JAVA OOP

(Object Oriented Programming)

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.
It simplifies software development and maintenance by providing some concepts:

1. **Object**
2. **Class**
3. **Inheritance**
4. **Polymorphism**
5. **Abstraction**
6. **Encapsulation**

Object:

Any entity that has state and behavior is known as an object.

Class:

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance:

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism:

If *one task is performed in different ways*, it is known as polymorphism.
In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction:

Hiding internal details and showing functionality is known as abstraction.
In Java, we use abstract class and interface to achieve abstraction.

Encapsulation:

Binding (or wrapping) code and data together into a single unit are known as encapsulation.

Objects and Classes in Java

Object:

An entity that has state and behavior is known as an object.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is a class in Java:

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**

- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

Instance variable in Java:

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.

Method in Java:

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java:

The new keyword is used to allocate memory at runtime. All objects get memory in the Heap memory area.

3 Ways to initialize object:

There are 3 ways to initialize objects in Java.

1. By reference variable
2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

- ➔ Initializing an object means storing data into the object.

```
class Student{
 int id;
 String name;
}
public class TestStudent2{
 public static void main(String[] args){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

2) Object and Class Example: Initialization through method:
Initializing an object means storing data into the object.

```
class Student{
 int id;
 String name;
}
public class TestStudent2{
 public static void main(String[] args){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

3) Object and Class Example: Initialization through a constructor

There are many ways to create an object in java.
They are:

- By new keyword

- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

Anonymous object:
Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

Calling method through a reference:

```
Calculation c=new Calculation();
c.fact(5);
```

Calling method through an anonymous object:

```
new Calculation().fact(5);
```

Program to demonstrate the anonymous object:

```
class Calculation
{
void fact(int n)
{
int i=1,f=1;
for(i=1;i<=n;i++)
{
f=f*i;
}
System.out.println("factorial is"+f);
}
public static void main(String args[])
{
new Calculation().fact(5);
}
}
```

Example program for Classes and objects:

```
class Account
{
        int acc_no;
        String name;
        float amount;

        void insert(int a,String n,float am)
        {
                acc_no=a;
                name=n;
                amount=am;

        }

        void deposit(float amt)
        {
                amount=amt+amount;
                System.out.println("Amount deposited:"+amt);
        }

        void withdraw(float amt)
        {
                if(amt<amount)
                {
                        System.out.println("InSufficient Balance");
                }
                else
                {
                        amount=amount-amt;
                        System.out.println("Amount withdrawn:"+amt);
                }

        }
        void balance()
        {
                System.out.println("Balance is "+amount);
        }
        void display()
        {
                System.out.println(acc_no+" "+name+" "+amount);
        }

}
class TestAccount
{
        public static void main(String args[])
        {
                Account a1=new Account();
                a1.insert(2345, "Manoj", 10000);
                a1.deposit(20000);
                a1.balance();
                a1.withdraw(20000);
a1.balance()
        }
}
```

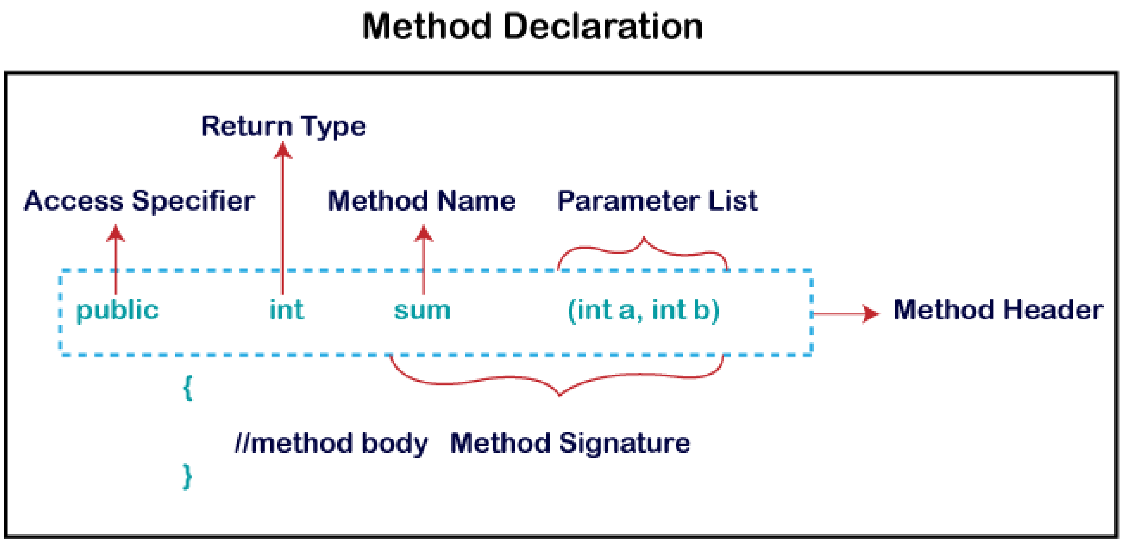
METHODS IN JAVA:

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code.

Method Declaration:

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

It has six components that are known as method header, as we have shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides four types of access specifier:

- Public: The method is accessible by all classes when we use a public specifier in our application.
- Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- Protected: When we use a protected access specifier, the method is accessible within the same package or subclasses in a different package.
- Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use a void keyword.

Method Name: It is a unique name that is used to define the name of a method. A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, leave the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Types of Method:

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method:

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method.

User-defined Method:

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

Static method

- ➔ A method that has a static keyword is known as static method.
- ➔ In other words, a method that belongs to a class rather than an instance of a class is known as a static method.
- ➔ The main advantage of a static method is that we can call it without creating an object.
- ➔ It can access static data members and also change the value of it.

Instance Method:

- ➔ The method of the class is known as an instance method.
- ➔ It is a non-static method defined in the class.

➔ Before calling or invoking the instance method, it is necessary to create an object of its class.

CONSTRUCTORS IN JAVA

Need of Constructors:
Whenever an object is created for a particular class by default the JVM provides default values to the instance variables in the class.
Therefore, in order to provide programmer defined values instead of default values provided by the JVM we use the concept constructors.

Definition of constructor:
A constructor is a special member method which will be called by the JVM implicitly for placing user/programmer defined values instead of placing default values.

Rules for creating Java constructor:

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

Types of Java constructors:

There are two types of constructors in Java:
1. Default constructor (no-argument constructor)
2. Parameterized constructor

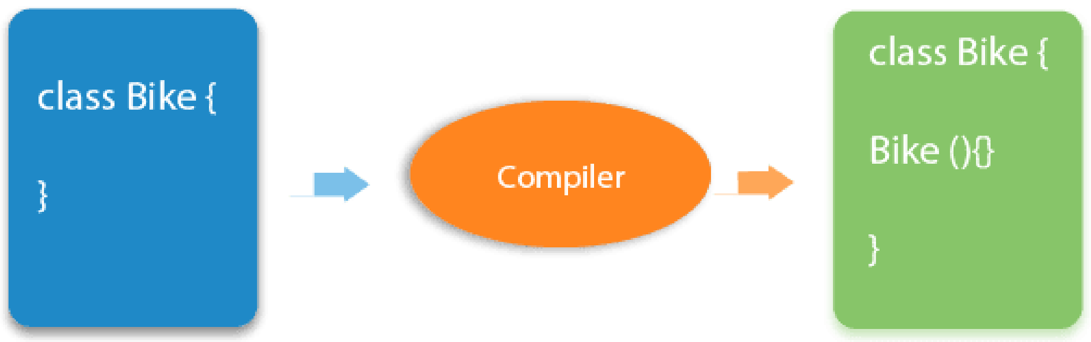
Java Default Constructor:

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){}
```

If there is no constructor in class, the compiler automatically creates it.



//default Constructor

Example for java default constructor:

```
class DefaultConstructor
{
int rno;
String name;
void display()
{
System.out.println("name:" +name+" and rollno:"+rno);
}
public static void main(String args[])
{
DefaultConstructor d1=new DefaultConstructor();
DefaultConstructor d2=new DefaultConstructor();
d1.display();
d2.display();
}
}
```

//We cannot create any constructor here so by default the constructor is created.

Java Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor.

Example for parameterized Constructor:

```
class Student5

{

int id;

String name;

 Student5(int i,String n)

 {

 id=i;

 name=n;

 }

 void display()

 {

 System.out.println("Student id:"+id);

 System.out.println("Student name:"+name);

 }

 public static void main(String args[])

 {

 Student5 s1=new Student5(510,"Manoj");

 Student5 s2=new Student5(520,"Sai");
```

```
  s1.display();

  s2.display();

  }
```

Constructor Overloading in Java:

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
They are arranged in a way that each constructor performs a different task.
They are differentiated by the compiler by the number of parameters in the list and their types.

Example for Constructor Overloading in Java:

```
class Student6
{
int age;
int rno;
String name;
Student6(int r,String n)
{
rno=r;
name=n;
}
Student6(int r,String n,int a)
{
rno=r;
name=n;
age=a;
}
 void display()
{
```

```
System.out.println("Name:"+name);
System.out.println("Rollno:"+rno);
System.out.println("Age:"+age);
}
public static void main(String args[])
{
Student6 s1=new Student6(510,"Manoj");
Student6 s2=new Student6(510,"Sai",20);
s1.display();
s2.display();
}
}
```

Java Constructor	**Java Method**
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be the same as the class name.	The method name may or may not be the same as the class name.

Java static keyword.

The **static keyword** in Java is used for memory management mainly.

We can apply static keywords with variables, methods, blocks and nested classes. The static keyword belongs to the class rather than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

Static variable.

The static variable can be used to refer to the common property of all objects.The static variable gets memory only once in the class area at the time of class loading.

Example for static variable:

```
class Student7

{

int rno;

String name;

static String college="SRKR";
```

```
Student7(int r,String n)

{

rno=r;

name=n;

}

void display()

{

System.out.println("Name:"+name);

System.out.println("Rollno:"+rno);

System.out.println("College:"+college);

}

public static void main(String args[])

{

Student7 s1=new Student7(510,"Manoj");

Student7 s2=new Student7(520,"Madhu");

Student7 s3=new Student7(530,"sai");

s1.display();

s2.display();

s3.display();

}
```

2) Java static method:

If you apply a static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data members and can change the value of it.

Example for Static method:

```
class Student8

{

    int rollno;

    String name;

    static String college = "SRKR";

    static void change()

         {

    college = "SRKREC";

    }

    Student8(int r, String n){

    rollno = r;

    name = n;

    }
```

```
     void display()
           {
                    System.out.println(rollno+" "+name+" "+college);
           }

    public static void main(String args[])
           {
    Student8.change();
    Student8 s1 = new Student8(510,"Manoj");
    Student8 s2 = new Student8(520,"Madhu");
    Student8 s3 = new Student8(530,"Sai");
    s1.display();
    s2.display();
    s3.display();
    }
}
```

Restrictions for the static method:

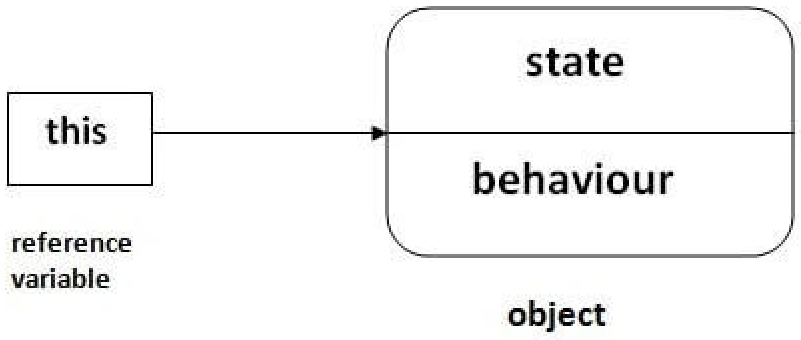
There are two main restrictions for the static method. They are:

1. The static method can not use non-static data members or call non-static methods directly.
2. this and super cannot be used in static context.

this keyword in java:

There can be a lot of usage of **java this keyword**.

In java, this is a **reference variable** that refers to the current object.



Uses of this keyword:

1) this: to refer current class instance variable

This keyword can be used to refer to a current class instance variable.

If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

In the above example, parameters (formal arguments) and instance variables are the same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}
}
```

Output:

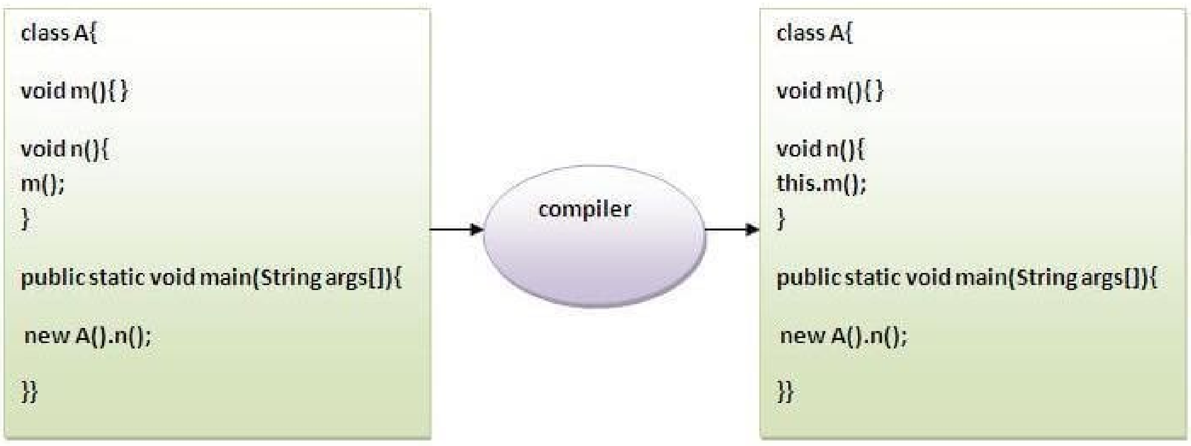
```
111 ankit 5000

112 sumit 6000
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

2) this: to invoke current class method

You may invoke the method of the current class by using **this** keyword. If you don't use **this** keyword, the compiler automatically adds this keyword while invoking the method. Let's see the example



```
class A{

void m(){}

void n(){
m();
}

public static void main(String args[]){

 new A().n();

}}
```

```
class A{

void m(){}

void n(){
this.m();
}

public static void main(String args[]){

 new A().n();

}}
```

```
class A
{
void m()
{
System.out.println("hello m");
}
void n()
{
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}
}
```

3) this() : to invoke current class constructor

This() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Real usage of this() constructor call:

The this() constructor call should be used to reuse the constructor from the constructor.

It maintains the chain between the constructors i.e. it is used for constructor chaining.

Let's see the example given below that displays the actual use of this keyword.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+"
"+fee);}
}
class TestThis7{
```

```
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

Rule: Call to this() must be the first statement in constructor.

4) this: to pass as an argument in the method:

The this keyword can also be passed as an argument in the method. It is mainly used in event handling. Let's see the example:

```
class S2{
 void m(S2 obj){
 System.out.println("method is invoked");
 }
 void p(){
 m(this);
 }
 public static void main(String args[]){
 S2 s1 = new S2();
 s1.p();
 }
}
```