Day-4 Date: 01/05/2023

OOPJ Notes

Constructors of the class and Constructor Overloading

```
class Student {
    int Rno;
    String Name;
    String Address;

    Student()
    {
        Rno=0;
        Name="No Name";
        Address="Laapta";
    }
    Student(int r, String n)
    {
        Rno=r;
        Name=n;
        Address="No Address";
    }

    Student(int r, String n, String a)
    {
        Rno=r;
        Name=n;
        Address=a;
    }

    void Display()
    {
        System.out.println("Roll No:    "+Rno);
        System.out.println("Name:    "+Name);
        System.out.println("Address:    "+Address);
    }

    public static void main(String[] args) {
        Student s1= new Student();  //Constructor with no arguments
        Student s2= new Student(103,"Prashat"); //Constructor with two-argumemts
        Student s3=new Student(101,"Malkeet","Kharghar");  //constructor with 3 arguments

        s1.Display();
        s2.Display();
        s3.Display();

    }

}
```

constructor chaining

Introduction to 'this' keyword

- This is a keyword in java.
- it always contain address of the current object
- it helps in differentiating local variables with instance variables if both have same name.

```
class Student {
    int Rno;
    String Name;
    String Address;

    void Setdata(int Rno, String Name, String Address)
    {
        this.Rno=Rno;
        this.Name=Name;
        this.Address=Address;
    }

    void Display()
    {
        System.out.println("Roll No:    "+Rno);
        System.out.println("Name:    "+Name);
        System.out.println("Address:    "+Address);
    }

    public static void main(String[] args) {
        Student s1= new Student();
        s1.Setdata(101, "Malkeet","Kharghar");
        s1.Display();
    }

}
```

Method overloading

- It is done at compile time;
- we can keep same name of the function and change their return type as well as list of arguments.
- function with same name and different list of arguments or return type is known as overloaded function.
- It is also know as compile time polymorphism.

```
class MethOL {

    double result;
    double add()
    {
        int a=10;
        int b=20;
        result=a+b;
        return result;

    }
    int add(int p, int q)
    {
        result=p+q;
        return (int)result;
    }
    float add(float s, float t)
    {
        result=s+t;
        return (float) result;
    }

    public static void main(String[] args) {

        MethOL m1=new MethOL();
        double res;
        res=m1.add();
        System.out.println("Double result is:   "+res);
        int resint;
        resint=m1.add(100,200);
        System.out.println("Int result is:  "+resint);
        float resfloat;
        resfloat= m1.add(45.56f, 56.78f);
        System.out.println("Float result is:    "+resfloat);

    }
}
```

Demo of Classes (Scanner, Date, Calendar, LocalDate, LocalTime, LocalDateTime and SimpleDateFormat)

static field, static method, static block

```
class Sttic {
static
{
    System.out.println("I am static block-1");
}
static
{
    System.out.println("I am static block-2");
}
static void Demo()
{
    System.out.println("am Demo Function and i am static");
}
public static void main(String[] args) {

    //Sttic.Demo();
    //System.out.println("Am main of class");
}
}
```

```
class Sttic {
    static int a=100;
    int b;
    void show()
    {
        System.out.println("value of static a="+a+"value of instance variable b="+b);
    }
static
{
    System.out.println("I am static block-1:"+a);
}
static
{
    System.out.println("I am static block-2:"+a);
}
static void Demo()
{
    System.out.println("am Demo Function and i am static"+a);
}
public static void main(String[] args) {

    Sttic s1= new Sttic();
    s1.show();
    System.out.println("I am Main"+a);
    System.out.println("I am Main"+s1.b);
}
}
```

public, private access modifier

- private property of the class can only be accessed inside that class.

```
class Sttic {
    private int a;
    private void Demo()
    {
        System.out.println("Value of a="+a);
    }
}
class S睡ticTest
{
    public static void main(String[] args) {
        Sttic s1=new Sttic();
        s1.a=100; //Not valid
        s1.Demo(); //Not valid
    }
}
```

Introduction to Getters and Setters

- Getters and Setters are used to access the private instance variable of the class outside of that class.
- These are used to achieve encapsulation.

```
class Sttic {
    private int a;

    void setA(int b)
    {
        a=b;
    }
    int getA()
    {
        return a;
    }
    void Demo()
    {
        System.out.println("Value of a="+a);
    }



}
class StticTest
{
    public static void main(String[] args) {
        Sttic s1=new Sttic();
        s1.setA(100);
        s1.Demo();
    }
}
```

path and CLASSPATH

- path is used by Operating System to locate the application.
- CLASSPATH is used by application like javac , javap to locate the class path of .class file.

Introduction to packages