Abdur Rawf Akand
ID: IT23050

Question 1:

A Java program that reads a series of number from a file (input.txt), that determines the higest number in the series, calculat the sum of natural number up to that higest number, and write the result to another file 'output.txt':

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

public class Qus1_SeriesSum {

    public static void main (String[] args) {

    try {
            File inputFile = new File("input.txt");

            Scanner scanner = new Scanner (inputFile);

    String inputLine = scanner.nextLine();
      scanner.close();

    String [] numberString = inputLine.split(",");

    int maxNumber = 0;
```

Abdur Rouf Akand
ID: IT 23050

~~// find the hig~~

```
for (String numStr: numberStrings) {
    int num = Integer.parseInt(numStr.trim());
    if (num > maxNumber) {
        maxNumber = num;
    }
}

int sum = maxNumber * (maxNumber + 1)/2;

PrintWriter writer = new PrintWriter("Output.txt");

Write.println(maxNumber + "," + sum);

writer.close();
}
Catch (FileNotFoundException e) {

    System.out.println("An error occurred.");

    e.printStackTrace();
}
}
}
```

Abdur Rawf Akand
ID: JT23050

Question 2 b

Differences Between static and final fields and methods in Java:

Feature	static	final
Definition	Belongs to the class rather than an instance	Once assigned, cannot be changed (for variables) or overridden (for methods)
Applicable To	fields, methods, inner classes	fields, methods, classes
Memory Allocation	stored in method Area and shared among all instances	Same as other normal variable / methods, but cannot be modified after initialization
Inheritance Effect	can be inherited but can not be overridden (can be hidden)	If applied to a class - prevents, inheritance, If applied to method: privent overriding, If applied to variable: makes it constant

Code Example Demostrating static and final :

```
class Test {
    static int static_field = 10;
    final int finalfield = 20;

  static void staticMethod () {
    System.out.println ("static method called");
    }

  final void final Method () {
    System.out.println ("Final Method called"); }
    }

public class main {
  public static void main (String [] args) {
      Test obj = new Test ();

  // Accessing static members using an object
System.out.println ("static field via Object: + obj.staticfi
  obj.static Method ();

// Correct way to access static members
System.out.println ("static field via Class: + Test.static fiel
Test.static Method ();
```

Abdur Rawf Akand
ID: IT 2 3050

```
// accessing final field
System.out.println("final field: " + obj.finalField);

// calling final method
obj.finalMethod();

}
}
```

Abdur Rauf Akand
ID: 1723050

Question 3:

A Java programe to find all fractorion numbers within a given range:

```
import java.BigInteger;
import java.util.Scanner;

public class Qus3_factorion {
    public static BigInteger fact (int f) {
        BigInteger sum = BigInteger.ONE;
    for (int i = 1; i<= f; i++) {
        sum= sum.multiply (BigInteger.valueof (i));
    }
    return sum;
}
public static void main (String[] args) {
    System.out.println ("Enter lower of the range: ");
    Scanner se = new Scanner (System.in);
int     n = se.nextInt ();
System.out.println ("Enter UPER bound of the range: ");
int nh = se.nextInt();
```

Abdur Rawf Akand
ID: IT23050

```
for (int p=n; p<=nh ; p++) {
    int temp = p;
    BigInteger totalsum = BigInteger.ZERO;

    while (temp>0)
        int fn = temp%10;
        totalsum = totalsum.add (fact (fn));
        temp = temp/10;

    //check if the sum of factorials equal the numbers
    if (totalsum.equals (BigInteger.valueOf(p))) {
        System.out.print (p + " ");  }
    }
}
}
```

Abdur Rauf Akand
ID: IT23050

Question 4 :

Differences among Class, Local and Instance Variable in Java :

Feature	Class Variable (Static Variables)	Instance Variable	Local Variable
Definition	Belongs to the class, Shared among all objects.	Defined within a class but outside methods; unique to each object.	Declared inside a method, constructor, or block
Memory Allocation	Stored in method Area (shared among instances)	Store in Heap Memory (separate for each object	Store in Stack Memory (exists only during method execution
Access	Accessed using ClassName.VariableName or Object.variableName	Accessed using Object.VariableName	Accessible only inside the method/block where declared
Lifetime	Exists as long as the class is loaded	Exists as long as the object exists	Exists only within the method/block scope
Default value	Default value are assigned.	default values are assigned	must be explicitly initialize before use

Abdur Rauf Akand
ID: IT23050

Example Programme Demonstrating Class, Local, and Instance Variables:

```
class Example {
    // class variable (static variable)
    static int classVar = 100;

    // instance variable
    int instanaVar;

    // constructor
    Example (int value) {
        this.instanceVar = value ; // Assing value using 'this'

    Void display() {
        // local variable
        int localVar = 10;
system.out.println("Class Variable: " + classVar);
system.out.println("Instance Variable: " + instanceVar);
system.out.println("Local Variable: " + localVar); } }

public class main {
    public static void main (String[] args) {
        Example obj1 = new Example(20);
        Example obj2 = new Example(50);
        obj1.display();
```

Abdun Rawf Alcard
ID: IT23052

Question 5 :

A Java program that defines a method to calculate the sum of elements in an integer array and demonstrates its usage in the main method :

```
class ArraySum {
    static int calculateSum(int[] arr) {
        int sum = 0;
        for (int num : arr) { sum += num; }
        return sum;
    }
public static void main(String[] args) {
    // Define an array of integers
    int[] numbers = {10, 20, 30, 40, 50};
    //Call the method and store the result
    int sum = calculateSum(numbers);
    // Display the result
    System.out.println("Sum of array elements: " + sum);
    }
```

Abdur Rawf Akand
ID: 1723050

Question 6:

Access Modifiers in Java:

Access modifiers in Java define the scope (visibility) and accessibility of classes, methods, and variables. Java provides four class access modifiers.

1. public : Accessible everywhere
2. private : Accessible only within the same class
3. protected : Accessible with the same package and subclass.
4. Default - No modifier : Accessible within the same package only.

Comparison of public, privet and protected Modifiers:

Modifier	Same class	Same Package	Different Package (subclass)	Diff. Package (Non-Subclass)
public	Yes	Yes	Yes	Yes
privet	Yes	No	No	No
protected	Yes	Yes	Yes	No
Default (No modifier)	Yes	Yes	No	No

Abdur Rauf Akand
ID: IT23050

Different types of variables in Java:

In Java, variables are classified into three main types:

1. Instance variables:

Declared inside a class but outside any method, constructor, or block. Belong to an instance of the class. Example:

```
class Example {
    int instanceVar = 10; // Instance variable
```

2. Class variables (static variables):

Declared with the static keyword. Belong to the class, not to any specific instance. Example:

```
class Example {
    static int classVar = 20; // class Variable
}
```

Local variables: Declared inside a method, constructor, or block. Accessible only within the block where they are declared. Example: class Example {

Abdur Rauf Akand
ID: IT23050

Question 7:

A Java programme that find the smallest positive root of a quardratic equation of the form: $ax^2 + bx + c = 0$ using the quardidic formula

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

```
import java.util.Scanner;

public class Qus7_QuardaticRoot {

    public static void main (String[] args) {

Scanner scanner = new Scanner(System.in);

// Taking input for coefficients a, b, and c

System.out.print(" Enter co-efficients a, b, and c: ");

int a = scanner.nextInt();
int b = scanner.nextInt();
int c = scanner.nextInt();

double discriminant = b*b - 4*a*c

if (discriminant < 0) {
    System.out.println("No real roots");
    }
```

Abdur Rawf Aband
ID: IT 23050

```
double root1 = (-b + Math.sqrt(discriminant))/(2*a);

double root2 = (-b - Math.sqrt(discriminant))/(2*a);

// Finding the smallest positive root
double smallestPositiveRoot = Double.MAX_VALUE;

if (root2 > 0) smallestPositiveRoot = root1;
if (root1 > 0 && root2 < smallestPositiveRoot ) smallestPositiveRoot = root2;

// printing the result
if (smallestPositiveRoot == Double.MAX_VALUE) {
    System.out.println("No positive roots"); }
  else {
    system.out.println("The smallest positive root is: " + smallestR
  }
}
Scanner.close();
}
}
```

Abdur Rauf Akand
ID : IT28050

Question 8 :

[10] A program that can determine the letter, whitespace and digit in a string and count them;

```
import java.util.Scanner;

public class Qus8_CharacterAnalyzer {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // input string
        System.out.println("Enter a string : ");
        String input = sc.nextLine();

        int letters = 0;
        int whitespaces = 0;
        int digit = 0;

        // Analyze each character in the string
        for (char ch: input.toCharArray()) {
            if (character.isLetter(ch)) {
                letters ++;
```

Abdur Rawf Akand
ID: IT22050

```
else if (character.isWhitespace(ch)) {
    whitespaces++;
}
else if (character.isDigit(ch)) {
    digit++;
}
// output the results
System.out.println("Letters: " + Letters);
System.out.println("Whitespaces: " + whitespaces);
System.out.println("Digit: " + digits);
Scanner.close
sc.close();
}
}
```

Abdur Rauf Akand
ID: CT23050

Passing an Array to a Function in Java :

In Java we pass an array as an argument to a method by specifying it's type (int[], double[], String[], etc). Arrays are passed by reference, meaning changes inside the method affect the original array. Example:

```
Public class ArrayPassEx {
    static void printArray (int[] numbers) {
        system.out.print ("Array elements : ");
    for (int num : numbers) {
        system.out.print(num + " ");
        }
    system.out.println ();  }
Public static void main (String[] args) {
    int myArray = {10, 20, 30, 40, 50};
        printArray (myArray);
    }
```

Abdur Rauf Akand
ID: CT23050

Passing an Array to a Function in Java :

In Java we pass an array as an argument to a method by specifying it's type (int[], double[], String[], etc). Arrays are passed by reference, meaning changes inside the method affect the original array. Example:

```
Public class ArrayPassEx {
    static void printArray (int[] numbers) {
        system.out.print ("Array elements : ");
    for (int num: numbers) {
        system.out.print(num+ " ");
        }
        system.out.println ();  }
Public Static void main (String[] args) {
    int myArray = {10, 20, 30, 40, 50};
        printArray (myArray);
```

Abdur Rawf Akand
DO: IT 23050

Question 9:

Method Overriding in Java (Inheritance Context):

Method overriding in Java occurs when a subclass provides a new implementation for a method that is already defined in its superclass. The overridden method in the subclass must have the same name, return type, and parameters as in the superclass.

There are many features of Method Overriding like:

1. Same method signature: The method in the subclass must have the same name, parameters, and return types as in the superclass.
2. ~~Retu~~ Runtime Polymorphism: Overriding enables dynamic method dispatch, meaning the method that gets executed is determined at runtime, based on the actual object type.
3. Access Modifiers: The overriding method cannot be a more restrictive access modifier than the method in the superclass.

Abdur Rawf Akand
ID: IT21050

Example: If a method in the superclass is protected, it cannot be overridden as private in the subclass.

9. @Override Annotation: Helps catch errors during compilation if the method is not correctly overriding a superclass method.

Example of Method Overriding:

```
Class Animal {
    void makeSound(){
        System.out.println("Animal makes a sound");
    }
}
Class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}

public class MethodOverriding {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.makeSound();
        Animal myDog = new Dog();
```

Abdur Rawf Akand
ID: LT22050

Using super keyword in Overriding:

The super keyword is used to call the superclass method inside a subclass. This is useful when the subclass wants to extend the functionality of the superclass method instead of completely replacing it.

Example of supper keyword usage:

```
class Animal {
  void makSound () {
      System.out.println("Animal makes a sound");
  }
}

class Dog extends Animal {
    @override
    void makeSound () {
  super.make.Sound ();
    System.out.println("Dog barks");
  }
}

public class Qus9_SuperExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound();
```

10

Abdur Rawf Akand
ID: IT23050

Question 10:

Differences Between static and Non-Static Members in Java:

Feature	Static	Non-static
Belongs to	Class	Object
Access	className.method()	Obj.method()
Memory	Method Area (one copy)	Heap (each object has its own)
can access non-static?	No	Yes
Example	static int x;	int y;

```
class Example {
    static int a = 10; // static
    int b = 20;  // non-static
}
```

Abdur Rawf Pkard
ID: IT23050

Q1 A program that able to check either a number or String is palindrome or not:

```
import java.util.Scanner;

public class Qus10_PalindromeChecker {

    public static void main (String[] args) {

    Scanner sc = new Scanner(System.in);

    String input = sc.next();

    if (isPalindrome(input)) {
        System.out.println( input + " is Palindrome:");
    }
    else {
    System.out.println (input + " is not a Palindrome:");
    }
    sc.close();
public static boolean isPalindrom (String str) {

    Return str.equals (new StringBuilder (str).reverse().to string
    }
}
```

Abdur Rauf Pkard
ID: IT25050

Question 11 :

Class Abstraction and Encapsulation :

Abstraction : Hiding implementation details and showing only functionality. Achived using abstract classes and interfaces.

Encapsolation : Warpping data and method into a single unit (class) and restricting direct access using privet variables with gatter & satter methods.

Exmple :

```
// Abstraction using Abstract class

abstract class vehicle {

    abstract void start();
}

class Car extends Vehicle {

    void start() {

        system.out.println("Car stows with a key");
    }
```

Abdur Rauf Akand
ID: DT23050

```
// Encapsulation
class person {
private String name ; // privet variable.
void setName ( String n)
  {
    name = n;
  }
String getName() {
return name;
  }
}
```

Abdun Rawf Akand
ID: It 23050

Abstract class vs Interface :

Feature	Abstract class	Interface
Methods	Can have both abstract & non-abstract methods	only abstract methods (Java 7), default, static (Java 8)
Variables	Can have instance Variable	Only public, static, final
Constructor	Yes	No
Inheritance	Supports single inheritance	Support multiple inheritance
Usage	when Common behavior is shared	for Complete abstraction

Example :

```
abstract class Animal {

    abstract void sound();

}

interface Flyable {

    void fly();

}
```