IBT-23035

(1)

```
import Java.io.*;
import Java.util.*;
public class NumberProcessor{
public static void main(String[] args){
try {
  Scanner scanner = new Scanner(new file("
input.txt");
String line = scanner.nextline();
String[] numbers = line.split(",");
scanner.close();
List<Integer> numList = new ArrayList<>();
int highest = Integer.MIN_VALUE;
for (string numStr: numbers){
int num = Integer.parseInt(numStr.trim
());
numList.add(sum);
for (string numStr: numbers){
  int num = Integer.parseInt(
    numStr.trim());
```

```
numList.add(num);
if (num > highest) {
                highest = num; } }
int sumOfHighest = highest * (highest+1)/2;
PrintWriter writer = new PrintWriter(new file
("Output.txt");
writer.println("Highest Number: " +highest);
Writer.println("Sum of natural numbers up to
highest : " + sumOfHighest);
writer.close();
System.out.println("Processing Complete.
check output.txt for results."); } catch(IOException e)
{
    System.err.println("Error: "+e.get
Message());
    }
    catch (NumberFormatException e) {
    System.err.println("Invalid number
format in input file.");}}
}
```

Student ID: IT23035

Ans. to the Q. No. 02

Differences between static and final fields and methods in java.

Feature	Static	final Fields
Per Defination	Belongs to the class, not instances (objects).	Constant: value can not be changed after initialization
Usage	Variables, methods, blocks, nested classes	variables, methods, classes.
Memory Allocation	Shared by all instances of the class Stored in the class's memory (common to all objects)	Each instance can have it's own copy. Normal instance variables or methods (unless also static)
Access	Can be accessed directly with the class name	Accessed through objects or class name (if static)
Initialization	Variables can be initialized multiple times	Variables can only be initialized once

ID: IT23035

```
public class MyClass {
    public static int classCounter = 0; // Static var
    public final int instanceValue; // final variable

    public MyClass(int value) {
        this.instanceValue = value;
        classCounter++;
    }
    public static void incrementCounter() { // static method
        classCounter++;
    }
    public final void displayValue() { // final method
        System.out.println("Value: " + instanceValue);
    }
}
```

⇒ What while accessing static fields/methods through an object does not cause a compilation error, it is discouraged because it suggests instance-level access when, in reality they belong to the class.

IT- IT-23035

(3)

```
import java.util.Scanner
public class factorion {
    public static void main(String[] args){
Scanner sc = new Scanner(System.in);
int range1, range2;
System.out.println("Enter the lower bound
of the range: ");
range1 = sc.nextInt();
System.out.print("Enter the lower bound
of the range: ");
range2 = sc.nextInt();
System.out.println("Factorion numbers in
the range: ");
for (int num = range1; num <= range2;
    num++){
int currentNum = num;
int sum = 0;
int fact = 1;
```

```
while (currentNum > 0) {
int digit = currentNum % 10;;
 currentNum /= 10;
for (int i = 2; i <= digit ; i++){
    fact = fact * i;
}
if (num == num) {
~~System.out.print(num);~~
~~if num! = 40585)~~
System.out.print(num + " , ");
}
}
}
}
```

Output : 1, 2, 195, 40585

ID: IT23035

Ans. to the Q. No. 04

Difference Among class, Local and instance Variables in JAVA :-

Variable Type	Scope	Lifetime	Declaration	Example
class (static) variable	Available throughout the class	Exists from the moment the class is loaded until the program terminates	Declared with static inside a class but outside any method	static int rate of Class;
Instance (Non-static) Variables	Available as long as the object exists	Available as long as the object exists	Declared with inside a class but outside any method without static	String name;
Local Variables	Limited to the method or block where it is defined	Exists only within the block where it's declared	Declared inside a method, construction or block	int counter; (inside a for loop)

JD - IT23035

In Java, class, local, instance variable's significance is:

① class (static) variables:
Belong to the class, not individual objects (shared among all instances). Stored in class memory and initialized only once.

② Local variables:
Declared inside a method, block or constructor. Exist only during method execution (stored in the stack). Must be initialized before use.

③ Instance Variable:
Belong to an object (each object has its own copy). Stored in heap memory and persist as long as the objects exists.

ID : JT23035

Ans. to the Q. No. 06

Access modifiers in JAVA are keywords that control the visibility or accessibility of classes, interfaces, constructors, methods and variables. They determine which parts of your code can access and use a particular element. Java has four access modifiers, they are public, protected, default, private.

Comparison the accessibility of public, private and protected modifier : ~~Describe different types of variable in JAVA with example~~

(1) Public : public modifier is the most accessible. It can be accessed anywhere (same class, same package, different package, subclass). It is used when the member should be completely open.

ID - IT23035

② protected: It can be accessed within the same ~~pak~~ package and by subclasses in other packages. It is used when data should be accessible only to derived (child) classes.

③ private: Private modifiers is least accessible. It can be accessed only within the same class. It is used for encapsulation (hiding penmitive data)

Different Types of variables in Java:-

Java has ~~main~~ three main types of variables:

① Local variables:

* Declared inside a method, constructor or block
② Must be initialized before use

(3) Not accessible outside the method / block.

Example:

```
class Example {
    void show() {
        int localVar = 10; // Local Variable
        System.out.println("Local Variable: " +
        localVar);
    }
}
```

(11) Instance Variable:

(1) Declared inside a class but outside any method

(2) Each object has it's own copy

(3) Stored in heap memory and automatically initialized if not possible (0, Null, false):

Example:

```
class Example {
    int instanceVar = 20; // Instance Variable
    void display() {
        System.out.println("Instance Variable: " +
            instanceVar);
    }
}
```

ID : IT23035

(iii) class variable (static variables)

(1) declared using static keyword
(2) shared across all objects of the class
(3) Memory allocated only once in the class area.

Example :

```
class Example {
    static int classVar = 30;
    void display() {
    System.out.println("class Variable: " +
classVar);
    }
}
```

(7)

```
import java.util.Scanner;
public class QuadRoot {
 public static void main (String[ ] args){
Scanner sc = new Scanner (System.in);
System.out.print (" Enter coefficients a,b,
and c : ");
int a = sc.nextInt(),b = sc.nextInt(), c = sc.NextInt();
double d = b*b - 1*a*c;
if (d<0) { System.out.println (" No real roots.");
else { double r1 = (-b + Math.sqrt(d))/(2*a);
       double r2 = (-b - Math.sqrt(d))/(2*a);
   double minRoot = Double.MAX_VALUE;
   if (r1 >0) minRoot = Math.min(minRoot, r1);
   if (r2 >0) minRoot = Math.min(minRoot, r2);
System.out.println(minRoot == Double.MAX_
Value ?" No positive roots" : the smallest positive
root is : " + minRoots); } }
```

(8)

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

IOT-23035

In Java, an array can be passed as an argument to a method just like any other variable.

```
public class ArrayParam {
    static int sumArray(int[] arr) {
        int sum = 0;
        for (int num : arr) {
            sum += num;
        }
        return sum;
    }
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        int total = sumArray(numbers);
        System.out.println("Sum of array elements: "
            + total);
    }
}
```

IT-23035

(9) Method overriding in Java is a key concept in object-oriented programming that allows a subclass (child class) to provide a specific implementation for a method that is already defined in its superclass (parent class). It's a way to customize or specialize the behavior inherited from the superclass.

key rules of overriding:

The method in the child class must have the same name, return type and parameters as in the parent class.

The child class method can not have a more restrictive access modifier than the parent method.

IT-23035

The annotation is used to indicate that a method is overridden (optional but recommended).

Ans to the Q. No. 10

Difference between static and non-static members:

In object-oriented programming (OOP), static and non-static members behave differently in terms of accessibility, memory allocation and usage: Here's a breakdown of their key differences:-

Feature	Static Members	Non-static Mem
Defination	Belong to the class rather than an instance	Belong to an instance of the class

Feature	Static	Non-Static
Memory Allocation	Stored in a single memory location and shared among all instances	A separate copy is created for each instance
Access	Accessed using the class name	Accessed using an instance of the class
Usage	Used for shared properties utility methods or constants	used for instance specific data and behavior

IT23035

Ans. to the Q. No. 11

Both abstraction and encapsulation are fundamental concepts of object-oriented programming (OOP) but they serve different purposes.

Abstraction:
Abstraction is the process of hiding the implementation details and showing only the essential features of an object. It helps to reduce complexity and increase reusability.

Encapsulation: Encapsulation is the technique of wrapping data and methods together into a single unit and restricting direct access to some details. It ensures data security and prevents accidental modifications.

IT-23035

Both abstract classes and interfaces help achieve abstraction in Java but they have key differences in their implementation and usage.

Feature	Abstract class	Interface
Defination	A class that can have abstract and concrete methods	A blueprint that defines only abstract methods but can have default / static method
Access modifiers	Methods can have any access modifier (private protected, public)	All methods are implicitly public abstract
use case	used when objects share common behavior	used when objects share common capabilities but may have different implementations.

IT-23035

(12)

```
import java.util.Scanner;

class BaseClass {
    void printResult(String msg, object
        result) {
    System.out.println(msg + result);
    }
}

class SumClass extends BaseClass {
    double sumSeries() {
        double sum = 0.0;
    for (double i = 1.0; i >= 0.1; i -= 0.1) {
        sum += i;
    }
    return sum;
}
}
```

ICP- IT- 23035

```
class DivisorMultiple class extends BaseClass.
    int gcd (int a, int b){
        while (b != 0){
            int temp = b;
            b = a % b
            a = temp; }
    return a; }
class NumberConversion class extends
BaseClass {
        String toBinary (int num){
    return Integer to Binary string (num); }
String toHex (int num){
    return Integer.toHex String (num) to
UpperCase();
class customPrintClass {
        void pr (String msg)
        System.out.println (">>" + msg);
    }
}
```

IT 23035

```
public class MainClass {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

public class MainClass {
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
SumClass sumObj = new SumClass();
DivisorMultiple.MultipleClass dmObj = new DivisorMultipleClass();
NumberConversionClass noObj = new NumberConversionClass();
CustomPrintClass printObj = new CustomPrint
                                        ();
```

IT 23035

(13)

```
import java.util.Date;
Class Geometric object {
    String color;
    boolean filled;
Date dateCreated = new Date();
Geometric object (String color, boolean filled){
    this.color = color;
    this.filled = filled; }
public String to string(){
return "color:" + color + ", filled:" + filled +
", Created :" + date(reated) }

}

Class Circle extends Geomatric object
    double radius;
```

IT 23035

```
Circle (double radius, String color, boolean
filled) {  super(color, filled);
           this.radius = radius; }
Public String toString() {
return "Circle → " + super.toString() + ",
Radius": " + radius + ", Area: " + (Math.PI *
radius * radius); } }

public class Main {
    public static void main(String[] args) {
System.out.println(new Circle(5, "Red",
                true));
    System.out.println(new Rectangle(4,7,
    " Blue ", false));
}
}
```

(90) (15)

Abstract class:

(1) It's have both abstract methods and concrete methods.

(2) Can have instance variables with any access modifier

(3) Can have constructors.

(4) A class can extend only one abstract class.

Interface:

(i) It's have only abstract methods.

(ii) Can only have public static final constants

(iii) A class can implement multiple interface.

(iv) Can not have constructors.

(v) Methods are public by default.

Yes a class in java can implement multiple interfaces. This is a key advantage of interfaces over abstract classes as java does not support multiple inheritance with classes.

Example

```
interface A { void display (); }

interface B { void display(); }

class C implements A, B {
    public void display () {
System.out.println ("resolved display
method in class C"); }
}
```

IT-23035

16

Polymorphism in Java:

Polymorphism is a ability of a single interface to represent forms. In Java, polymorphism is achieved through method overriding (runtime polymorphism) and method overloading (compile time polymorphism)

Dynamic method dispatch:

Dynamic method dispatch refers to the process where method calls are resolved at runtime instead of compile time. When a subclass overrides a method from its superclass and its referenced through a parent type, Java determines the method to execute based on the actual object type no

Example :

```
class Animal {
    void makesound() {
        System.out.println("Animals make a sound");
    }
}
class Dog extends Animal {
    void makesound() {
        System.out.println("Dog barks");
    }
}
public class main {
    public static void main(String[] args) {
        Animal myAnimal;
        myAnimal = new Dog();
        myAnimal.makesound();
        myAnimal = new cat();
        myAnimal.makesound();
    }
}
```

IT-23035

(17)

(A) ArrayList is backed by a dynamic array. This means elements are stored contiguously in memory.

(B) LinkedList is a doubly linked list. Each element stores its data and pointers to the previous and next node in the sequence.

Difference

Operation	ArrayList	LinkedList
Access	O(1) - Direct access via index	O(n) - sequential
Insertion (add at end)	O(1) - Amortized	O(1) Direct tail insertion
Insertion (add at index)	O(n) - shifting element	O(n) - Traversing to the index
Deletion (remove at index)	O(n) "	O(n) "
Deletion remove first/ last	O(n) "	O(1) Direct head/tail removal

IT-23035

(18)

Custom random number generator: According to the question: public class

```
class CustomRandomGenerator {
    private static final int[] predefineArray
            = {3, 7, 11, 13, 17};
    private static final int maxValue = 1000;
public static int myRand(){ return
    myRand(1)[0]; }
public static int[] myRand(int n) {
int[] result = new int[n];
long time = System.currentTimeMillis();
    for(int i=0; i<n; i++) {
result[i] = (int)((time * predefineArray[i%
predefineArray.length]) % maxValue);
    public static void main(String[] args){
```

~~ICS~~ IT-23035

```
for (int num : myRand(5)) System.out.println
                                    (num);
System.out.println(myRand());
}
```