

Topic Name : _____

Day : _____

Time : _____

Date : / /

1. Write a Java program that reads a series of numbers from a file `input.txt` determines the highest number in the series, calculates the sum of natural numbers up to the highest number, and write the result another file `output.txt`. Use `Scanner` to read from the file and `PrintWriter` to write to the file. Assume the numbers in the input file separately.

Sample Input.txt:

10, 55, 1000, -----, 100

Output.txt:

55, 1540, 500500, -----, 5050

The code is given below:

Code:

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

public class ReadFile_SumSeries {
    public static void main(String[] args) {
        try {
            File infile = new File("input.txt");
            Scanner in = new Scanner(infile);
            PrintWriter out = new PrintWriter(new File("output.txt"));

            if (in.hasNextLine()) {
                String line = in.nextLine();
                String[] nums = line.split(",");
                for (int i=0; i<nums.length; i++) {
                    int n = Integer.parseInt(nums[i]);
                    int sum = (n*(n+1))/2;
                    out.print(sum);
                    if (i!=nums.length-1) {
                        out.print(",");
                    }
                }
                in.close();
                out.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Topic Name : _____ Day : _____
Time : _____ Date : / /

```
System.out.println("File written successfully.");  
} catch (Exception e){  
    System.out.println("file not found.");  
}  
}  
}
```

2. what are the difference you have even found between static and final fields and method
Exemplify what will happen if you try to access the static method/field with the object instead of class name.

Ans:

The difference between static and final field is given below:

| | |
|---|---|
| Feature static | final |
| Definition Belong to the class, not instances | Once assigned, cannot be changed |
| Fields Shared among all instances | Value cannot be modified once initialized |
| Methods Belongs to the class, can be called without an instance | Cannot be overridden by subclasses |
| Usage Uses for constants, utility methods and share data | Used to enforce immutability and prevent method overriding |

Example of static and final field:

```
class Example {  
    static int staticVar = 10;  
    final int finalVar = 20;  
  
    static void staticMethod() {  
        System.out.println("Static Method called");  
    }  
  
    final void finalMethod() {  
        System.out.println("Final Method called");  
    }  
}
```

```
public class Test {  
    public static void main( String[] args ) {  
        Example obj = new Example();  
  
        obj.staticMethod();  
        System.out.println( Example.staticVar );  
    }  
}
```

Topic Name : _____ Day : _____
Time : _____ Date : / /

If I Access static Members Using an Object

1. It compile and run successfully but not recommended. Java allows calling static member using and object reference, but it gives a warning because static members belong to the class not the object.
2. Final Field behave normally finalVar must be initialized either during deelaration or in the constructor.
3. Final method can be accessed normally, but cannot be overrride in subclass.

Topic Name : _____

3. Write a java program to find all factorion ~~num~~ numbers within a given range. A number is called a factorion if the sum of the factorials of its digits equals the number itself. The program should take user input for the lower and upper bound.

Code :

```
import java.util.Scanner;  
  
public class FactorionFinder {  
    public static void main (String [] args) {  
        Scanner scanner = new Scanner (System.in);  
        System.out.print ("Enter lower bound: ");  
        int lower = scanner.nextInt();  
        System.out.print ("Enter upper bound: ");  
        int upper = scanner.nextInt();  
        scanner.close();  
        for (int i=lower; i<=upper; i++) {  
            if (isFactorion(i)) System.out.print (i + " ");  
        }  
    }  
}
```

TT-23024

Topic Name :

private static boolean isFactorion (int num) {

 int sum = 0, temp = num;

 while (temp > 0) {

 sum += factorial (temp % 10);

 temp /= 10;

}

 return sum == num;

}

private static int factorial (int n) {

 int[] fact = { 1, 2, 6, 24, 120, 720, 5040,

 40320, 362880};

 return fact[n];

}

}

Topic Name:

Day:

Time:

Date: / /

4. Distinguish the difference among class, local, and instance variables. What is significance of this keyword?

| Feature | Class variable | Instance variable | Local variable |
|----------------|--|--|--|
| Definition | A variable defined with the static keyword inside a class, shared by all instances of the class. | A variable inside a class but outside any method unique to each object instance. | A variable defined inside a method, constructor, or block. |
| Scope | Class-wide, accessible to all instances of the class. | Object-specific, exists as long as the object exists. | Method or block-specific, exists only during the method execution. |
| Lifetime | As long as the class is loaded in memory | As long as the object instance exists | Limited to the execution time of the method block |
| Memory | Stored in the static memory | Stored in the heap part of the object instance | Stored in the stack memory during method execution. |
| Initialization | Initialize when the class is loaded | Initialize when the object is created | must be initialized before use. |

The significance of the this keyword:

Definition: The this keyword is a reference variable referring to the current instance of the class. It is often used inside instance methods or constructors to refer to the current object.

Uses:

1. Distinguishing Instance variable: It helps distinguish instance variable from parameters or local variable that have the same name.
2. Accessing Instance Methods/ Variables: It allows access to instance variables and methods.
3. Passing current object: It can be used to pass the current object as a parameter to another method or constructor.

Example:

```
class MyClass {
    int instanceVar;
    MyClass(int instanceVar) {
        this.instanceVar = instanceVar;
    }
}
```

5. Write a java program that defines a method to calculate the sum of all elements in an integers array. The method should take an integers array as parameter and return the sum. Demonstrate this method by passing an array of integers from the main method;

code:

Q. What is called access modifiers? compare the accessibility of Public, Private and protected modifier desribble the different type of variable in java with example.

An access modifier in Java is a keyword used to specify the visibility or accessibility of classes, methods, variables, or constructors.

The main access modifiers are:

1. public
2. private
3. protected
4. default

| Modifier | Accessibility | Description |
|-----------|--|---|
| public | Accessible from anywhere | Members declared as public and can be accessed from anywhere |
| private | Accessible only within the same class | Members declared as private not accessible from anywhere. They can only be accessed within the class they are defined |
| protected | Accessible within the same package and by subclass | members declared as protected |

Variable one:

1. Instance variable
2. Class variable
3. Local variable
4. Parameter

1. Instance variable: A variable defined in a class but outside methods. Each object of the class has its own copy.

Code:

```

class Car {
    String model;
    void displayModel() {
        System.out.println("Model! " + model);
    }
}

public class Test {
    public class Test {
        static void main(String[] args) {
            Car car1 = new Car();
            car1.model = "Toyota";
            car1.displayModel();
        }
    }
}

```

Topic Name :

Day : _____
Time : _____ Date : / /

2. class variable:

A variable declared with the static keyword.
Shared by all instances of the class.

Example:

```
class Can {  
    static int count = 0;  
  
    Can()  
        count++;  
  
    }  
  
}  
  
public class Test {  
    public class static void main (String[] args) {  
  
        new Can();  
        new Can();  
        System.out.println ("Total cans: " + Can.count)  
  
    }  
  
}
```

Topic Name :

Date:

Time:

Grade:

3. local variable: A variable declared inside a method or block, only accessible within that method or block.

Example:

```
class Example {  
    void addNumbers () {  
        int num1 = 5, num2 = 10;  
        int sum = num1 + num2;  
        System.out.println ("Sum: " + sum);  
    }  
}  
  
public class Test {  
    public static void main (String [] args) {  
        Example obj = new Example ();  
        obj.addNumbers ();  
    }  
}
```

Topic Name : _____

Day : _____

Time : _____

Date : / /

4 Parameters: A variable passes to a method.
Only accessible within the method

```
class Example{  
    void printMessage( String message)  
    {  
        System.out.println(message);  
    }  
}  
  
public class Test{  
    public static void main( String[] args)  
    {  
        Example obj = new Example();  
        obj.printMessage("Hello, Java!");  
    }  
}
```

Topic Name :

Day :

Time :

Date : / /

7. Write a Java program to find the smallest positive root of a quadratic equation of the form.

$$ax^2 + bx + c = 0$$

Using the quadratic formula,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program should:

1. Take integer input for coefficient a, b and c.
2. Compute the root using `Math.sqrt(double a)`.
3. Determine the smallest positive root using `Math.min(double a, double b)`
4. Print the smallest positive root if real root exist. otherwise print "No real roots".

Sample Input: a, b and c: 1, -3, 2

Sample output:

The smallest positive root is: 1.0;

Code:

```

import java.util.Scanner;

public class QuadraticEquation {
    public static void main (String [] args) {
        Scanner scanner = new Scanner (System.in);
        System.out.print ("Enter the co-efficient a, b and
c: ");
        int a = scanner.nextInt (), b = scanner.nextInt (),
c = scanner.nextInt ();

        double discriminant = b * b - 4 * a * c;
        if (discriminant >= 0) {
            double root1 = (- b + Math.sqrt (discriminant)) / (2 * a);
            double root2 = (- b - Math.sqrt (discriminant)) / (2 * a);
            if (root1 > 0 && root2 > 0)
                System.out.println ("The smallest positive root is: "
+ Math.min (root1, root2));
            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 real roots.");}  
}  
scanner.close();  
}
```

8. write a program that can determine the letter, whitespace and digit. How do we pass an array to function? write an example:

Code:

Topic Name :

```
public class CharacterType {
    public static void determineCharacterType
        (char[] chars) {
        for (char c : chars) {
            if (Character.isLetter(c)) {
                System.out.println(c + " is a letter.");
            } else if (Character.isDigit(c)) {
                System.out.println(c + " is a digit.");
            } else if (Character.isWhitespace(c)) {
                System.out.println("white space detected.");
            }
        }
    }

    public static void main (String[] args) {
        char[] characters = { 'A', ' ', '5', 'b' };
        determineCharacterType (characters);
    }
}
```

Q. In Java, explain how method overriding works in the context of inheritance. What happens when a subclass overrides a method from its superclass. How does the superclass help calling superclass method, what are the potential issues when overriding methods, especially when dealing with constructors.

Method overriding allows a subclass to provide a specific implementation of a method already defined in its superclass. This is a core feature of inheritance in object-oriented programming and is essential for achieving runtime polymorphism.

How super Helps:

The `super` keyword is used to call a method or constructor from the superclass.

`super.methodName()` is used to call the superclass's overridden method from subclass.

This is useful when the subclass wants to add extra behaviour to superclass method instead of completely replacing it.

37-29024

Topic Name :

Day:

Time:

Date:

10. Differentiate between static and non-static members including necessary exampk. write a program that able to check either a number or string is palingdom or not.

| Aspect | static members | Non-static members |
|-------------------|-------------------------------------|-------------------------------------|
| memory allocation | Allocate once for the class | Allocate separately for each object |
| Access | Accessed using class name | Accessed using an object |
| Scope | Shared across all instances | unique to all instances |
| Usage | class-level functionality constants | Instance level functionality |

check a numbers for palingdom or no

11. what is called class abstraction and encapsulation? describe with example. what are the differences between abstract a class and Interface?

Ans:

Abstraction:

Abstraction is the concept of hiding the internal implementation details of a class and only exposing essential features. It allows the user to focus on what object does rather than how it does it.

Example of Abstraction using an abstract class.

abstract class Vehicle {

 abstract void start();

}

class Car extends Vehicle {

 void start() {

 System.out.println("Car starts with a

 key");

Topic Name :

Day : _____
Date : _____

```

public class main {
    public static void main (String [] args) {
        Vehicle myCar = new Car();
        myCar.start();
    }
}

```

Encapsulation:

Encapsulation is the practice of wrapping data variables and code (methods) into a single, unit class and restricting direct access to some details.

Example:

```

class Person {
    private String name;
    public void setName
        this.name = name;
    }
    public String getName () {
        return name;
    }
}

```

```

public class Encap {
    public static void main( String[] args ) {
        Person p = new person();
        p.setName("John");
        System.out.println( p.getName() );
    }
}

```

Difference between Abstract class and Interface.

| Points | Abstract Class | Interface |
|-----------------|---|--|
| Definition | A class that contain abstract methods and may have concrete methods | A collection of abstract methods that must be implemented by any class that uses it. |
| Method | can have both abstract and non-abstract methods | can only have abstract method. |
| variable | can have instance variable | only public static and final variable |
| Access modifier | can have any access modifier such as private, protected | methods are always public by default. |

Topic Name :

Day :

Time :

Date :

12. Create a Java program using inheritance to perform multiple numerical operations. Implement a 'BaseClass' with common functionalities and extend it into four specialized classes. to handle different tasks. Use a 'Mainclass' to execute all method.

Answer:

```
class BaseClass {
    void print ( string msg, Object res )
    {
        System.out.print( msg + res );
    }
}
```

```
class SumClass extends BaseClass {
    double computeSum()
    {
        double sum = 0;
        for( double i=1; i >= 0.1; i -= 0.1)
            sum += i;
        return sum;
    }
}
```

Topic Name : _____

```

class DivisionMultipleclass extends BaseClass {
    int gcd( int a, int b)
    {
        return b == 0 ? a : gcd( b, a % b);
    }
    int lcm( int a, int b)
    {
        return (a * b) / gcd(a, b);
    }
}

```

```

class NumberConversionClass extends BaseClass {
    string toBinary( int num)
    {
        return Integer.toBinaryString( num);
    }
    string toHex( int num)
    {
        return Integer.toHexString( num);
    }
    string toOct( int num)
    {
        return Integer.toOctalString( num);
    }
}

```

11-23024

Topic Name

```
class CustomPoint class extends BaseClass {  
    void pn( string msg ) {  
        System.out.println( "*** " + msg + " ***" );  
    }  
}
```

public class Inheritance To Perform Numerical OP {

```
public static void main( String [ ] args ) {  
    sumclass sumobj = new SumClass();  
    sumobj . point( " sum:" ); sumobj . computeSum();  
}
```

DivisonMultiple class
DivisonMultiple class divMulobj = new DivisonMultiple class();

int a = 96, b = 60;

```
divMulobj . point( " GCD ", divMulobj . gcd( a, b ) );  
divMulobj . point( " LCM ", divMulobj . lcm( a, b ) );
```

NumberConversion class numConObj = new NumberConversion class();

Topic Name :

```
int num = 42;
```

```
numConvObj.print("Binary: ", numConvObj.toBinany(num));
```

```
numConvObj.print("Hex: ", numConvObj.toBinany(num));
```

```
numConvObj.print("Octal: ", numConvObj.toOctal(num));
```

```
new CustomPrint class().pnf("Execution Completed");
```

{

{