MODULE 2

Java OOPs Concepts

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

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.In java, we use abstract class and interface to achieve abstraction.

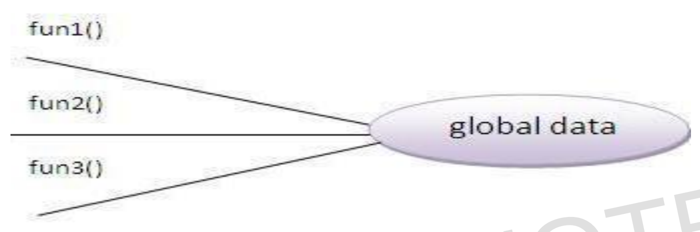
Encapsulation

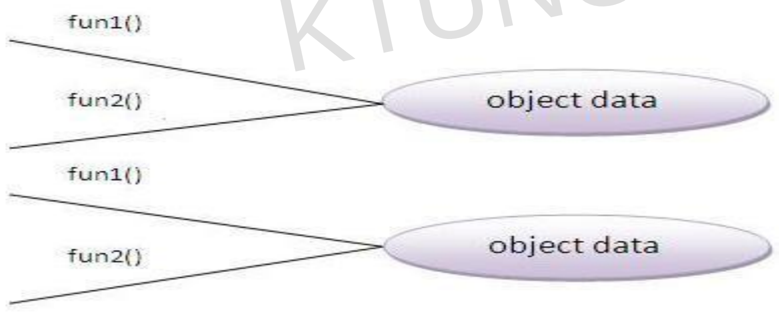
Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

- 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.





Java Naming conventions

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

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 (tengible and intengible). The example of integible 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.

Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member**
- **method**
- **constructor**
- **block**
- **class and interface**

Syntax to declare a class:

```
class <class_name>{
    data member;
    method;
}
```

Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
class Student1{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)
  public static void main(String args[]){
  Student1 s1=new Student1();//creating an object of Student
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

```
Output:0 null
```

Instance variable in Java

A variable that 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 runtime 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 behaviour of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword

The new keyword is used to allocate memory at runtime.

Example of Object and class that maintains the records of students

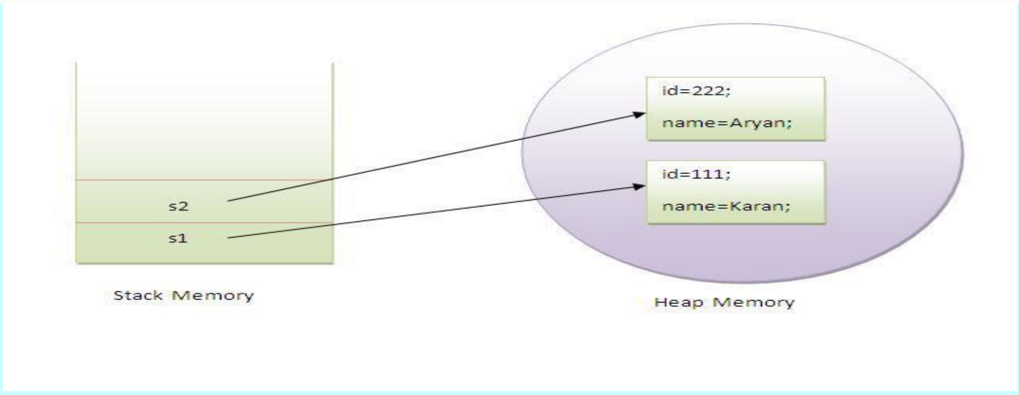
In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
class Student2{
 int rollno;
 String name;
  void insertRecord(int r, String n){  //method
  rollno=r;
  name=n;
 }
  void displayInformation(){System.out.println(rollno+" "+name);}//method
  public static void main(String args[]){
  Student2 s1=new Student2();
  Student2 s2=new Student2();
    s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
    s1.displayInformation();
  s2.displayInformation();

 }
}
```

Test it Now

```
111 Karan
222 Aryan
```



As you see in the above figure, object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

Another Example of Object and Class

There is given another example that maintains the records of Rectangle class. Its exaplanation is same as in the above Student class example.

```
class Rectangle{
 int length;
 int width;
   void insert(int l,int w){
  length=l;
  width=w;
 }
   void calculateArea(){System.out.println(length*width);}
   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();
}
}
```

```
Output:55
       45
```

Different ways to create an object in Java

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

- By new keyword
- By newInstance() method
- By clone() method
- By factory method etc.

We will learn, these ways to create the object later.

Annonymous object

Annonymous simply means nameless.An object that have no reference is known as annonymous object.

If you have to use an object only once, annonymous object is a good approach.

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

```
Output:Factorial is 120
```

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

```
Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
```

Let's see the example:

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

```
Output:55
       45
```

Method Overloading in Java

Different ways to overload the method

- By changing the no. of arguments
- By changing the datatype

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b (int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

java method overloading

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

- There are two ways to overload the method in java
- By changing number of arguments
- By changing the data type
- In java, Method Overloading is not possible by changing the return type of the method.

1) Example of Method Overloading by changing the no. of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
class Calculation{
 void sum(int a,int b){System.out.println(a+b);}
 void sum(int a,int b,int c){System.out.println(a+b+c);}
  public static void main(String args[]){
 Calculation obj=new Calculation();
 obj.sum(10,10,10);
 obj.sum(20,20);
 }
}
```

```
Output:30
    40
```

2) Example of Method Overloading by changing data type of argument

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
class Calculation2{
 void sum(int a,int b){System.out.println(a+b);}
 void sum(double a,double b){System.out.println(a+b);}
  public static void main(String args[]){
 Calculation2 obj=new Calculation2();
 obj.sum(10.5,10.5);
 obj.sum(20,20);
  }
}
```

Output:21.0

```
      40
```

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur: because there was problem:

```
class Calculation3{
 int sum(int a,int b){System.out.println(a+b);}
 double sum(int a,int b){System.out.println(a+b);}
  public static void main(String args[]){
 Calculation3 obj=new Calculation3();
 int result=obj.sum(20,20); //Compile Time Error

 }
}
```

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

<u>Main overloading</u>

```
class Overloading1{
 public static void main(int a){
 System.out.println(a);
 }
    public static void main(String args[]){
 System.out.println("main() method invoked");
 main(10);
 }
}
```

<u>Output</u>:main() method invoked

```
      10
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:

Method overloading with type promotion

As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.

Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
 void sum(int a,long b)
{
System.out.println(a+b);
}
 void sum(int a,int b,int c)
{
System.out.println(a+b+c);
}
  public static void main(String args[]){
 OverloadingCalculation1 obj=new OverloadingCalculation1();
 obj.sum(20,20);//now second int literal will be promoted to long
```

```
  obj.sum(20,20,20);

  }
}
```

Output:40   60

Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class OverloadingCalculation2{
  void sum(int a,int b)
{
System.out.println("int arg method invoked");
}
  void sum(long a,long b)
{
System.out.println("long arg method invoked");
}
    public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```

Output:int arg method invoked

Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```
class OverloadingCalculation3{
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}
    public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```

Output:Compile Time Error

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

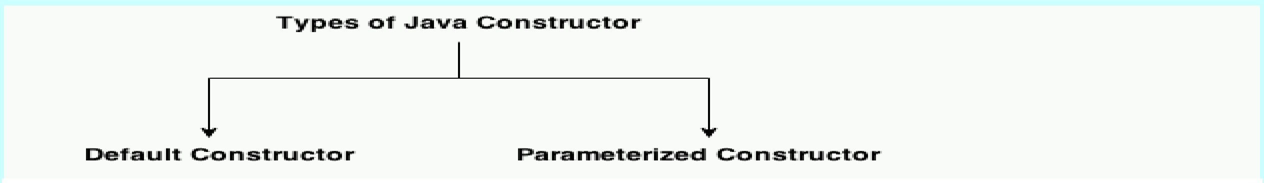
There are basically two rules defined for the constructor.

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.

Syntax of default constructor:

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

Example of default constructor

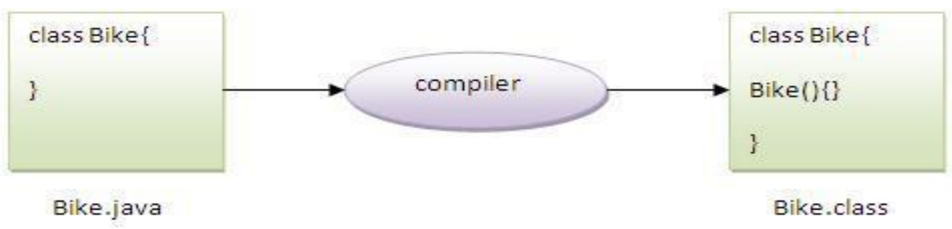
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

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

Output:

```
Bike is created
```

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



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();
}
}
```

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Use parameterized constructor

Parameterized constructor is used to provide different values to the distinct objects.

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,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 222 Aryan
```

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,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 0 222 Aryan 25
```

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++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

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,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}
    void display(){System.out.println(id+" "+name);}
      public static void main(String args[]){
    Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
```

```
  s2.id=s1.id;
  s2.name=s1.name;
  s1.display();
  s2.display();
 }
}
```

Output:

```
111 Karan
111 Karan
```

Access Modifiers in java

There are two types of modifiers in java: **access modifiers and non-access modifiers**.

The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

- **Private**
- **Default**
- **Protected**
- **Public**

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

1) **private access modifier**

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{

private int data=40;

private void msg(){System.out.println("Hello java");}

}

  public class Simple{

 public static void main(String args[]){

   A obj=new A();

   System.out.println(obj.data);//Compile Time Error

   obj.msg();//Compile Time Error

   }

}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{

private A(){}//private constructor

void msg(){System.out.println("Hello java");}

}

public class Simple{

 public static void main(String args[]){

   A obj=new A();//Compile Time Error

 }

}
```

2) **default access modifier**

If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
package pack;

class A{

  void msg(){System.out.println("Hello");}

}

package mypack;

import pack.*;

class B{

  public static void main(String args[]){

   A obj = new A();//Compile Time Error

   obj.msg();//Compile Time Error

  }

}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3**) protected access modifier**

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
package pack;

public class A{

protected void msg(){System.out.println("Hello");}

}

package mypack;

import pack.*;

class B extends A{

  public static void main(String args[]){

   B obj = new B();

   obj.msg();

  }

}

Output:Hello
```

4**) public access modifier**

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

package mypack;

import pack.*;

class B{

  public static void main(String args[]){
   A obj = new A();

   obj.msg();

  }

}
```

Output:Hello

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Java access modifiers with method overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class A{

protected void msg(){System.out.println("Hello java");}

}

  public class Simple extends A{

void msg(){System.out.println("Hello java");}//C.T.Error

 public static void main(String args[]){

   Simple obj=new Simple();

   obj.msg();

   }

}
```

The default modifier is more restrictive than protected. That is why there is compile time error.