Assignment 01:

ID: IT23030:


```
import java.util.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.printwriter;
public class ASS01{
    public void main(String[] args){
try{ File file=new file("/mnt/OOP/inputASS01.txt");
    if(!file.exists(){
        System.out.println("Error");
        return;
    }
    Scanner input = new Scanner(file);
    ArrayList<Integer> num=new ArrayList<>();
    while(input.hasNextLine()){
        String line=input.nextLine().trim();
        if(line.isEmpty()) continue;
        String[]values=line.split(",");
        for(String val:values){
            val=val.trim();
            if(!val.isEmpty()){
            num.add(Integer.parseInt(val));
}
```

```
        }
    }
input.close();
int sum = num.stream().mapToInt(Integer::intValue
int maxnum = Collections.max(num);
File outfile = new File("/mnt/OOP/outputAss02.txt");
writer.println("Series sum is: ", sum);
writer.close;
}
catch (FileNotFoundException e) {
    System.out.println("Error");
}
catch (Number
    }
}
}
}
```

Assignment 02:

ID: IT23030:

	Static	final
Definition:	Belongs to the class rather than instances (obj)	Once assigned, its value cannot be changed
Variables:	Shared all instances of the class	cannot be re assigned
methods:	Can be called without instance	cannot be overridden in subclass
Access:	Accessed via the class name	Accessed like a normal

Assignment 03:

```
import java.util*;
public class ASS03{
    static void main myfact(){
        Scanner input = new Scanner(System.in);
        System.out.println("Please Enter a number:");
        int num = input.nextInt();
        int tmp = num;
        int sum = 0;
        while (tmp > 0){
            int fact = 1;
            int dg = tmp % 1;
```

```
for (int i=1; i<=dg; i++){
    fact*= i;
  }
sum+= fact;
  tmp/=10;
  }
System.out.println("Factorial of number sum is: "+sum
if number == sum{
      System.out.println("Yes");
  else{
      System.out.println("No");
    }
}
public static void main(String[] args){
  my fact();
}
}
```

Assignment 04:

Instance variable	Local var
They are defined in class but outside the body of methods	They are defined as a type of variable

Assignment 04:

class	Instance	Local variable
Available to all instances of a class	Tied on object	Declared inside a method or block
Stored in method area	Stored in heap memory	Stored in stack memory
depends on data type	depends on data type	Stored in stack memory
can have access modifiers	can have access modifiers	no access modifiers

Assignment 05:

```
import java.util.*
public class Ass05 {
    public class int calculateSum(int[] arr) {
        int sum=0;
        for(int n:arr){
            sum+=n;
        }
        Return sum;
    }
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the size of Array");
        int size = input.nextInt();
        int[] arr = new int[size];
        System.out.println("Enter the elements of array");
        for(int i=0; i<size; i++){
            arr[i] = input.nextInt();
        }
        int total_sum = calculateSum(arr);
        System.out.println("Total sum is: "+total_sum);
    }
}
```

Assignment 06 :

Access modifiers: Access modifiers in java are keywords that define the visibility (scope) of a class, method, or variable.

Comparison of Access modifiers :

modifiers	Accessibility within same class	Accessibility within same Package	Accessibility in Subclass	Accessibility outside package
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	~~Yes~~ No
private	Yes	~~Yes~~ No	~~Ye~~ No	No
default	Yes	Yes	No	No

Example :

(i) public :

```
pubic class Example {
    public int num = 10;
}
```

(ii) Protected :

```
class Parent {
    protected int num = 20;
}

class child extends Parent {
    void display() {
        System.out.println(num);
    }
}
```

(iii) private :

```
class Example {
    private int num = 30;
    void show() {
        System.out.println(num);
    }
}
```

(iv) default :

```
class Example {
    int num = 40;
}
```

Types of variable in Java → instance variable
→ class
→ Local

```
class Example {
    int instanceVar = 10;
    static int staticVar = 20;
    void method() {
        int localVar = 30;
        System.out.println("Local var : " + localVar);
    }
```

Assignment 07:

```
import java.util.*;
public class ASS07 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double a, b, c;
        System.out.println("Enter coefficients : ");
        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;
```

```
double root2=(-b-Math.sqrt(d))/2;
double res2=Math.min(root1, root2);
System.out.println("The smallest positive root is:"
                   +res);
} else {
    System.out.println("Zero real root exist.");
input.close();
}
}
```

Assignment 08:

```
import java.util.*;
public class ASS08 {
    public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the string:");
    String s=input.nextLine();
    int letter=0, digit, space=0;
for(int i=0; i<s.length(); i++){
    if(s.charAt(i)==' '){
        space++;
    }else if(s.charAt(i)>='0'&&s.charAt(i
    {digit++;
    } else {
        letter++;
```

```
        input.close();
    }
}
```

Assignment 09:

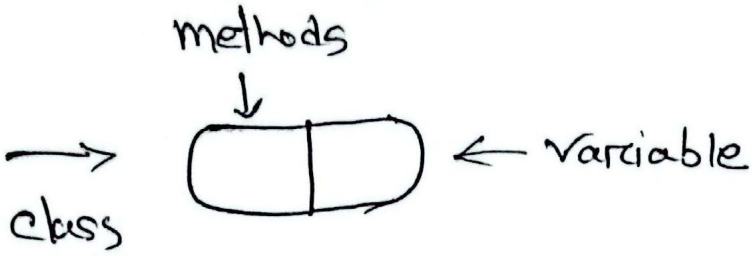
Method overriding is a feature in java that allows a subclass to provide a specific implementation of a method that already defined in its superclass.

```
class Animal {
    public void displayInfo() {
        System.out.println("I am an animal");
    }
}

class dog extends Animal {
    @Override
    public void displayInfo() {
        System.out.println("This is a dog!");
    }
}

class main{
    public static void main(String[] args) {
        Dog d1 = new Dog();
        d1.displayInfo();
    }
}
```

Encapsulation



Benifits:

- read-only or write-only
- control over the data
- data hiding.
- easy to test.

Example:

```
public class person {
    private String name;
    private int age;
    // Getters
    public String getName() {
        return name;
    }
    // Setters
    public void setName(String newName) {
        this.name = newName;
    }
}
```

main class:

```
public class main {
    pubic static void main (string[] args) {

        Person myObj = new Person();
        System.out.println(myobj.name);
        myObj.setName("Shuvon");
        system.out.println(myObj.getName());
    }
}
```

Method Overloading:

If a class has multiple method having same name but different in parameters, it is known as method overloading.

Example:

```
public class Practise extends class A {
    public static void main (String[] args)
    { Practise pr = new Pract
      add ( );  Practise.(
    }
public int add (int a, int b) {
    return a+b;
}
```

public int add (string a, ~~int~~ string b) {

return a+b;

}

04.12.24

~~Inheritance~~:

Static Vs public:

Static	Public
• It can be accessed without creating an object	• Can not be access without creating object.

Static Vs pubic

```
pubic class Main ( )
    static void myStaticMethod ( ) {
        System.out.println ("Static methods can
        be called without creating object");
    }
```

```
//public method
public void myPublicMethod() {
    System.out.println("Public methods must be
    called by creating objects.
    }

//Main Method
public static void main(String[] args) {
    myStaticMethod();
    Main myObj = new Main(); //Create an
                              Obj of main
    myObj.myPublicMethod();
    }
}
```

~~publi~~ Problem: Find out factorial sum of an int number using two class.

```
import java.util.*
public class Factorial {
    public static void main(String[] args)
    Scanner new input= new Scanner(S
    System.out.println("Please, Enter a number
    int num= input.nextInt();
    findFactorial (int num);
    }
}
```

second class:

```
class Main {
public void findFact (int number) {
    int count=0;
    for(int i=0; i<=number; i++) {
        int tmp=i;
        int sum=0;
        while(tmp>0) {
            int fact=1;
            int digit=tmp%10;
            for(int j=1; j<=digit; j++){
                fact*=j;
```

```
sum += fact;
    temp /= 10;
  }
  if(sum == i) {
      count++;
      System.out.println("Factorion number is:"
  }
System.out.println("Total : " + count);
  }
}
```

Math Practice

```
import java.util.*;
import java.lang.Math;
public class MathPractice {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.
        System.out.println("Equation 1: Calculate the
                height of a right triangle.");
        System.out.print("Enter base (b): ");
        double b = scanner.nextDouble();
        System.out.print("Enter angle (theta in
                degrees): ");
        double theta = scanner.nextDouble();
        double height = b * Math.tan(Math.toRadians
        System.out.println("Height: " + height);
        System.out.println("\nEquation 2: Compo
                Interest calculation.");
        System.out.print("Enter Principal (P): ");
        double p = scanner.nextDouble();
        System.out.println("Total Amount: " + A);
```

* Java class attributes: ~~variable ir~~ কাজকে variable গুলোকে Java class attributes বলে।

```
public class Main {
    int x = 5;
    int y = 3;
}
```

x, y are java class attributes

For accessing attributes
(i) create an object
(ii) object.variable name

Example:

```
class Main {
    int x = 30;
    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
    }
}
```

Java method: (function in other languages)

```
class main {
    static void myMethod() {
        System.out.println("Hello world");
    }
}
```

call myMethod():

```
class second {
    public static void main(String[] arg
        myMethod();
    }
}
```

Java constructors: a special method that is used to initialize object

```
public class Main {
    int x; // create a class attribute
// Create a class constructor for the main
    class
    public Main() {
        x = 5;
    }
public static void main(String[] args) {
    Main myObj = new Main();
    System.out(myObj.x);
    }
}
```

Java Inhercitanee: Inhercitanee is one of the key feacture of OOp that allows us to create a new class forcom an existing class.

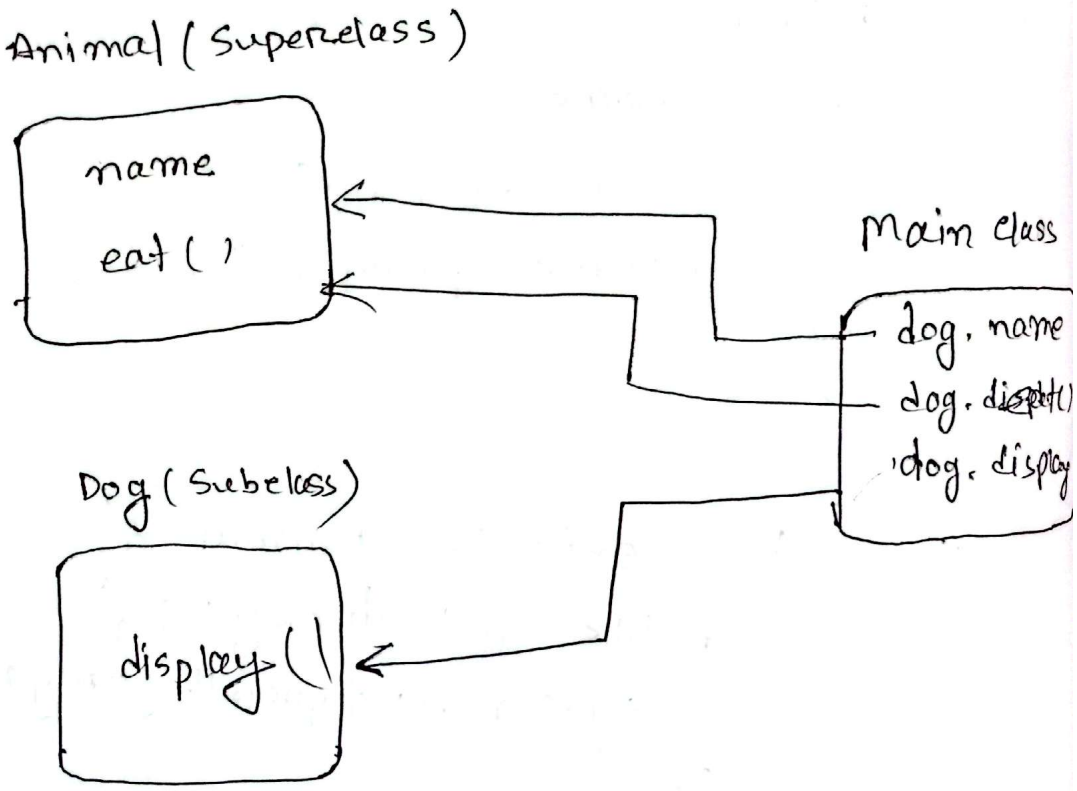
```
class Animal {
    // methods and fields
}
// to perform inhercitanee
class Dog extends Animal {

}
```

Example:

```
class Animal {
    string name;
    public void eat () {
        system.out.println("I cam eat");
    }
}
class Dog extends Animal {
    public void display() {
        system.out.println("my name is " +
                           name);
    }
}
```

```
class main {
    public static void main (String[] args)
        Dog dog = new Dog();
        dog.name = "Jhon";
        dog.display();
        dog.eat();
    }
}
```



Java method Overriding: Override the method of super class,

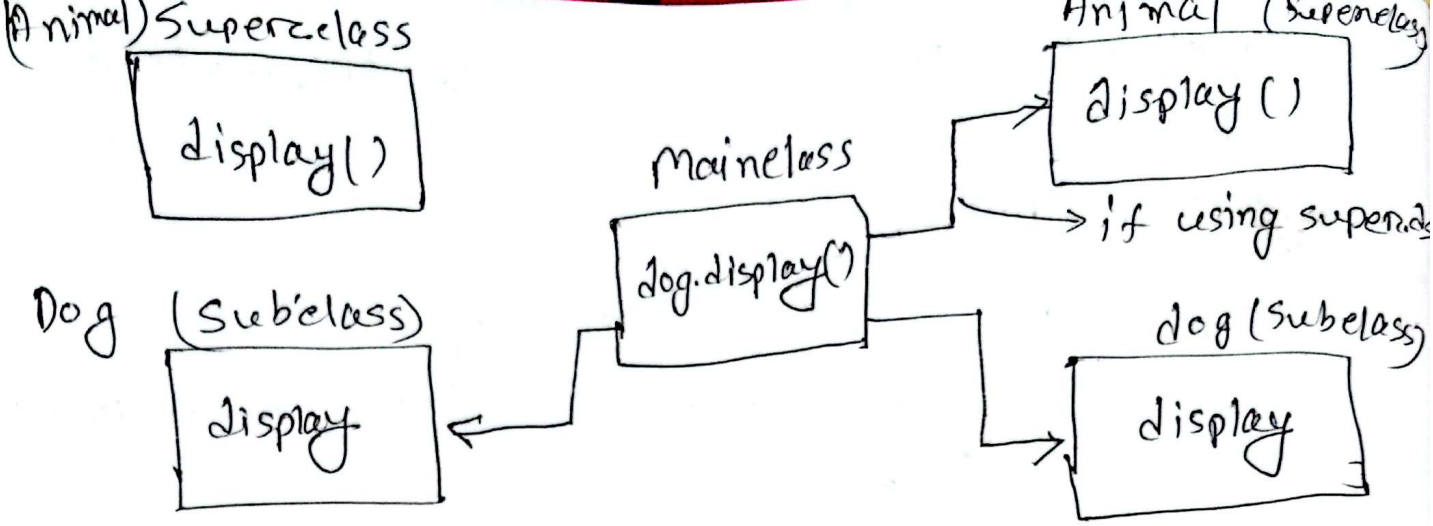
```
class Animal {
    public void display() {
        System.out.println("Dog is an Animal
    }
}

class Dog extends Animal {
    @Override
    public void display() {
        super.display();
        System.out.println("This dogs name is "
                           nam);
    }
}

class methodOverride {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.nam = "Jon";
        dog.display();
    }
}
```

Output : This dogs name is JON

If we use super.display(); Output : Dog is an Anim
This dogs nam is



Java abstract method: Which method does not have body

```
abstract void display();
```