closure. It can be used without being a method or a class. It means that closure can access variables that are not defined in the parameter list and are also assigned to a variable.

For example;-

```
@FunctionalInterface

public interface Operation
```

```
{
void operate(int n);
}
 ClosureExample.java
public class ClosureExample
{
public static void main(String args[])  {
int x=2;
int y=3;
doSum(x, new Operation()  {
@Override
public void operate(int n)  {
System.out.println("Sum is: "+(n+y));
}
});
}



private static void doSum(int i, Operation op)  {
op.operate(i);
}
}
```

OUTPUT:-

Sum is: 5

2)

```
function getElementById(id)
```

```
{
 let match = null;
const doFind = node => {
   if (!match && node.id === id) match = node;
   if (!match)
   return [...node.childNodes].find(doFind);
  }
doFind(document.body);
 return match;
 }
console.log(getElementById('subchild').innerHTML
);
<div id="parent">
<div id="child1">
 </div> <div id="child2">
<div id="subchild"> subchild! </div>
 </div> </div>
```

3)

```
const isPrime = num => {
for(let i = 2, s = Math.sqrt(num); i <= s; i++) {
  if(num % i === 0)
    return false; }
return num > 1;
}
```