Answer to the question no: 01

Java program for determine sum of series

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class SumSeries{
    public static void main(String[] args){
        String inputFile = "input.txt";
        String outputFile = "output.txt";
        String result = "";

    try{
        Scanner scanner = new Scanner(new File(inputFile));
        String line = scanner.nextLine();
        String[] number = line.split(",");
        for (int i=0; i< number.length; i++)
    { int num = Integer.parseInt(number[i].trim());
        int sum = (num * (num+1))/2;
        result += sum;
        if (i< number.length -1){
            result += ","; }
    }
```

```
        PrintWriter writer = new PrintWriter(outputFile);

        scanner
        writer.println(result);
        System.out.println("Result written to " + outputFile);

        scanner.close();
        writer.close();

    }
    catch (FileNotFoundException e)
    {
        System.out.println("Error: Input file not found!");

    }

}

}
```

Answer to the question no: 02

1. static Fields and Methods

Purpose:

1. static is used to define class-level members (fields or method) that belong to the class itself, rather than to any specific instance of the class.

Behaviour:

1. A static field is shared accross all instances of the class. There is only one copy of the field in memory, regardless of how many objects are created.

2. A static method can be called without creating an instance of the class it can only access static fields or other static methods directly.

Example:

```
class MyClass{
    static int staticField =10;
    static void staticMethod(){
        System.out.println("Static method called");
    }
}
```

2. final Fields and Methods

Purpose:
final is used to define constants (for fields) or to prevent overriding (for methods).

Behavior:
a. A final field cannot be modified after it is initialized. It must be assigned a value either at the time of declaration or in the constructor.

b. A final method cannot be overridden by subclasses.

Example:

```
class MyClass{
    final int finalField = 20;
    final void finalMethod(){
        System.out.println("Final method called");
    }
}
```

Feature	static	final
purpose	Defines class-level members shared across all instances.	Define constant or prevents modification/ overriding.
Field Behavior	Shared accross all instances; only one copy exists.	Cannot be modified after initialization.
Method Behavior	Can be called without an instance; cannot access non-static members.	Cannot be overridden in subclasses.
Memory	Allocated once in memory for the class.	Allocated per instance (for non-static fields).

Accessing static Methods/Fields with an Object what happens?

- You can access static methods and fields using an object, but it is not recommended because it can lead to confusion.
- The jave compiler allows it, but it is considered bad practice. The IDE or compiler may even issue a warning.

Example :

```
class MyClass{
    static int staticField = 10;
    static void staticMethod(){
        System.out.println("static method called");
    }
}
public class Main{
    public static void main(String[] args){
        MyClass obj = new MyClass();
        System.out.println(obj.staticField);
        obj.staticMethod();
        System.out.println(MyClass.staticField);
        MyClass.staticMethod();
    }
}
```

Answer to the question no: 3

Java program for find all factorion numbers in a given range

```
import java.util.Scanner;
public class Factorion{
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Enter lower bound: ");
        int low = input.nextInt();
        System.out.println("Enter upper bound: ");
        int up = input.nextInt();
        System.out.println("Factorion numbers in the
        range:");
        boolean found = false;
        for(int i = low; i<=up; i++){
            if(isFactorion(i)){
                if(found){
                    System.out.print(",");
                }
                System.out.print(i);
                found = true;
            }
        }
```

```
        if(!found){
            System.out.println("No factorion numbers found
            in this range.");
        }
        input.close();
    }
    private static boolean isFactorion(int number){
        int sum = 0;
        int temp = number;
        while(number>0){
            int digit = number%10;
            sum += factorial(digit);
            number/=10;
        }
        return sum==temp;
    }
    private static int factorial(int n){
        if(n==0 || n==1){
            return 1;}
        int result =1;
        for(int i=2;i<=n;i++){
            result*=i;
        }
        return result;
    }
}
```

Answer to the question no: 4

1. Class Variables

Definition: Class variables are declared with the static keyword and belong to the class itself, not to any specific instance of the class.

Scope: They are accessible throughout the class and can also be accessed using the class name.

Lifetime: Class variables are created when the class is loaded into memory and destroyed when the class is unloaded. They exist for the entire duration of the program.

Memory Allocation: Only one copy of a class variable is shared among all instances of the class.

Example:

```
class MyClass{
    static int classVariable = 10;
}
```

2. Instance Variables

Definition : Instance variables are declared inside a class but outside any method, constructor, or block. They belong to an instance of the class.

Scope: They are accessible within the class and can be accessed using an object of the class.

Lifetime: Instance variables are created when an object of the class is instantiated using the new keyword and destroyed when the object is garbage collected.

Memory Allocation: Each object of the class has its own copy of the instance variables.

Example ;

```
Class MyClass{
    int instanceVariable;

}
```

3. Local Variables :

Definition: Local variables are declared inside a method, constructor, or block. They are temporary and exist only during the execution of the method, constructor or block.

Scope: They are accessible only within the block in which they are declared.

Lifetime: Local variables are created when the method, constructor, or block is executed and destroyed when the execution completes.

Memory Allocation: Local variables are stored in the stack memory.

Example :

```
class MyClass {
    void myMethod() {
        int localVariable = 20;
    }
}
```

significance of this keyword

The this keyword in Java is a reference to the current object of the class. It is primarily used to:

1. Differentiate Between instance Variables and Local Variables:

when a local variable has the same name as an instance variable, this is used to refer to the instance variable.

Example:

```
class MyClass{
    int a;
    void setA(int a){
        this.a = a;
    }
}
```

2. Invoke Current Class Constructor

this() can be used to call one constructor from another constructor in the same class (constructor chaining).

Example:

```
class MyClass{
    MyClass(){
        this(10);
    }
    MyClass(int x){
        System.out.println("Value: "+x);
    }
}
```

3. Pass Current Object as a Parameter:

this can be passed as an argument to a method or constructor to represent the current object.

Example:

```
class MyClass{
    void display(){
        print(this);
    }
    void print(MyClass obj){
        System.out.println("Printing object");
    }
}
```

4. Return Current Object:

this can be used to the current from a method.

```
class MyClass{
    MyClass getObject(){
        return this;
    }
}
```

Answer to the question no: 5

```
public class ArraySum {
    public static int calculateSum(int[] numbers){
        int sum = 0;
        for(int num : numbers){
            sum += num;
        }
        return sum;
    }
public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};
        int sum = calculateSum(numbers);
        System.out.println("Sum of array elements :" +sum);
    }
}
```

Explanation:

1. calculateSum(int[] numbers)

a. Takes an integer array as input

b. Uses a for-each loop to through the array and calculate the sum.

c. Returns the total sum.

2. main Method

a. Defines an example array {10, 20, 30, 40, 50}

b. Calls calculateSum(numbers).

c. print the result.

IT-23054

Answer to the question no: 6

Access modifiers in java define the scope and visibility of classes, methods, and variables, Java provides four access modifiers:

1. Public - Accessible from everywhere.
2. Private - Accessible only within the same class.
3. Protected - Accessible within the same package and subclasses.
4. Default (No modifier) - Accessible only within the same package.

Comparison of Public, Private and Protected Modifiers

Access Modifier	Accessible in the same class	Accessible in the same package	Accessible in sub-classes (inheritance)	Accessible in other packages
public	Yes	Yes	Yes	Yes
private	Yes	No	No	No
Protected	Yes	Yes	Yes	No (only in subclasses)
Default	Yes	Yes	No	No

Types of Variables in Java

In Java, variables are categorized into three types based on their scope and lifetime:

1. Instance Variables:

- Declared inside a class but outside any method.
- Belong to an instance of the class (object).
- Each object has its own copy of instances variables

Example:

```
public class MyClass {
    int instanceVar = 10;
}
```

2. static Variables:

- Declared with the static keyword.
- Belong to the class, not to any specific instance.
- Shared among all instances of the class.

Example:

```
public class MyClass {
    static int staticVar = 20;
}
```

3. Local Variables

- Declared inside a method, constructor, or block
- ~~Assenb~~ Accessible only within the method/block when they are declared.
- Must be initialized before use.

Example:

```
public class MyClass{
    public void myMethod(){
        int localVar = 30;
        System.out.println(localVar);
    }
}
```

Answer to the question no: 7

```
import java.util.Scanner;

public class QuadraticEquation{
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter coefficients a, b, and c: ");
    int a = scanner.nextInt();
    int b = scanner.nextInt();
    int c = scanner.nextInt();
    double discriminant = b*b - 4*a*c;
;f (discriminant < 0){
    System.out.println("No real roots.");
}else{
    double sqrtDiscriminant = Math.sqrt(discriminant);
    double root1 = (-b + sqrtDiscriminant)/(2*a);
    double root2 = (-b - sqrtDiscriminant)/(2*a);
;f(root1>0 && root2>0){
    double smallestRoot = Math.min(root1, root2);
    System.out.println("The smallest positive root
      is: " + smallestRoot);
    }
```

```
else if (root1 > 0){
    System.out.println("The smallest positive root is: " + root1);
  } else if (root2 > 0){
    System.out.println("The smallest positive root is: " + root2);
  }
  else {
    System.out.println("No positive roots.");
    }
  }
  scanner.close();
}
}
```

Answer to the question no: 8

~~public class ArrayPassingExample {~~

Java Program to determine Letters, whitespace and Digits.

```
import java.util.Scanner;
public class CharacterCounter{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a string: ");
        String input = sc.nextLine();
        int letters = 0, digits = 0, spaces = 0;
for(char ch : input.toCharArray()){
    if(Character.isLetter(ch)){
        letters++;
    } else if(Character.isDigit(ch)){
        digits++;
    }
    else if(character.isWhitespace()){
        spaces++;
    }
}
System.out.println("Letters: " + letters);
System.out.println("Digits: " + digits);
```

```
        System.out.println("Whitespace: "+spaces);

        scanner.close();
    }
}
```

Passing an Array to a function in Java

In Java, arrays are passed to function by reference. This means that any changes made to the array inside the function will affect the original array.

Example:

```
Public class ArrayExample{
    public static void main(String[] args){
        int[] numbers = {1,2,3,4,5};
        printArray(numbers);
        multiplyByTwo(numbers);
        System.out.println("Modified Array: ");
        printArray(numbers);
    }
    public static void printArray(int[] arr){
        for(int num: arr){
            System.out.println(num + " ");
        }
    }
```

```
    public static void multiplyByTwo(int[] arr){
        for(int i=0; i<arr.length; i++){
            arr[i] *= 2;
        }
    }
}
```

Explanation :

1. Array declaration: An array numbers is initialized with values {1, 2, 3, 4, 5}.

2. Passing Array to Function:

- The printArray function is called to print the array.
- The multiplyByTwo function is called to modify the array by multiplying each element by 2.

IT-23054

Answer to the question no-09

Method Overriding:

Method Overriding is a feature in java that allows a subclass to provide a new implementation for a method that is already define in its superclass. How its works in inheritance —

(i) When a subclass overrides a method, the subclass version of the method gets executed, even if the method called on a superclass reference holding a superclass object.

(ii) The process is called runtime polymorphism, because the method call is resolved at runtime.

(iii) Overriding enables customized behavior for subclass object while maintaining a consistent interface.

What happens when a subclass overrides a method -

1. Subclass method executes, replacing the superclass method.
2. Runtime polymorphism determines method execution at runtime.
3. Superclass method is hidden unless called using super.

4. Overriding rules apply.

5. super can call the overridden superclass method.

The super keyword is used to call an overridden method from the superclass. This allows the superclass to extend or modify the behavior without completely replacing it.

1. Potential issue when overriding methods:

(i) Visibility Restriction - Cannot reduce access (eg. public → private)

(ii) Exception Limitation - Cannot throw broader exception.

(iii) Final & static methods - fina methods can't be overridden.
static methods are hidden, not overridden.

2. Issue with constructors:

(i) Constructors cannot be overridden because they are not inherited.

(ii) Super() must be used for superclass initialization.
• If the superclass has a parameterized constructor, the supclass must explicitly call.
super(arguments);

.If no explicit super() is used, Java insents a default constructor call, which may cause an error if the super class lacks a no angument constructor.

Answer to the question no: 10

Difference Between static and Non-static Members in Java

Feature	static Members	Non-static Member
Pefirition	Belong to the class rather than instances (object).	Belong to an instance of the class
Access	Can be accessed using the class name (ClassName.member).	Can only be accessed through an object of the class
Memory Allocation	Memory is allocated once per class	Memory is allocated separately for each object.
Modification	Changes apply to all instances of the class	Changes are instance-specific
usage	Useful for shared resources, constants, and utility methods	Used for instance-specific behaviours and properties.
Example	static int count; static void display()	int age, void show().

IT-23054

Check if a number or string is Palindrome -

```
import java.util.Scanner;
public class PalindromeChecker{
    static boolean isPalindrome(String str)
        String reversed= new StringBuilder(str).reverse().
        toString();
        return str.equalsIgnoreCase(reversed);
}
static boolean isPalindrome(int num){
    int original = num, reversed=0, remainder;
while (num>0){
    remainder= num%10;
    reversed = reversed*10+remainder;
    num /=10;
}
return original==reversed;
}
public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
System.out.println(num+(isPali "Enter a number : ");
int num = sc.nextInt();
System.out.println(num+(isPalindrome(num)?" is a
    Palindrome.":" is not a Palindrome"));
```

```
        sc.nextLine();
        System.out.println("Enter a string:");
        String str = sc.nextLine();
        System.out.println("
        System.out.println(" str +(isPalindrome(str)? " is a Palindrom
                        : "is not a Palindrome."));

        sc.close();
    }
}
```

IT-23054

Answer to the question no-11

(a) Class Abstraction:

Class abstraction is the process of simplifying complex reality by modeling only the essential atributes and behaviors of an object while hiding unnecessary details, It focuses on "what" an object does rather than how it does.

Example: A vehicle class may define as abstract method start(), but each subclass(e.g. car, bike) provides its own implementation.

(b) Class Encapsulation:

Encapsulation is the building of data (attributes) and methods (behavior) that operate on that data within a single unit. It also involves controlling the access to the internal data of an object, preventing direct modification from the outside. This is often achived through access modifiers like private, protected and public.

Example: A BankAccount class has a balance variable marked private and users can access it only through getbalance() and deposit() methods. These methods can include logic to validate the operations and maintain the integrity of the data.

IT-23054

Difference between abstract class and Interface -

Abstract Class	Interface
1. Abstract class can have both abstract and concrete methods.	1. Interface contains only abstract methods
2. Can have instance variables with an access modifier.	2. can have only public, static, final constants (no instance-variables).
3. Can have constructors.	3. Cannot have constructors.
4. Can include both implemented and non-implemented methods.	4. All methods are public and abstract by default
5. Used when classes share common behavior and need some abstraction	5. Used when different class must follow a common contract but share no implementation
6. Example: abstract class Animal{}	6. Example: Interface Animal{}