
Lecture – 7 (OOP Concept)

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

| Name | Convention |
|------------|--|
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface | should start with uppercase letter and be an adjective e.g. Runnable, Remote, |

| | |
|----------------|--|
| name | ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc. |

CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

Object and Class in Java

Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, 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 etc. known as its state. It is used to write, so writing is its behavior.

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

Object Definitions:

- Object is a real world entity.
- Object is a run time entity.
- Object is an entity which has state and behavior.
- Object is an instance of a class.

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**

Instance variable in Java

A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

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

3 Ways to initialize object

There are 3 ways to initialize object in java.

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

1) Object and Class Example: Initialization through reference

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

File: TestStudent2.java

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

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
class Student{
    int id;
    String name;
}
class TestStudent3{
    public static void main(String args[]){
        //Creating objects
        Student s1=new Student();
        Student s2=new Student();
        //Initializing objects
        s1.id=101;
        s1.name="Jon";
        s2.id=102;
        s2.name="Kabir";
        //Printing data
        System.out.println(s1.id+" "+s1.name);
        System.out.println(s2.id+" "+s2.name);
    }
}
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
class Student{
    int rollNo;
    String name;
    void insertRecord(int r, String n){
        rollNo=r;
        name=n;
    }
}
```

```
}  
void displayInformation(){System.out.println(rollno+" "+name);}  
}  
class TestStudent4{  
    public static void main(String args[]){  
        Student s1=new Student();  
        Student s2=new Student();  
        s1.insertRecord(111,"Jon");  
        s2.insertRecord(222,"Kabir");  
        s1.displayInformation();  
        s2.displayInformation();  
    }  
}
```

3) Object and Class Example: Initialization through constructor

We will learn about constructors in java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
class Employee{  
    int id;  
    String name;  
    float salary;  
    void insert(int i, String n, float s) {  
        id=i;  
        name=n;  
        salary=s;  
    }  
    void display(){  
        System.out.println(id+" "+name+" "+salary);  
    }  
}  
public class TestEmployee {
```

```
public static void main(String[] args) {  
    Employee e1=new Employee();  
    Employee e2=new Employee();  
    Employee e3=new Employee();  
    e1.insert(101,"Jon",45000);  
    e2.insert(102,"Kabir",25000);  
    e3.insert(103,"Tony",55000);  
    e1.display();  
    e2.display();  
    e3.display();  
}  
}
```

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
class Rectangle{  
    int length;  
    int width;  
    void insert(int l, int w){  
        length=l;  
        width=w;  
    }  
    void calculateArea(){  
        System.out.println(length*width);  
    }  
}  
  
class TestRectangle1{  
    public static void main(String args[]){  
        Rectangle r1=new Rectangle();  
        Rectangle r2=new Rectangle();  
        r1.insert(11,5);  
        r2.insert(3,15);  
        r1.calculateArea();  
    }  
}
```

```
r2.calculateArea();  
}  
}
```

Real World Example: Account

File: TestAccount.java

```
class Account{  
    int acc_no;  
    String name;  
    float amount;  
    void insert(int a,String n,float amt){  
        acc_no=a;  
        name=n;  
        amount=amt;  
    }  
    void deposit(float amt){  
        amount=amount+amt;  
        System.out.println(amt+" deposited");  
    }  
    void withdraw(float amt){  
        if(amount<amt){  
            System.out.println(" Insufficient Balance");  
        }else{  
            amount=amount-amt;  
            System.out.println(amt+" withdrawn");  
        }  
    }  
    void checkBalance(){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(832345,"Sagor",1000);  
        a1.display();  
    }  
}
```

```
a1.checkBalance();  
a1.deposit(40000);  
a1.checkBalance();  
a1.withdraw(15000);  
a1.checkBalance();  
}  
}
```

Constructor in Java

Constructor in java is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

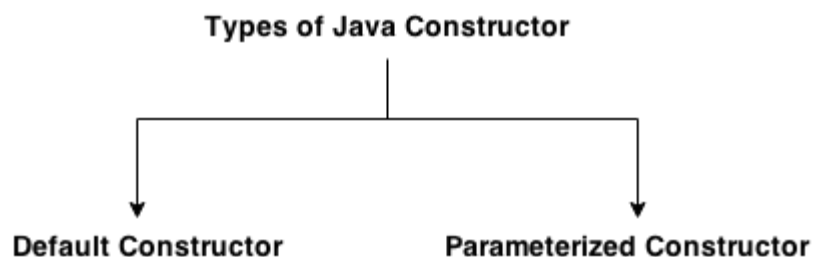
There are basically two rules defined for the constructor.

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

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor that have no parameter is known as default constructor.

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{
    Bike1(){System.out.println("Bike is created");}
    public static void main(String args[]){
        Bike1 b=new Bike1();
    }
}
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Q) What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
class Student3{
    int id;
    String name;
    void display(){
        System.out.println(id+" "+name);
    }
    public static void main(String args[]){
        Student3 s1=new Student3();
        Student3 s2=new Student3();
        s1.display();
        s2.display();
    }
}
```

}

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
    int id;
    String name;
    Student4(int i,String n){
        id = i;
        name = n;
    }
    void display(){
        System.out.println(id+" "+name);
    }
    public static void main(String args[]){
        Student4 s1 = new Student4(111,"Jon");
        Student4 s2 = new Student4(222,"Kabir");
        s1.display();
        s2.display();
    }
}
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
    int id;
    String name;
```

```
int age;
Student5(int i,String n){
    id = i;
    name = n;
}
Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
    Student5 s1 = new Student5(111,"Jon");
    Student5 s2 = new Student5(222,"Kabir",25);
    s1.display();
    s2.display();
}
}
```

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

In this example, we are going to copy the values of one object into another using java constructor.

```
class Student6{
    int id;
    String name;
    Student6(int i,String n){
        id = i;
        name = n;
    }
    Student6(Student6 s){
        id = s.id;
        name =s.name;
    }
}
```

```
}  
void display(){System.out.println(id+" "+name);}  
public static void main(String args[]){  
    Student6 s1 = new Student6(111,"Lincoln");  
    Student6 s2 = new Student6(s1);  
    s1.display();  
    s2.display();  
}  
}
```

Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
class Student{  
    int rollno;
```

```
String name;  
String college="ITS";  
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

Example of static variable

//Program of static variable

```
class Student8{  
    int rollno;  
    String name;  
    static String college ="ITS";  
    Student8(int r,String n){  
        rollno = r;  
        name = n;  
    }  
    void display (){  
        System.out.println(rollno+" "+name+" "+college);  
    }  
    public static void main(String args[]){  
        Student8 s1 = new Student8(111,"Tony");  
        Student8 s2 = new Student8(222,"Asif");  
        s1.display();  
        s2.display();  
    }  
}
```

Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

```
class Counter{
int count=0;//will get memory when instance is created
Counter(){
count++;
System.out.println(count);
}
public static void main(String args[]){
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter2{
static int count=0;//will get memory only once and retain its value
Counter2(){
count++;
System.out.println(count);
}
public static void main(String args[]){
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

Java static method

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

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

Example of static method

//Program of changing the common property of all objects(static field).

```
class Student9{
    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
        college = "BBDIT";
    }
    Student9(int r, String n){
        rollno = r;
        name = n;
    }
    void display (){
        System.out.println(rollno+" "+name+" "+college);
    }
    public static void main(String args[]){
        Student9.change();
        Student9 s1 = new Student9 (111,"Jon");
        Student9 s2 = new Student9 (222,"Kabir");
        Student9 s3 = new Student9 (333,"Raj");
        s1.display();
        s2.display();
        s3.display();
    }
}
```

Restrictions for static method

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

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

```
class A{
    int a=40;//non static
    public static void main(String args[]){
```

```
System.out.println(a);  
}  
}
```

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Example of static block

```
class A2{  
    static  
    {  
        System.out.println("static block is invoked");  
    }  
    public static void main(String args[]){  
        System.out.println("Hello main");  
    }  
}
```

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.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Example:

```
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,"Jon",5000f);
        Student s2=new Student(112,"Sejan",6000f);
        s1.display();
        s2.display();
    }
}
```

2) this: to invoke current class method

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

```
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

The 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.

Calling default constructor from parameterized constructor:

```
class A{
A(){
System.out.println("hello a");
}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}
}
```

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 the 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();
    }
}
```

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{
    A4 obj;
    B(A4 obj){
        this.obj=obj;
    }
    void display(){
        System.out.println(obj.data);//using data member of A4 class
    }
}
class A4{
    int data=10;
    A4(){
        B b=new B(this);
        b.display();
    }
}
```

```
}  
public static void main(String args[]){  
    A4 a=new A4();  
}  
}
```

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
return_type method_name(){  
    return this;  
}
```

Example of this keyword that you return as a statement from the method

```
class A{  
    A getA(){  
        return this;  
    }  
    void msg(){System.out.println("Hello java");}  
}  
class Test1{  
    public static void main(String args[]){  
        new A().getA().msg();  
    }  
}
```