1. The Fibonacci series

Starting with 0 and 1, the Fibonacci series is a series of elements in which the previous two elements are added to get the next element.Any number of integers from the Fibonacci series can be displayed using this programme. Each number in the Fibonacci sequence equals the sum of the two numbers before it.

```
package com.learning;
public class FibonacciSeries {
public static void Fibonacci(int N)
{
int num1 = 0, num2 = 1;
int counter = 0;
// Iterate till counter is N
while (counter < N) {
// Print the number
System.out.print(num1 +" ");
// Swap
int num3 = num2 + num1;
num1 = num2;
num2 = num3;
counter = counter + 1;
}
}
public static void main(String[] args) {
// Given Number N
int N = 10;
// Function Call
Fibonacci(N);
}
}
```

o/p

```
<terminated> FibonacciSeries [Java App
0 1 1 2 3 5 8 13 21 34
```

2. Checking for prime number

write a programme to determine whether a given number is prime or composite.

A prime number is greater than 1 and can only be divided by 1 or itself.

For instance, Prime numbers include 2, 3, 5, 7, 11, 13, and 17.

We'll take a number variable and verify whether it's prime or not in this Java programme.

```
package com.learning;
public class PrimeNumberExample {
public static void main(String[] args) {
int i, m = 0, flag = 0;
int n = 3;// it is the number to be checked
m = n / 2;
if (n == 0 || n == 1) {
System.out.println(n + " is not prime number");
} else {
for (i = 2; i <= m; i++) {
if (n % i == 0) {
System.out.println(n + "is not prime number");
flag = 1;
```

```
break;
}
}
if (flag == 0) {
System.out.println(n + " is prime number");
}
} // end of else
}
}
```

Output:

```
3 is prime number
```

3. String palindrome

It is a program that determines whether or not a given string is a palindrome. A palindrome is a string, which, when read forward and backward, means the same. Although to check whether or not a string is a palindrome, a string needs to be compared with its reverse.

So let us consider two strings, "12121" and "apple", now the task is to find out if its reverse string is the same as it is.

JAVA PROGRAM

```
package com.learning;
import java.util.Scanner;
public class StringPalindrome {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter string :");
String str = scanner.nextLine();
String reverseStr = "";
for (int i = str.length() - 1; i >= 0; i--) {
reverseStr = reverseStr + str.charAt(i);
}
if (str.equals(reverseStr)) {
System.out.println("String is palindrome");
} else {
System.out.println("String is not palindrome");
}
}
}
```

```
OUTPUT
Enter string :Malayalam
String is palindrome

Enter string :Good
String is not palindrome
```

4. Bubble sort

Bubble Sort is the most basic sorting algorithm, and it works by repeatedly swapping adjacent elements if they are out of order. However, because of its average and worst-case time complexity, this algorithm is unsuitable for large data sets.

In each iteration, each element of the array moves to the end of the array, similar to how air bubbles rise to the water's surface. That's why it's referred to as a bubble sort.

```
package com.learning;
import java.util.Arrays;
public class BubbleSort {
public static void main(String[] args) {
int[] data = { -2, 45, 0, 11, -9 };
// call method using class name
BubbleSort.bubbleSort(data);
System.out.println("Sorted Array in Ascending Order:");
System.out.println(Arrays.toString(data));
}
// perform the bubble sort
public static void bubbleSort(int array[]) {
int size = array.length;
// loop to access each array element
for (int i = 0; i < size - 1; i++)
// loop to compare array elements
for (int j = 0; j < size - i - 1; j++)
// compare two adjacent elements
// change > to < to sort in descending order
if (array[j] > array[j + 1]) {
// swapping occurs if elements
// are not in the intended order
int temp = array[j];
array[j] = array[j + 1];
array[j + 1] = temp;
}
}}
```

OUTPUT

```
Sorted Array in Ascending Order:

[-9, -2, 0, 11, 45]
```

5. Armstrong number

An Armstrong number is simply the sum of the cubes of its digits. This is a Java programme to verify whether or not a given number is an Armstrong number. As an input, you need to enter the integer to be checked. Then, you can use modulos and division operations, loops and if-else statements to get the output.
In the case of an Armstrong number of 3 digits, the sum of cubes of each digit is equal to the number itself. For example, 153 is an Armstrong number because
153 = 1*1*1 + 5*5*5 + 3*3*3

JAVA PROGRAM:

```
package com.learning;
import java.util.Scanner;
public class ArmStrongNumber {
public static void main(String[] args) {
int n, count = 0, a, b, c, sum = 0;
Scanner s = new Scanner(System.in);
```

```
System.out.print("Enter any integer you want to check:");
n = s.nextInt();
a = n;
c = n;
while (a > 0) {
a = a / 10;
count++;
}
while (n > 0) {
b = n % 10;
sum = (int) (sum + Math.pow(b, count));
n = n / 10;
}
if (sum == c) {
System.out.println("Given number is Armstrong");
} else {
System.out.println("Given number is not Armstrong");
}
}
}
```

OUTPUT:

```
Enter any integer you want to check:153

Given number is Armstrong

Enter any integer you want to check:67

Given number is not Armstrong
```

6.MERGE SORT

Merge Sort is one of the most effective sorting algorithms, operating on the divide-and-conquer principle. It is built on the notion of breaking down a list into several sub-lists, each of which contains a single element. Then merge those sublists in a way that produces a sorted list.

```
package com.learning;
public class MergeSort {
public static void main(String[] args) {
int[] arr = { 70, 50, 30, 10, 20, 40, 60 };
int[] merged = mergeSort(arr, 0, arr.length - 1);
for (int val : merged) {
System.out.print(val + " ");
}
}
public static int[] mergeTwoSortedArrays(int[] one, int[] two) {
int[] sorted = new int[one.length + two.length];
int i = 0;
int j = 0;
int k = 0;
while (i < one.length && j < two.length) {
if (one[i] < two[j]) {
sorted[k] = one[i];
```

```
k++;
i++;
} else {
sorted[k] = two[j];
k++;
j++;
}
}
if (i == one.length) {
while (j < two.length) {
sorted[k] = two[j];
k++;
j++;
}
}
if (j == two.length) {
while (i < one.length) {
sorted[k] = one[i];
k++;
i++;
}
}
return sorted;
}
public static int[] mergeSort(int[] arr, int lo, int hi) {
if (lo == hi) {
int[] br = new int[1];
br[0] = arr[lo];
return br;
}
int mid = (lo + hi) / 2;
int[] fh = mergeSort(arr, lo, mid);
int[] sh = mergeSort(arr, mid + 1, hi);
int[] merged = mergeTwoSortedArrays(fh, sh);
return merged;
}
}
```

OUTPUT:

```
10 20 30 40 50 60 70
```

7. FACTORIAL

Factorial is one of the simplest and most popular Java programs and is commonly asked in Java interviews and other programming languages such as C or C++. For example, the interviewer may examine converting a recursive solution to an iterative one or vice versa.

The factorial of a number is calculated using the formula number*(number - 1) until zero. Because the value of factorial zero is 1, it serves as the base case in the recursive version of the factorial method.

Also, let’s look at a complete code example of a Java programme that calculates a number’s factorial in both recursive and iterative ways.

However, if you look closely, you will notice that the recursive solution to calculating factorial is much easier to write and read because it represents the formula number* (number -1).

```
package com.learning;
import java.util.Scanner;
public class Factorial {
public static void main(String[] args) {
Scanner obj= new Scanner(System.in);
System.out.println("Enter the number to find factorial");
int num=obj.nextInt();
//int factorial = factorialusingrecursion(num);
int fact = fact(num);
//System.out.println("Factorial of "+num+" is "+factorial);
System.out.println("Factorial of "+num+ " is "+fact);
}
/*
* Java program example to find factorial of a number using recursion
* @return factorial of a number
*/
public static int factorialusingrecursion(int number){
//base case
if(number == 0){
return 1;
}
return number*factorialusingrecursion(number -1); //is this tail-recursion?
}
/*
* Java program example to calculate factorial using while loop or iteration
* @return factorial of a number
*/
public static int fact(int number){
int result = 1;
while(number != 0){
result = result*number;
number--;
}
return result;
}
}
```

OUTPUT:

```
Enter the number to find factorial

3

Factorial of 3 is 6
```