IT-29006

1.
```
import java.io.*;
import java.util.Scanner;
public class FileNumberSum {
    public static void main (String args[])
                                throws exception {
    Scanner sc=new Scanner(new File("input.txt"));
    sc.useDelimiter(",");
    while(sc.hasNextInt()) {
        int n = sc.nextInt();
        long sum=(long)n*(n+1)/2;
        pw.print(sum);
    if(sc.hasNext()) pw.print(",");
        }
        sc.close();
        pw.close();
    }
}
```

2. Difference between static and final:

| Static | final |
|---|---|
| 1. Static belongs to class | 1. final belongs to variable/ Method/ Class |
| 2. Value change allowed | 2. Value change not allowed |
| 3. Override | 3. No override |
| 4. Memory single copy | 4. Depends. |

```
class Test {
    static int x = 10;
    }
Test t = new Test();
System.out.println(t.x); // it will work but not recommended.

System.out.println(Test.x); // correct.
```

## 3. Factorian Number Program:

```
import java.util.Scanner;
public class Factorian {
    static int fact(int n) {
      int f = 1;
   for (int i = 1; i <= n; i++)
       f = f * i;
       return f;
    }

  static boolean isFactorian(int n) {
       int sum = 0, temp = n;
      while (temp > 0) {
          sum += fact(temp % 10);
            temp /= 10;
          }
       return sum == n;
          }
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int low = sc.nextInt();
    int high = sc.nextInt();
    for(int i = 1; i <= high; i++) {
        if (isFactorian(i))
        System.out.println(i + " ");
        }
    }
```

Output: 1 2 145 40585

4. Distinguish the differences among class, local and instance variables:

| class | local | instance |
|---|---|---|
| 1. It declared inside class and outside method. | 1. It declared inside class and outside methods. | 1. It declared inside a method, block or constructor. |
| 2. Static keyword | 2. No keyword | 2. No keyword. |
| 3. Memory stored in class area | 3. Memory stored in heap | 3. Memory stored in stack. |
| 4. Scope accessible by all objects. | 4. Unique to each object. | 4. Only ~~while~~ within method / block |

5. Sum of Array Elements:

```
public class ArraySum {
    static int sum(int[] arr) {
        int s=0;
        for (int x:arr)
            s+=x;
        return s;
    }
public static void main (String [] args) {
    int [] a= {1,2, 3, 4};
    System.out.println (sum(a));
    }
}
```

6. An access modifier in Java is a keyword that controls the visibility and accessibility of classes, methods, variables and constructors. It determines from where and by whom a class member can be accessed.

- Types of access modifier in JAVA:

1. Public → Same class, same package, subclass.
2. Private → Same class, different package.
3. Protected → Same class, same package, subclass.
4. ~~Private~~
4. Default → Same class, same package, No subclass.

IT-24006

## 7. Quadratic Equation:

```
import java.util.Scanner;
public class Quadratic{
  public static void main(String[] args){
    Scanner sc=new Scanner(System.in);
     Int a=sc.nextInt();
     Int b=sc.nextInt();
     Int c=sc.nextInt();
  double d=b*b-4*a*c;
   If(d<0){
    System.out.println("No real roots");
      }return;
  double r1=(-b+Math.sqrt(d))/(2*a);
  double r2=(-b-Math.sqrt(d))/(2*a);
  double min=Math.min(r1,r2);
  If(min>0)
   System.out.println("The smallest positive root is:"+
                                          min);
  else
  System.out.println("No positive roots");
   }
  }
```

8. • Letter, Digit, Whitespace check:

```
char c='A';
System.out.println(Character.isLetter(c));
System.out.println(Character.isDigit(c));
System.out.println(Character.isWhitespace(c));
```

• Passing array to function:

```
Static void print(int[] a){ }
```

9. Method overriding occurs when a subclass provides its own implementation of a method that already exists in its superclass using same method name, same parameter list, same or covariant return type. It supports runtime polymorphism.

```
class parent{
  void show(){
    System.out.println("Parent method");
  }
}
class child extends Parent{
    void show(){
    System.out.println("Child method");}}
public class Test{
    public static void main(String[] args){
        Parent obj = new Child();
        obj.show(); }}
```

10. Diffences between static and non-static members

| Features. | Static member | Non-static member |
|---|---|---|
| Keyword | static | No keyword |
| Belongs to | class | object. |
| Memory | One shared copy. | Seperate copy per object. |
| Access | Using class name. | Using object. |
| Polymorphism | No | Yes. |

Static:
```
Class Demo{
    static int x=10;
    static void show(){
        System.out.println(x);
    }
}
```
Access: `Demo.show();`

Non-Static:
```
Class Demo{
    int y=20;
    void display(){
        System.out.println(y);
    }
}
```
Access:
```
Demo obj = new Demo();
obj.display();
```

IT-24006

## Palindrome check:

```
import java.util.Scanner;
public class PalindromeCheck{
 static boolean isNumberPalindrome(int num){
   int original=num, rev=0;
 while(num>0){
   rev =rev*10 +num%10;
    num/=10;
   }
  return original==rev; }
static boolean isStringPalindrome (String str){
   String rev=new StringBuilder(str).reverse().toString(
   return str.equalsIgnoreCase(rev); }
public static void main (String[] args){
   Scanner sc=new Scanner (System.in);
System.out.println("Enter number or string: ");
   String input=sc.next();
if (Character.isDigit (input.charAt(0))){
   int num = Integer.parseInt (input);
   System.out.println (isNumberPalindrome (num)?
          "Palindrome Number":"Not palindrome"); }
   else {
   System.out.println(isStringPalindrome (input)?
          "Palindrome String":"Not palindrome");}}
        sc.close();
      }
   }
```

IT-24006

11. Abstraction means hiding implementation details and showing only essential features to the users.

Example:

```
abstract class Shape {
    abstract void draw();
}
Class Circle extends Shape {
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

Encapsulation means binding data and methods together and protecting data using access modifiers.

Example:

```
Class Student {
    private int id;
    public void setId(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
}
```

- Difference between abstract class and interface:

| Feature | Abstract class | Interface |
|---|---|---|
| Methods | Abstract + concrete | Abstract (default allowed) |
| Variables | Instance variables | Public static final. |
| Inheritence | Single | Multiple |
| Constructors | yes | no |
| Keywords | extends | implements |

IT-24006

12. Inheritence Based Numerical Operations:

⇒
```
class BaseClass {
    void printResult(String msg) {
        System.out.println(msg);
    }
}

class SumClass {
    void calculateSum() {
        double sum = 0.0;
        for(double i = 1.0; i >= 0.1; i -= 0.1) {
            sum += i;
        }
        printResult("Sum = " + sum);
    }
}

class DivisionMultipleClass extends BaseClass {
    int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b); }
    void findGcdLcm(int a, int b) {
        int g = gcd(a, b);
        int l = ~~lcm(a,b)~~ (a * b) / g;
        printResult("GCD = " + g + ", LCM = " + l);
    }
}
```

IT-24006

```
Class NumberConversionClass extends BaseClass{
    void convert (int num) {
     printResult ("Binary : "+Integer.toBinaryString(num));
     printResult ("Octal : " +Integer.toOctalString (num));
     printResult ("Hex : " +Integer.toHexString(num));
        }
    }
Class CustomPrintClass { extends BaseClass {
     void pr (String msg) {
       printResult (">> " + msg);
        }
    }
public class MainClass {
    public static void main (String[] args) {
    new SumClass().calculateSum();
    new DivisionMultipleClass().findGcdLcm(12,18);
    new NumberConversionClass(). convert (25);
    new CustomPrintClass(). pr ("All operations done");
        }
    }
```

14. Significance of BigInteger:
⇒ Handles very large integers.
⇒ No size limit
⇒ Used in cryptography, factorials, scientific computing.

Factorial using BigInteger:

```
import java.math.BigInteger;
class Factorial {
    static BigInteger factorial(int n) {
        BigInteger f = BigInteger.ONE;
        for(int i=1; i<=n; i++) {
            f = f.multiply(BigInteger.valueOf(i));
        }
        return f;
    }
    public static void main(String[] args) {
        System.out.println("Factorial = " + factorial(50));
    }
}
```