IT23053

(1)

```
import java.io.*;
import java.util.*;
public class exam1{
    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(outpu
    read.useDelimiter(",");
    List<Integer> numbers = new ArrayList<>();
    while (read.hasNextLine()){
    numbers.add(read.nextInt());}
    if (numbers.isEmpty()){
    write.println("No number found");
    read.close();
    write.close();
    return;}
    int maxnum = Collection.max(numbers);
    write.println("Maximum Number: " + maxnum);
    for (int x: numbers){ int sum = (x*(x+1))/2;
    write.println(sum + ", ");}
    write.close(); read.close(); }}
```

Detailed Differences between static and final fields in java:

static	final
A member (field or method) marked as static belongs to the class rather than any specific instance	A member (field or method) marked as final belongs to instance rather than any specific class.
Applies to variables, methods, blocks, nested classes	Applies to variables, methods, classes.
Shared by all instances of the class. A single copy exits in memory	Once assigned, its value cannot be changed. If its an object, the refarance cannot be change.
can be called without creating an object (class.method)	cannot be overridders in subclasses but can be inherited.
stored in the method area.	stored in the heap of stack.
static int count = 0;	final int max = 100;

codei

```
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 ();
        staticExample obj = new staticExample ();
        System.Out.Println (obj.count);
    }
}
```

Output:

static method called.

IT23053

(3)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.io.*;
Public class exam2{
 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 r = num%10;
 num = num/10; add.add(r);
 }
 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 + " ");
 }
}
```

IT23053

(4)

Class variable, local variable, instance variable differ primarily in their scope, life-time, 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 object 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;

(2) Instance variable:

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

- Declared within a method, constructors or block.
- Exist only during the execution of the method or block where they are declared.
- Do not retrain their values between method's 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 constructors.
- It is used to distinguish between instance variable and local variable when they have the same name.

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

(5)

```
Public class exam 1 {
    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;
    }
}
```

(6) 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.

Comparison of Access modifiers:

1. Public:

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

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

```
public class Myclass {
    public int numbers;
}
```

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 we want to restrict access to class members for encapsulation purposes, usually to ensure data protection.

```
public class MyClass {
    private int number;
    public void setNumber(int num) {
        number = num;
    }
}
```

3. Protected:

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

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

```
public class myclass {
    protected int number;
}
```

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.

Exm: public class person {
    String name;
}

(2) Local variable:

- These variable 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

method is invoked, distroyed when the method completes.

Exm! Public class Example{

Public void addNumbers(){

int sum=0;

sum = sum+10;

S.out(sum);}}

3. static (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.

IT23053

(7)
```
import java.util.Scanner;
Public class exam1 {
  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 x1 = (-b + Math.sqrt(b*b-4*a*c))
                     /(2*a);
    double x2 = (-b - Math.sqrt(b*b-4*a*c))
                     /(2*a);
    System.out.printf("%.2f\n", Math.
                     min(x1, x2));
  }
}
```

(8)
```
import java.util.Scanner;
Public class exam1 {
  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 ("Its a Digit ");
    }
    else if (character.is whitespace (input)) {
      system.out.println ("Its a whitespace");}
    else {system.out.println ("Its 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 {
    Public void method (int[] arr) {
      :
    }
main
    ArrayEx ss = new ArrayEx ( );
    ss.method (Arry);
```

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) class name. 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.

IT23053

```
import java.util.Scanner;
Public class exam1 {
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 ;
  while (a<b) {
  if (s.charAt (a) != s.charAt (b)){
      System.out.println ("Not a palindrom");
      return ; }
      a++; b = -; }}
  else {int a =0, b = length-1;
  while (a<=b) { if (s.charAt (a) == s.charA
  { system.out.println ("Not a Palindrom"
      return ; }
      a++;
      b--;
  System.out.println ("Its a Palindrom ");
}
}
```

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 stard ();
}
class car extends vehicle {
    void start () {
        system.out.println ("car starts with a key");
    }
}
```

IT23053

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
- Accesson (getter) and Mutator (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 {
    psvm () {
        person p = new person ();
        p.setName ("Alice");
        s.out (p.getName);
    }
}
```

IT23053

Abstract class	Interface
A class that can have abstract and concrete methods	A blueprint that only contains abstract methods
can have both abstract and non-abstract methods	only abstract methods
can have instance variable.	variables are public, static and final.
A class can extend only once abstract class.	A class can implement multiple interfaces.
used when classes share common behaviour	used to achieve full abstraction and multiple inheritance.