IT-22053(Re-Ad)

Q1:

```
import java.io.File;
import java.util. Scanner;
import java.io.printWriter;
import java.io.fileNotFoundException;
Public class Series_sum{
public static void main(String[] args){
try {
File inputfile = new File ("input.txt");
 Scanner scanner = new Scanner (inputFile);
 File outputfile = new file ("output.txt");
 PrintWriter writer = new PrintWriter (outputFile);
 While(scanner.hasNest()) {
    int n = scanner.nextInt();
    int sum=0;
   for(int i=1; i<=n; i++){
      sum = sum+i;
    }
    writer.println(sum);
   }
   scanner.close();
   writer.close();
  } catch(FileNotFound Exception e){
   e.printstackTrace();
   }
  }
}
```

Q1:

```
import java.io.File;
import java.util.Scanner;
import java.io.printWriter;
import java.io.fileNotFoundException;
Public class Series_sum{
public static void main(String[] args){
try{
 File inputFile = new File("input.txt");
  Scanner scanner = new Scanner(inputFile);
  File outputFile = new file("output.txt");
  PrintWriter writer = new PrintWriter(outputFile);
  While(scanner.hasNest()){
     int n = scanner.nextInt();
     int sum=0;
    for(int i=1; i<=n; i++){
        sum = sum+i;
      }
      writer.println(sum);
     }
    Scanner.close();
    writer.close();
   } catch(FileNotFoundException e){
     e.printstackTrace();
     }
   }
```

Q2: In Java, static and final are two distinct keywords used to define different characteristics of fields and methods

Here are the key differences:

Feature	static	final
1. Definition	Belongs to the class rather than any specific instance	marks a field, method, or class as unchangable or non-overridable
Scope	Shared across all instance of the class	Applies only to the specific instance (if field) or method (if method is final)
Modification	Can be changed (if not final)	Cannot be changed once assigned (for field)
Inheritance	Inherited by subclasses but not overridden	cannot be overridden if applied to methods
Usage	Accessed via class name	Prevents modification or method

Q3:

```
import java.util.Scanner;
public class Factorion{
public static void main(String[] args){
Scanner input = new Scanner(System.in);
int i,num,temp;
System.out.println("Enter your Range:");
 i = input.nextInt();
 num = input.nextInt();
 int count =0;
for(int k=i; k<=num; k++){
  temp =k;
  int sum =0;
  while(temp!=0){
    int rem = temp%10;
     int fact = 1;
    for (int j=1;j<=rem; j++){
         fact = fact*j;
      }
      sum = fact +sum;
      temp = temp/10;
      }
     if(sum==k){
      count++;
    system.out.println(" Number is:"+ ~~sum~~ sum);
    }
   System.out.println("Total number is:"+ count);
   input.close();
   }
   }
```

Q4: Difference Among class, local, and Instance variable:

variable type	Scope	Storage	Access
class variable	shared across all instance of a class	stored at the class level	Accessed using classname.variable
Instance variable	unique to each instance of a class	stored within individual object instances	Acessed using self.variable
Local variable	Exists only within a function method	stored temporarily during function execution	Accessible only within the function

significance of 'this' keyword:

In many oop language, 'this' refers to the current instance of a class. It is used to:

i) Distinguishinstance variables from method parameters.

ii) Access instance methods and properties within a class.

iii) pass the current object to another method or constructor.

QN-05:

```
  import java.util.Scanner;
 Public class Arraysum{
 public static int array_sum(int[] ar, num){
    int sum=0;
  for(int i=0; i<num; i++){
     sum=sum+ar[i];
   }
  return sum;
 }
 public static void main(String[] args){
  Scanner in=new Scanner(System.in);
     int n = in.nextInt();
  int[]A = new int[n];
  for(int i=0; i<n; i++){
       A[i] =in.nextInt();
    }
  int result = array_sum(A,n);
  System.out.println("The sum of this array is:"+result);
  in.close();
 }
}
```

Q6:

Access Modifier: An Access Modifier in Java is a keyword that determines the visibility or scope of a class, method or variable. It controls how different parts of a program can access a particular member of a class.

There are four types of access modifier in Java:

① private, ⑪ protected ⑪ public ⑫ Default.

① private: The member is only accessible within the same same class

⑪ protected: The member is accessible within the same package and in subclasses of different packages.

⑭ public: The member is accessiable from anywhere in the program.

Instance variable: It defined inside a class but outside of any method:

```
class student {
    string name;
    int age;
}
```

Static variable (class variable): Declared with the static keyword. shared among all objects of the class.

QN-07

```
    import java.util.*;
 public class Rootfind {
public static void main(String[] args) {
  Scanner input = new Scanner(System.in);
    double a, b, c;
     a = input.nextDouble();
     b = input.nextDouble();
     c = input.nextDouble();
   double d = b*b - 4*a*c;
    if (d>0) {
   double root1 = (-b+math.sqrt(d))/2*a;
   double root2 = (-b-math.sqrt(d))/2*a;
    double result = math.min(root1, root2);
    System.out.println("The smallest positive root is:"+result)
   }
  else {
    system.out.println("No real root exist");
   }
  input.close();
 }
}
```

SN-08:

```
    import java.util.Scanner;
  Public class Checker {
  public static void main( String[] args){
    Scanner input = new Scanner( System.in);
     String s = input.nextLine();
   int letter =0, digit =0, space =0;
   for(int i=0; i<s.length(); i++){
     if(s.charAt(i)==' '){
       space++;
     }
   else if (s.charAt(i) >= '0' && s.charAt(i)<='9')
        digit++;
      }
    else {
      letter++;
     }
    }
 System.out.println("Letter:"+letter);
 System.out.println("Whitespace:"+ space);
 System.out.println("Digits:"+ digit);
  input.close();
 }
}
```

```
class student {
static string univernty ='MBSTU";
}
```

Local variable: Declared inside a method, constructor or Block. Only accessible within that method or block.

```
class Test {
 void display () {
  int x =10;
  system.out.println(x);
 }
}
```

Final variable: Declared using the 'final' keyword. Once assigned, its value cannot be changed.

```
class test {
final int MAX_VALUE =100;
}
```

Q9:

Method Overriding: method Overriding in java occurs when a subclass provides a specific implementation of 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 the method in the class superclass.

Using the 'super' Keyword: The 'super' keyword allows access to the superclass's methods and constructors. when a subclass overrides a method, it can still call the superclass's method using super.methodName(). Example:

```
class parent {
void display () {
System.out.println("parent class method");
}
}
class child extends parent {
@override
void display () {
super.display(); // calls the superclass method
System.out.println("child class method");
}
}
```

```
public class Test {
public static void main(String[] args){
 Child obj=new chid();
  obj.display();
 }
}
```

Sample output: parent class method
child class method

Q 10: Difference between static and non-static members:

Static members:

Static members belongs to the class itself rather than an instance of the class. They are shared among all objects of the class. Accessed using the class name instead of an instance.
Declared using the static keyword. Memory is allocated once when the class is loaded. Cannot access non-static members directly. Can be used without creating an object of the class.

Non-static members:

Non-static members belongs to each instance of a class
Every object gets its own copy of the non-static member

Can be accessed only through an object.

No static keyword. Memory is allocated separately for each instance. Can access both static and non-static members.

Q11:

i) Abstraction: Abstraction is a concept in OOP that is used to hide complex implementation details and only show the essential features of an object. It allows focusing on what an object does rather than how it does it.

ii) Encapsulation: Encapsulation is the process of wrapping data (variables) and methods into a single unit (class) and restricting direct access to some of the objects components, thus ensuring data protection and security.

Difference between Abstract class and Interface:

Abstract class	Interface
A class that contains abstract and non-abstract ~~class~~ methods.	A ~~but~~ blueprint that contains only abstract methods and default/static methods.
Can have both abstract and concrete methods.	cannot have concrete methods
Can have constructors	cannot have constructors

```
system.outprintln("Tiger:"+tiger.sound());
System.out.println("Chicken:"+ chicken.sound());
System.out.println("chicken"+ Chicken.howToEat());
system.out.println("Orange:"+ orange.howTOEat());
system out.println("Apple:"+ apple.howToEat());
}
}
```

QN-14

```
    import java.util.scanner;
    import java.math.BigInteger;
  public class FactorialBigInt {
   static BigInteger factorial (BigInteger n) {
       BigInteger fact = BigInteger.ONE;
  for (BigInteger i = BigInteger.ONE; i.compareTo(n) <=0;
                           i = i.add (BigInteger.ONE)) {
            fact = fact.multiply(i);
         }
     return fact;
   }
  public static void main(string[] args) {
    Scanner input = new Scanner(System.in);
  system.outprintln ("Enterthe number:");
   BigInteger num = input.nextBigInteger();
```

Abstract class	Interface
Cannot support multiple interitance directly	Suppopt multiple inheritanee.
Can have public, poivate, or Prooteeted methods.	methods are always public (By default)
Used foro code reuse and partial abstraetion	Used for full abstraetion and defining behaviour that multiple classes must implement

Q15:
Yes, a class can implement multiple interfaces in java. This is a key advantage of interfaces over abstract classes:

Example:

```
interfaceA {
 void method();
 }
 interface B {
   void method B();
 }
class myclass implements A, B {
public void methodA() {
system.out.println("Method A Implementation);
} public void methodB() {
System.outprintln("Method B Implementation");
```

Q16:

polymorphism: polymorphism in Java is the ability of an object to take many forms. It allows a single interface to be used for different data types, making code more flexible and maintainable. polymorphism primarily occurs in two ways:

i) Compile-time polymorphism (method Overloading) — multiple methods in the same class have the same name but different parameters

ii) Runtime polymorphism (method Overriding) — A subclass provides a specific implementation of a method already defined in its superclass.

Q17:

Difference between ArrayList and LinkedList focusing on time complexities are below:

operation	Arraylist	Linkedlist
Access (get(index))	O(1)	O(n) (traversal required)
Insertion at end	O(1) amortized	O(1)
Insertion at beginning	O(n) (shifting elements)	O(1)
Insertion in middle	O(n) shifting required	O(n) (traversal required)

Deletion at end	O(1)	O(1)
Deletion at beginning	O(n) shifting elements)	O(1)
Deletion at middle	O(n) (shifting required)	O(n) (traversal required)
Iteration(next())	O(1)	O(1)

Q 19:

Multithreading in java is achieved using the Thread class and the Runable interface. It allows multiple threads to run concurrently, improving performance in application that involve tasks like I/O operation, parallel processing, and background computations.

Difference between Thread class and Runnable Interface:

Thread class	Runnable Interface
Inheritance → Extends Thread (cannot extend another class)	Implements Runnable, allowin multiple inheritances
Implementation → Need to override run() method	Needs to implement run() method
Instantiation → Thread t = New My Thread (): t.start()	Thread t = new Thread (new Runnable()); t.start()
Flexibility → Not recommended for reusability	More flexible and recom-mended for thread creatio

IT-22053 (Re-Ad)

Q 20:

Exception handling is a powerful mechanism used to handle runtime errors, allowing the program to maintain normal flow of execution even when an exception occurs. Java provides a robust exception-handling framework using try, catch, finally, throw and throws.

How exception handling works:

- ~~Try~~ Block ~~catch~~: code that might throw an exception is placed inside the try block.
- Catch Block: After the try block, the catch block is used to handle specific exceptions that may be thrown. It specifies the type of exception it handles.
- Finally Block: This Block, if present is always executed after the try block and any associated catch blocks, regardless of whether an exception occured or not. It is typically used for cleanup activities, like closing files or releasing resources

Checked VS Unchecked Exceptions:

Checked exception: These exception are checked at compile-time, and the compiler forces you to handle using a try-catch block or declaring them with the throws keyword. Ex: IOException.

Uncheeked Exeeptions: These exceptions are not cheeked at compile-time and typically extend RuntimeException. They represent programming bugs or logical errors, such as Nullpoint Exception.

The role of throw and throws keywords:

- throw keyword: This is used to explicitly throw an exception in a method ~~or~~ block of code
- throws keyword: This is a method signature to declare that the method might throw one or more exception. This informs the caller of the method about the potential exceptions

Q 21:

In Java 8 and beyond, the introduction of default methods in interfaces has created a notable shift in how inter- faces and abstract classes are used.

Defalt Default methods: Default methods in interfaces allow you to provide a method implementation directly within the interface. This was previously not possible, as interface could only declare methods without implementation.

QN-07

```
    import java.util.*;
  public class Rootfind {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
     double a, b, c;
      a = input.nextDouble();
      b = input.nextDouble();
      c = input.nextDouble();
   double d = b*b - 4*a*c;
     if (d>0) {
    double root1 = (-b+math.sqrt(d))/2*a;
    double root2 = (-b-math.sqrt(d))/2*a;
     double result = math.min(root1, root2);
     System.out.println("The smallest positive root is:"+resul
   }
  else {
    system.out.println("No real root exist");
    }
  input.close();
 }
}
```

QN-08:

```
   import java.util.Scanner;
 Public class Checker {
 public static void main( String[] args) {
   Scanner input = new Scanner( System.in) ;
     String s = input.nextLine();
  int letter =0, digit =0, space =0;
  for(int i=0; i< s.length(); i++) {
    if( s.charAt(i) == ' ') {
      }space ++;
  else if ( s.charAt(i) ~~==~~ >= '0' && s.charAt(i)<='9') {
        digit++;
      }
   else {
      letter++;
    }
  }
System.out.println("Letter:"+ letter);
System.out.println("Whitespace:"+ space);
System.out.println("Digits:"+ digit);
input.close();
}
}
```

```
system.out.println ("Factorial of"+num+"is:"+ factorial (num));
 input.close();
}
}
```

QN-10:

```
import java.util.scanner;
    public class parlindrome {
  public static void main (string[] args) {
      Scanner input = new Scanner(system.in);
        String s = input.nextLine();
        String r = " ";
      for (int i = s.length()-1; i>=0; i--){
                r = r + s.charAt(i);
           }
        boolean cheek = true;
      for( int i = 0; i<s.length; i++){
           if (s.charAt(i)! = r.charAt(i)){
                cheek = false;
                 break;
             }
    if (cheek) system.out.println("Palindrome");
    else  System.out.println("Not palindrome");
        input.close();
    }
system.outprintln("Tiger:"+tiger.sound());
System.out.println("Chicken:"+ chicken.sound());
System.out.println("chicken"+ Chicken.howToEat());
system.out.println("Orange:"+ orange.howToEat());
system out.println("Apple:"+ apple.howToEat());
}
}
```

QN-14

```
      import java.util.scanner;
      import java.math.BigInteger;
 public class FactorialBigInt {
  static BigInteger factorial (BigInteger n) {
      BigInteger fact = BigInteger.ONE;
 for (BigInteger i = BigInteger.ONE; i.compareTo(n) <=0;
                         i = i.add (BigInteger.ONE)) {
          fact = fact.multiply(i);
       }
    return fact;
  }
public static void main(string[] args) {
  Scanner input = new Scanner(System.in);
 system.outprintln ("Enter the number:");
  BigInteger num = input.nextBigInteger();
```

```
return" chicken clucks!";
}
@override
 public string howToEat() {
return "could be fried or grilled";
 }
}
abstract class Fruit {
  } // Not implementing Edible directly

class Orange extends Fruit implements Edible{
  @override
 public string howtoEat(){
  return" peel and eat fresh";
  }
  }
Class Apple extends fruit implements Edible{
 @override
 public string howToEat(){
 return" Wash and eat raw";
Public class TestEdible {
Public static void main(String[] args) {
Animal tiger = new Tiger();
Animal chicken = new Chicken();
Fruit orange = new Orange();
Fruit apple     = new Apple();
```

2. Extending Thread class:

```
Class MyThread extends Thread{
 public void run(){
System.out.println("Thread is running using thread class...");
 }
public static void main(String[] args){
 MyThread obj = new MyThread();
      obj.start();
   }
 }
```

QN-27

```
  interface Edible{
  string howToEat();
  }
 abstract class Animal{
abstract string sound();
 }
 Class Tiger extends Animal{
 @override
  string sound(){
   return "Tiger roars!";
  }
 }
Class chicken extends Animal implements Edible{
@override
String sound(){
```

```
System.out.print("Enter radius");
 double radius = input.nextDouble();
 try {
  Circle.setRadius (radius);
 system.outprintln("Arear:"+ Circle.Area());
 } catch (IllegalArgumentException e) {
 System.out.println ("Error:"+ e.getMessage());
 }
 input.close();
}
```

QN-26

Thread: A thread is the smallest unit of execution within a process. I

a lightweight subprocess that allows concurrent execution of tasks

a program.

There are two ways to create a thread in java.

1. By implementing the Runnable interface

2. By extending the Thread class

1. Using Runnable Interface:

```
class MyRunnable implements Runnable {
 Public void run() {
System.out.println(" Thread is running using Runnable interface.
}
public static void main(String[] args) {
Thread obj = new Thread(new MyRunnable());
PrintWriter outputfile = new PrintWriter (new File ("output.txt"));
  outputfile. println ("sum of numbers:"+ sum);
  outputFile. Println("Highest value;"+ higest);
   outputfile.close();
   inputfile1. close();
  } catch (FileNotfoundException e) {
   System.out.println("File not found:"+ e);
  }
  }
```

QN 25:

```
import java.util.scanner;
       public class Circle{
       private double radius;
  public void setRadius(double radius) throws IllegalArgumentException{
     if (radius<0) {
     throw new IllegalArgumentException ("Radius can't be negative");
      }
    this.radius = radius;
   }
  public double Arear(){
   return Math.PI *radius *radius;
   }
  public static ~~void~~ void main (string[] args) {
 Scanner input = new Scanner(system.in);
   Circle cirle = new Circle();
else.
   double quatient = Math.ceil( (double) A1[i] /A2[indexA2]);
   int remainder = A1[i] % A2 [indexA2];
system.outprintln ("For element" + A1[i] + "and divisor"+ A2[indexA2]+"
system.out.println("Quotient (ceiling):"+ quotient);
system.out.println( "Remainder:"+ remainder);
}
}
}
```

QN 29 :

```
import java.io. *;
      import java. util. *;
  public class ReadandWrite {
public static void main( string[] args){
try{
 scanner inputfile = new Scanner(new File("input.txt"));
   int sum = 0;
   int highest = ~~Intg~~ Integer.MIN_VALUE;
while(inputfile.hasNestInt()){
   int num = inputFile.nextInt();
   sum= sum + num;
 if (num~~ber~~ > higest) {
    highest = num;
  }
}
```

QN 30:

```
    import java.util.scanner;
public class ArrayDivision{
public static void main(string[] args){
Scanner input = new Scanner(system.in);
system.out.println("size first array (n>20):");
    int n = input.nextInt();
if (n<=20){
system.out.println("greater than 20");
 return;
}
int[] A1 = new int[n];
system.out.println("Enter elements:");
for(int i=0; i<n; i++){
    A1[i] = input.nextInt();
}
int m =(int) math.Ceil(n/10);
int[] A2 = new int[m];
for(int i=0; i<m; i++){
     A2[i] = input.nextInt();
}
for(int i=0; i<n; i++){
    int indexA2 = i%m;
   if (A2[indexA2]==0){
system.out.println("Division by zero is not allowed");
}
```

```
public static void main(string[] args){
  Scanner sc =new Scanner(system.in);
    int num = sc.nextInt();
  for(int i=0; i< num; i++){
      new CounterClass();
  System.outprintln("Instance Count:"+ Counterclass.getInstanceCount
  }
 }
}
```

QN 31 :

```
import java. text . SimpleDateFormat;
import java.~~utl~~util. Date;
public class CurrentDateTime {
public static void main (string[] args){
SimpleDateFormat  dateformat = new SimpleDateFormat("yyyy-MM
                                        -dd  HH:mm:ss");

 Date now = new Date();

system.out.println("Current Dateandtime :"+ dateformat.~~format~~
                                                  format(now));
}
}
```

```
else if (type.equals("Largest")&& num>extreme){
        extreme =num;
      }
    }
   return extreme;
  }
  public static void main (String[] args){
  int x =findExtreme ("Smallest", 5,2,9,1);
  int Y = findExtreme ("Largest", 8,3,10,4);
  System.out.println("Smallest: "+x);
  System.out.println("Largest:"+Y);
 }
}
```

QN 32:

```
import java.util.Scanner;
   public class CounterClass{
private static int instanceCount =0;
public CounterClass(){
 ~~ints~~ instanceCount++;
   if(instance Count >50)
        instance Count= 0;
      }
    }
public static int getInstanceCount(){
 return instanceCount;
}
```

```
@ override
public void methodE () {
System.out.println("methodE implemented");
}
}
```

QN 34:

output and explanation:

1. true → ".equals()" checks content equality; S1 and S2 have th
   same content.
2. false → "==" checks reference equality; S1 and S2 are
   different objects:
3. true → S1 and S3 refere to the same string literal in the
   String pool.

QN 33:

```
   public class Extremefinder {
public static int findExtreme (String type, int... numbers){
 if (numbers.length==0){
 throw ne IllegalArgumentException ("At least one number must be
                                              provided");
 }
 int extreme = numbers[0];
 for (int num : numbers){
 if (type.equals("smallest")&& num<extreme) {
    extreme=num;
 }
```

QN 35&36:

```
interface Alpha{
  void method A();
   void metho B();
 }
 interface Beta{
  void methodC ();
  void methodD();
 }
abstract class AbstractBase implements Alpha{
  public abstract void methodE();
 }
Class Finalclass extends AbstractBase implements Beta{
 @Override
 public void methodA(){
System.outprintln("methodA implemented");
}
 @override
 public void methodB(){
 System.out.Println("methodB implemented");
 }
@override
public void methodC(){
 System.out.println("methodC implemented");
}
@Override
public void methodD (){
System.out.println("methdD implemented");
}
```

QN-37

The issue with the given java code is that class Z inherits conflictin default methods from both X and Y. Since both interfaces provides a default implementation of show(), the compiler does not know which one to use, leading to a compilation error.

solution1: Override show in Z.

Explicitly override the show() method in class Z and provides it own implementation:

```
public class Z implements X, Y {
  @override
  public void show() {
  system.out.println("Z's show method");
  }
  public static void main(String[] args) {
      Z obj = new Z();
      obj. show();
    }
  }
```

solution2: Use 'super' to Specify an Interface's method:

```
public class Z implements X, Y {
  @override
  public void show() {
  X.super.show(); // or Y.super.show();
  }
  public static void main(String[] args) {
   Z obj = new Z();
   obj.show();
  }
  }
```

Q40:

```
import java.util.Scanner;
 public class ArithmeticOperations {
public static void main (String[] args) {
try {
Scanner input = new Scanner();
System.out.println("Enter two number:");
 int a, b;
  a = input.nextInt();
  b = input.nextInt();
 System.out.println("sum:"+ (a+b));
 System.out.println("Difference:" + abs(a-b));
 System.out.println("product:" + (a*b));
System.out.println("Quotient:"+ (a/b));
} catch(Exception e) {
System.out.println("Exception:"+ e);
System.out.println(" You must Enter integer.Try again.");
}
input.close();
}
}
```

The-end