IT 23013

Question-1:

Code:

```
import java.io.*;
import java.util.Scanner;
public class Main{
    public static void main(String[] args)
            throws FileNotFoundException{
        String inputfile="src/input.txt";
        String outputfile="src/output.txt";
        Scanner scanner=new Scanner
                (new File(inputfile));
        PrintWriter writer=new
                PrintWriter(outputfile);
```

IT 23043

```
while (scanner.hasNextLine()) {
    StringBuilder result = new StringBuilder();
    String ~~roco~~ roco = Scanner.nextline();
    String[] column = re roco.split(",");
    for(String S : column){
        String strim = S.trim();
     if (!strim.~~em~~isEmpty()){
        int number = Integer.parseInt(strim)
        int sum = number * (number+1)/2;
        result.append(sum);
        result.append(",");
        }
}
String answer = "";
is (result.isEmpty()){
    answer = result.substring
                (0, result.length()-2);
} write.println(String.valueOf(answer));
```

IT23043

```
        scanner.close();
        writer.close();
    }
}
```

Question-2:

Field: It's a variable declared inside a class or object that holds data or state.

Method: A block of code that defines a behavior or action that an object or class can perform

Static field: Belongs to the class itself, not any specific instance. Shared by all instances of the class

IT-23543

Static Method: Belongs to the class and can be called without creating an instance

- Top-level (outermost) class can't be static
- nested classes can be static

Final Field: Constant, can't be changed

Final Method: can't be overridden by subclass

- Top level class (outermost) can be final and can't be subclassed
  no one can inherit it, can create instance
  extends — compile time error.
- Nested → Static — without instance
  → Non-static with instance
  → no can can inherit

IT23043

Static + final field → Can't be modified but shared across all the instance

Static + final method → can be called without creating instance and can't be overridden.

In Java static field/method can be accessed through object Static belongs to class not instance so they should be accessed via classname.

IT-23043

Question- 3:

Code:

```
import java.io.*;
import java.util.Scanner;
public class Main {
    public static int factorial(int n){
        int fact= 1;
        for(int i=1; i<= n;i++){
            fact= fact x i;
        }
        return fact;
}
public static boolean isFactorian(int number){
    int originalNumber=number;
    int sum=0;
    while (number>0){
    int digit= number % 10;
    sum= sum+ factorial(digit);
    number= number/10;
```

JTC230413

```
    }
    return sum == originalNumber;
}
public static void main(String[] args){
    Scanner scanner= new Scanner(System.in);
    System.out.print("Enter the lower bound
                        of the range: ");
    int lowerBound= scanner.nextInt();
    System.out.print("Enter the upper bound
                        of the range: ");
    int upperBound= scanner.nextInt();
    System.out.println("Factorion numbers in the
                        range: ");
    for(int i=lowerBound; i<=upperBound;i++){
        if (isFactorion(i)){
            System.out.print(i+", ");
    }   }
    scanner.close();
}
```

IT-23043

Question-48

Class Variable (Static)	Instance Variable	Local Variable
Declared with static keyword	Declared inside class but outside any method/constructor/block, without static	Inside method/constructor/block
Belongs to class shared by all object	Belongs to an instance (object). Each object gets own copy.	Only inside method/constructor/block
Stored in method area (shared by all instance)	Heap memory part of object	~~Stack~~ Stack memory
Default value assigned (0, null, false ...)	→	must be explicitly initialized before use
Can have ~~access~~ public, private, protected & default ~~modifier~~ access	→	No access modifier
Exists as long as class loaded	as long as object exists	only exist when method executes

IT-23043

```
class Example{

    static int a=100; → class variable
    int b; → instance variable
    void hello(){
        int c=5; → local variable
    }

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

    Example obj=new Example();
    obj.b=50;
    obj.display();

    }
}
```

IT 23043

This keyword:

The this keyword in java refers to the current instance of a class It helps differentiate between ~~str~~ instance variables, method parameters or local variables with the same name.

1. Distinguish instance - local vars:

```
class Person{
    String name;
    static String fname;

    Person(String name){
        this.name = name.
        fname = name.
    }
```

can't use this in static

IT-23043

2. Call another constructor:

```
Class Demo {
    Demo() {
        this(10);
        System.out.println("Hello");
    }
    Demo(int x) {
        System.out.println(x);
    }
}
```

3. Return Current Object.

```
class Test {
    Test getobject() {
        return this;
    }
}
```

IT-23043

21. Pass Current object as arguments.

```
class A{
    void display(A obj){
        system.out.println("Method called");
    }
    void show(){
        display(this);
    }
}
```

IT23041 3

Question-5 :

```
import java.util.*;
public class Calculator{
    public static int sum(int[] numbers){
        int sum=0;
        for(int num:numbers){
            sum=sum+num;
        }
        return sum;
    }
    public static void main(String[] args){
        Scanner scanner=new Scanner
                        (System.in);
        System.out.print("Enter number of
                        elements: ");
        int n=scanner.nextInt();
        int[] numbers=new int[n];
        for(int i=0;i<n;i++){
            numbers[i]=scanner.nextInt();
        }
```

IT-23043

```
int result = calculateSum(numbers);
System.out.println(result);
scanner.close();
}
}
```

Question-6:

Access modifiers in Java define the scope (visibility) of variables, methods and classes. There are four main types:

1. Public - Accessible from anywhere
2. Private - Within same class
3. Protected - Same package and by subclass
4. Default - Same package

IT23043

Modifiere	Access within class	Access within package	Access in Subclass	Access outside packag.
Public	✓	✓	✓	✓
Ptrivate	✓	✗	✗	✗
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗

Types of variable:

Local → inside method/constructor

Instance → Belongs to class, ~~shared~~ availabale to all methods

Static → Shared among all objects

Final → Value can't be changed.

IT23043

Example :

```
class Example{

    int a=10; → instance
    static int b=20; → static

    void method(){
        int n=5; → local
    final int ~~final~~ x = 10; → final.
    }
```

IT23043

Question - 70

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        int a=scanner.nextInt();
        int b=scanner.nextInt();
        int c=scanner.nextInt();

    double d= b*b-4*a*c;
    if(d>0){
        double r1=(-b+Math.sqrt(d))/(2*a);
        double r2=(-b-Math.sqrt(d))/(2*a);
        if (r1==r2+1){
        if(r1>0 && r2>0){
            double m.
            System.out.println(Math.min(r1,r2));
```

IT23043

```
else if (r[1]>0){
    System.out.println(r[1]);
}
else if (r2>0){
    System.out.println(r2);
}else{
    System.out.println("no positive roots");
}
}
else if (d==0){
    double root = -b/(2*a);
    if (root>0){
        System.out.println(root);
    else{
        System.out.println("no positive
                            root");
```

```
else{
    System.out.println("No reals roots");
}
scanner.close();
}
}
```

Question-8

```
import java.util.*;

public class Main {
    public static void main(String[] args){
        Scanner scanner = new Scanner
                          (System.in);
        char ch = scanner.next().charAt(0);
        if(Chara
```

IT23043

```
String s= scanner.nextLin();
for(Char ch: s.to CharArray()){
    if (Character.isLetter(ch)){
        System.out.println ("Letter");
    }else if (Character.isDigit(ch) {
            Systemout.println("Diget");
    }else if (Character.isWhitespace(ch)){
        Sysetem.out.printfln("white space");
    }else{
        System.out.println("special char");
    }
}
scanner.close();
```

IT23043

Question-98

When a subclass overrides a method, java calls the overridden method in the subclass even when an object is referenced using a superclass type. (runtime polymorphism).

```
class Animal {  ⟹ Superclass
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}  ⟹ Subclass
class Dog extends Animal {
    @Override → Annotation
    void makeSound() {
        System.out.println("Dog Barks");
    }
```

IT23043

```
public class Test{
    public static void main(String[] args){
        Animal mypet= new Dog();
        mypet.makeSound();
    //dynamic dispatch
```

If the subclass wants to call the original method in superclass it can use 'super' keyword

Dog Class →

```
    void makeSound(){
        super.makeSound();
        System.out.println("Bark")
    }
}
```

Output → Animal makes Sound
Bark

IT 23043

Q4

Potential Issue:

- Constructors can't be overridden in java as they are not inherited
- A subclass constructor must call a superclass constructor if the superclass has no argument constructor.

```
class parent {
    Parent (int x) {
        SOUT {"Parent x"};
    }
}

child class extends Parent {
    Child () {
        super (10);
    }
}
```

* A method ~~can't~~ in the subclass can't have stricter access modifier that superclass.

Your can increase visibility but can't decrease it.

* Final method can't be overridden

* Static methods are not overridden, they are hidden

* Private method can't be inherited

* In overridding we can return only original return type

IT23043

Ques - 10:

Static	Non-Static
Belongs to class	To individual object
Allocated once in a memory	Allocated separately for each other.
Using class name	Requires instance of the class
changes affect all instance	only specific instance
Statin int count;	int age.
Usage-constants.	Behaivior, properties

```
import java.util.Scanner;
public class Main {
    static boolean isPalindrome(int num){
        int original = num, reversed = 0;
        while(num > 0){
            reversed = reversed * 10 + num % 10;
            num /= 10;
        }
        return original == reversed;
```

IT 23043

```
static boolean is palindrome(String str){
    str= str.toLowerCase();
    int left=0, right =str.length()-1;
    while (left <= right){
        if(str.charAt(left)!= str.charAt(right)){
            return false;
        }
        left++; right --;
    return true;
    }
}
public static void main(String[] args){
Scanner scanner= new Scanner(System.in);
int num=Scanner.nextInt();
if (isPalindrome (num)){
    System.out.println (num+" Palindrome");
```

```
else{
    System.out.println(num+"Not Palindrome");

scanner.nextLine();
String str=scanner.nextLine();
if (isPalindrome(str)){
    System.out.println(str+"Palindrome");
}else{
    System.out.println(str+"Not palindrome");
```

IT-23043

Question-11:

Abstraction: The process of hiding implementation details and showing necessary features of an object. Helps reducing complexity and increase code maintainability

```
abstract class Vehicle{
    abstract void start();
    void fuel(){
      System.out.println("Needs fuel");
    }
}

class car extends Vehicle{
    @override
      void start(){
        System.out.println("Starts")
    }
}
```

IT-23043

Encapsulation: The process of wrapping data and code (methods) together inside a single unit (class) and restricting direct access to some components.

- Achieved using private variables with public setter and getter.
- Ensure data security and control over modifications.

IT23043

Abstract Class	Interface
Can have both abstract + concrete method	only abstract class (JJ) / abst default (J8+)
Can have abstract, non-abstract method	J8 – all abstract J8, J8+ both default, Stati
public, protected, private	public by default
Can have instance variable contants	only public static final
has constructer	no constructer
Supports ~~sig~~ single inheritance	multiple inheritance

IT23043

Question 14:

In Java, int → 32 bit, long → 64 bit

When working with very large number normal data types overflow But for bigInteger it can store as large as available memory allows.

```
import java.math.*;
import java.util.*;
public class Main{
    public static BigInteger factorial(int n){
        BigInteger fact=BigInteger.ONE;
        for (i=2;i<=num;i++){
            fact=fact.multiply(BigInteger.
                valueOf(i));
        return fact;
}
```

IT23043

```
public static void main (String[] args){
    Scanner scanner = new Scanner
                      (System.in);
    System.
    int n = Scanner.nextInt();
    System.out.println (factorial(n));
    Scanner.close()
    }
}
```