

TOPIC NAME: _____ TIME: _____ DATE: / /

```
import java.io.*; import java.util.*;  
public class even {  
    public static void main(String[] args) {  
        String input = "input.txt";  
        String output = "output.txt";  
        Scanner read = new Scanner(new File(input));  
        PrintWriter write = new PrintWriter(new File(output));  
        read.useDelimiter(",");  
        List<Integer> numbers = new ArrayList<>();  
        while (read.hasNextLine()) {  
            numbers.add(read.nextInt());  
        }  
        if (numbers.isEmpty()) {  
            write.println("No numbers found");  
            read.close();  
            write.close();  
            return;  
        }  
        int maxnum = Collections.max(numbers);  
        write.println("Maximum Number : " + maxnum);  
        for (int x : numbers) {  
            int sum = (x * (x + 1)) / 2;  
            write.println(sum + ",");  
        }  
        write.close();  
        read.close();  
    }  
}
```

(2)

IT-22023

TOPIC NAME:

DAY: _____

TIME: _____

DATE: / /

Detailed Differences Between static and final in java.

static	final
1. A member (field or method) marked as static belongs to the class rather than any specific instance.	2. A member (field or method) marked as final belongs to instance rather than any specific class.
2. Applies to variables, methods, blocks, nested classes.	2. Variables, methods classes.
3. Shared by all instances of the class. A single copy exists in memory.	3. Once assigned, its value cannot be changed. if its an object reference the reference cannot be change.
4. Can be called without creating an object (class.method)	4. Cannot be overridden in subclasses but can be inherited.
5. Stored in the method area.	5. Stored in the heap of stacks
6. static int count = 0;	6. Final int max = 100;

TOPIC NAME: _____

Code:

```

class staticExample {
    static int count = 0;
    static void display() {
        System.out.println("static method
                           called");
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println(staticExample.count);
        staticExample.display();
        // works
    }
}

```

staticExample obj = new staticExample();

System.out.println(obj.count);

Output:

0
static method called

both works

③

IT-22023 (Re-add)

DAY: _____

TIME: _____

DATE: / /

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.io.*;

public class enamz {
    public static void main(String[] args) {
        int start, end;
        Scanner sc = new Scanner(System.in);
        start = sc.nextInt();
        end = sc.nextInt();
        for (int j = start; j <= end; j++) {
            int num = j;
            int num0 = num;
            List<Integer> arr = new ArrayList<>();
            while (num != 0) {
                int n = num % 10;
                num = num / 10;
                arr.add(n);
            }
            int sum = 0;
            for (int x : arr) {
                int mul = 1;
                for (int i = 1; i <= n; i++) {
                    mul = mul * i;
                }
                sum = sum + mul;
            }
            if (sum == num0) {
                System.out.print(num0 + ", ");
            }
        }
    }
}
```

3332

④

IT-22023

HOD'S TT

DAY: _____

TOPIC NAME: _____

TIME: _____

DATE: / /

Class variable, local variable, instance variable differs primarily in their scope, lifetime, and usage within a class.

1. Class variables:

- Declared with the static keyword
- Belong to the class, not instances of the class
- Shared by all objects of the class, meaning changes to the class variable affect all instances.
- can be accessed using the class name or through an instance

Example: static int count; for eg.

2. Instance variable

- Declared without the static keyword.
- Belong to a specific instance of the class.

TOPIC NAME: _____

- Each object of the class has its own copy of the instance variable.
- can be accessed using the object reference.

Example: int age;

3. Local variables:

- Declared within a method, constructor or block.
- Exist only during the execution of the method or block where they are declared.
- Do not retain their values between methods calls.
- Cannot be accessed outside the method, constructor, block.

Example: int sum=0;

Significance of the this keyword:

- The this keyword is a reference to the current object (the instance of the class) within a non-static method or constructor.
- It is used to distinguish between instance variable and local variable when they have the same name.

```
= class car {  
    String color; // local  
    car (String color) {  
        this.color = color;  
    }  
}
```

⑤ IT-22023 (re-add) .

TOPIC NAME: _____

DAY: _____

TIME: _____

DATE: / /

public class Exam1 {

 public static void main (String [] args)
 { int [] array = {1, 2, 4, 2, 1, 5, 6, 9};

 InnerExam obj = new InnerExam();

 int sum = obj.sumArray (array);

 System.out.println ("Sum = " + sum);

}

public class InnerExam {

 public int sumArray (int [] arr) {

 int sum = 0;

 for (int x : arr) {

 sum = sum + x;

}

 return sum;

}

}

Access Modifiers in Java:

Access modifiers in Java define the visibility (accessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in Java are:

1. public:

- Accessibility: Accessible from any other class, no matter what package the belongs to.

- Usage: used when we want a class, method, or variable to be accessible from anywhere in the program.

```
= public class MyClass {  
    public int numbers;  
}
```

⑥

IT-22023 (Re-add)

TOPIC NAME: _____

DAY: _____

TIME: _____

DATE: / /

Access Modifiers in Java:

Access modifiers in Java define the visibility (accessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in Java are:

1. public:
• Accessibility: Accessible from any other class, no matter what package the belongs to.

• Usage: used when we want a class, method, or variable to be accessible from anywhere in the program.

```
= public class MyClass {  
    public int numbers;  
}
```

Access Modifiers in Java:

Access modifiers in Java define the visibility (accessibility) of classes, methods and variables. They specify who can access a particular class, method, or variable. The four main access modifiers in Java are:

1. public:
 - Accessibility: Accessible from any other class, no matter what package the belongs to.
 - Usage: used when we want a class, method, or variable to be accessible from anywhere in the program.

```
= public class MyClass {  
    public int numbers;  
}
```

TOPIC NAME: _____

2. Private:

- Accessibility: Accessible only within the same class. Other classes cannot access private members, even if they belong to the same package.

- Usage: Used when you want to restrict access to class members for encapsulation purposes, usually to ensure data protection.

```
= public class MyClass {
```

```
    private int numbers;
```

```
    public void SetNumbers(int num)
```

```
{    numbers = num; }
```

3. 3rd level code =

3. 3rd level code =

3. Protected :

Accessibility: Accessible within the same package and by subclasses (even if they are in a different package)

Usage: used when you want a member to be available to subclasses but not accessible to every class.

= public class MyClass{

protected int numbers;

}

Q & A

=

Types of Variable in Java:

1. Instance Variable

These variable are declared within a class but outside any method or constructor.

- Each object of the class has its own copy of the instance variable

Example:

```
public class person {
```

```
    string name;
```

```
}
```

2. Class Variable:

- These variable are declared with the static keyword.

• They are shared among all instances of the class.

There is only one copy of a static variable, it can be accessed directly by the class name or an object.

= public class Counters {

 static int count = 0;

}

3. Local variable:

- These variables are declared within a method or block of code.
- They are only accessible within the method or block in which they are declared and are created when the method is invoked, destroyed when the method completes.

= public class Example {

 public void addNumbers() {

 int sum = 0;

 sum = 5 + 10;

 System.out.println(sum);
 }

TOPIC NAME: _____

DAY: _____

TIME: / / DATE: / /

Parameters:

- These variables are used in method declaration to accept values passed when calling the method.

```
public int add(int a, int b)
```

variables are known as parameters.

return type is also known as return type.

parameters are also known as arguments.

Additional notes on methods:

and private methods can be

in both private classes or

control methods, respectively.

Method overloading:

{ segment code } { }

{ } { without this block nothing

can be run

(or it's all

else (all) true

```
import java.util.Scanner;
public class Exam {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = sc.nextInt();
        if ((b * b - 4 * a * c) < 0) {
            System.out.println("No real root");
        } else {
            double r1 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
            double r2 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
            System.out.printf("%.2f\n", Math.min(r1, r2));
        }
    }
}
```

⑧

IT-22023 (Re-add)

TOPIC NAME: _____

DAY: _____

TIME: _____

DATE: / /

```
import java.util.Scanner;  
public class even1 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        character input = sc.next().charAt(0);  
        if (Character.isDigit(input)) {  
            System.out.println("It's a digit");  
        }  
        else if (Character.isWhitespace(input)) {  
            System.out.println("It's a whitespace");  
        }  
        else {  
            System.out.println("It's a letter");  
        }  
    }  
}
```

In java we can pass an array to a function by simply specifying the array type in the methods parameter.

— public class ArrayEx { int[] arr
 void method () {

}

}

main. ArrayEx ss = new ArrayEx();

ss.method (arr);

— from 22 MB window is used.

→ (22 MB) → optional (will be used only if needed)

→ (22 MB) → optional (will be used only if needed)

→ (22 MB) → optional (will be used only if needed)

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. This allows a subclass to modify the behavior of inherited methods.

Key rules of method overriding

1. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.
2. The method in the subclass cannot have a weaker access modifiers
3. The overriding method cannot throw a broader exception than the method in the superclass.

9. Dynamic method Dispatch: When a method is overridden, Java decides which version of the method to execute at run time, based on the actual object type.

The 'super' keyword is used to call the method of a superclass from a subclass, which is useful when you want to extend or modify the inherited behavior instead of completely replacing it.

```
class Animal {  
    void sound() {  
        System.out.println("Animal makes sounds");  
    }  
}  
class Dog extends Animal {  
    @Override
```

IT-22023

TOPIC NAME:

DAY:

TIME:

DATE: / /

void sound () {

super.sound();

System.out.println("Dog Barks");

}

public class Main {

public static void main (

String [] args) {

Dog d = new Dog();

d.sound();

}

}

Output:

Animals make sounds

Dog Barks

= Here super.sound () calls the

Sound() method of Animal,
and then the Dog class adds its
own behaviors.

⇒ Issues when overriding Methods

- Constructors cannot be overridden
- Constructors are not inherited,
→ constructors are not overridden. However,
so they cannot be overridden.
- A subclass constructor can explicitly
call a superclass constructor using
super().

• Constructors call order.
→ When creating an object of
a subclass, the constructor of
the superclass is called first.
⇒ If a superclass does not have
a no-argument constructor

TOPIC NAME: _____

the subclass must explicitly call
 a parameterized constructor
 using 'super()';

```

class Parent {
    Parent() {
        System.out.println("Parent
constructor");
    }
}

class Child extends Parent {
    Child() {
        super();
        System.out.println("Child
constructor");
    }
}

public class Again {
    public static void main(String[] args) {
        Child e = new Child();
    }
}
  //
```

Potential Issues with overriding:

• Incorrect 'super()' uses:

→ If the superclass has a parameterized constructor but the subclass does not explicitly call it, a compilation error occurs.

• Final methods cannot be overridden:

If a method is declared as final in the superclass, it cannot be overridden.

• Private methods are not inherited:

→ Private methods in a superclass are not visible in the subclass, so they cannot be overridden.

(10)

IT-22023

Score - IT

DAY: _____

TIME: _____

DATE: / /

TOPIC NAME: _____

Difference Between Static and non-Static Members

Static members

non-static members.

① Belong to the class rather than object	1. Belong to a specific object of the class.
② Can be accessed using the class name	② Can only be accessed through an object.
③ Allocated once in a memory and shared across all objects	③ Each object gets its own copy.
④ Classname.method() or object.method()	④ object.method()
⑤ changing a static variable affects all objects	⑤ changing a non-static variable affects only that object.

```

import java.util.Scanner;
public class even {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        int length = s.length();
        if (length % 2 == 0) {
            int a = 0, b = length - 1;
            while (a < b) {
                if (s.charAt(a) != s.charAt(b)) {
                    System.out.println("not a palindrome");
                    return;
                }
                a++;
                b--;
            }
            else {
                int a = 0, b = length - 1;
                while (a <= b) {
                    if (s.charAt(a) == s.charAt(b)) {
                        System.out.println("NOT a
palindrome");
                    }
                    a++;
                }
                System.out.println("Its a pallindrome");
            }
        }
    }
}

```

(11)

IT-22023 (Re-add)

TOPIC NAME:

DAY:

TIME:

DATE: / /

Abstraction: Abstraction is the process of hiding implementation details and showing only essential features. In Java abstraction is achieved using abstract classes and interface.

- Focuses on what an object does not how it does.
- Achieved using the 'abstract' keyword.
- Abstract methods must be implemented by subclasses.

// abstract class vehicle {

 abstract void start();

 class car extends vehicle {

 void start() {

 System.out.println("Car starts")

 with a key.");

//

Encapsulation: Encapsulation is the process of wrapping data (variables) and methods into a single unit (class) and restricting direct access to it. Data is hidden using 'private' variable.

- Accessors (getter) and Mutators (setter) method are used for controlled access.

```
// class person {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
public class Main {
    public static void main() {
        person p = new person();
        p.setName("Alice");
        System.out.println(p.getName());
    }
}
```

Abstract class

- A class that can have abstract and concrete methods.

- can have both abstract and non-abstract methods.
- can have instance variables.

- A class can extend only one abstract class

- used when classes share common behavior

Interface

- A blueprint that only contains abstract methods

- Only abstract methods
- Variables are public and static and Final

- A class can implement multiple interfaces

- used to achieve full abstraction and multiple inheritance.

```
import java.util.*;  
public class Mainclass{  
    public static void main(String[] args){  
        SumClass work1 = new SumClass();  
        work1.printResult();  
        System.out.printf("%.2f", work1.calculateSum());  
        DivisionMultipleClass work2 = new DivisionMultiple-  
-Class();  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter 2 numbers to calculate  
gcd");  
        int n = sc.nextInt();  
        int y = sc.nextInt();  
        work2.printResult();  
        System.out.println("Enter 2 numbers to calculate  
lcm");  
        x = sc.nextInt();  
        y = sc.nextInt();  
        System.out.println("LCM => " + work2.LCM(n,y));  
    }  
}
```

TOPIC NAME: _____ DAY: _____
TIME: _____ DATE: / /

NumberConversionClass wonk3 = new NumberConversionClass();

x = sc.nextInt();

System.out.println(wonk3.decToBinary(n));

x = sc.nextInt();

System.out.println(wonk3.decToOct(n));

x = sc.nextInt();

System.out.println(wonk3.decToHex(n));

CustomPrintClass wonka = new CustomPrintClass();

String s = sc.nextLine();

System.out.println(wonka.pr(s));

}

=

((full name - x))

((full name - y))

((full name - z))

== Public class NumberConversionClass {

Public String decToBinary (int num) {

return Integer. toBinaryString (num);

}

public String decToOct (int num) {

return Integer. toOctalString (num);

}

public String decToHex (int num) {

return Integer. toHexString (num). to

-UpperCase ();

}

}" Class BaseClass {

void printResult () {

System.out.print ("Your answer : ");

}

}

}

TOPIC NAME: _____

DAY: _____
TIME: _____ DATE: / /

// public class SumClass extends BaseClass {

 double sum = 0;

 public double calculateSum() {

 for (double i = 1; i >= 0.1; i = i - 0.1) {

 sum = sum + i;

 }

 return sum

}

// public class DivisorMulClass extends
BaseClass {

 public int GCD(int a, int b) {

 while (b != 0) {

 int temp = b;

 b = a % b;

 a = temp;

 return a;

 public int LCM(int a, int b) {

 return (a * b) / GCD(a, b);

TJ-22023

TOPIC NAME : _____ DAY : _____
TIME : _____ DATE : / /

DAY: _____

```
// public class CustomPrintClass {  
    public String pr(String s) {  
        s = "###" + s + "###";  
        return s;  
    }  
}
```

(T.G.1118) with two white sides

(O) fast forward 32 = 10 steps

(1) 2401) 3203 1000 = 23 801) 3203

168 To (1915-16) referring two messages

$$(6) \text{ left } 3b + 1 \leq c$$

In

IT-22023(Re-add)

TOPIC NAME: _____

DAY: _____

TIME: _____

DATE: / /

In Java, BigInteger class is used to handle very large numbers that exceed the limit of primitive type types like 'int', 'long'. It provides methods for Arithmetic operations, bit manipulation, GCD ...

```
// Import java.util.Scanner;  
public class mainClass {  
    public static void main(String [] args)  
    {  
        Scanner sc=new Scanner(System.in);  
        int n=sc.nextInt();  
        BaseClass bs=new BaseClass();  
        System.out.println("Factorial of "+n+"  
is => "+bs+fact(n));  
    }  
}
```

```
import java.math.BigInteger;
public class BaseClass{
    public BigInteger fact(int n){
        BigInteger ff=BigInteger.ONE;
        for(int i=1; i<=n; i++){
            ff=ff.multiply(BigInteger.valueOf(i));
        }
        return ff;
    }
}
```

//