LECTURE NOTES

UNIT I

Unit 1: Introduction

INTRODUCTION
Classes, Fields, Access Control, Creating Objects
Construction, Initialization, Methods, this, Overloading Methods, main Method, Native Methods, Class Design.
Statements & Blocks, if-else, switch, while and do-while, for, Labels, break, continue, return, goto

2.1. JAVA OOPS CONCEPTS

In this page, we will learn about basics of OOPs. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**,**polymorphism** etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

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 convense 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.

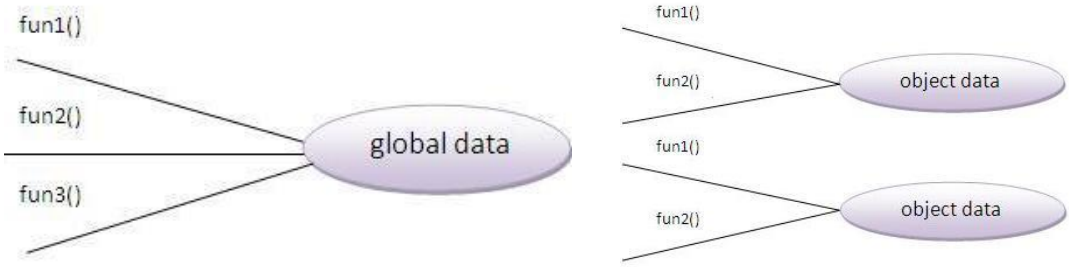
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

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 prgramming 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.



What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

2.2. JAVA NAMING CONVENTIONS

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

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

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 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 etc.

UNDERSTANDING 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.

2.3. METHOD OVERLOADING IN JAVA

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 behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

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

1. By changing number of arguments
2. By changing the data type

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.

```
1. class Calculation{
2.   void sum(int a,int b){System.out.println(a+b);}
3.   void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   obj.sum(10,10,10);
8.   obj.sum(20,20);
9.
10.  }
11. }
```

```
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.

```
1. class Calculation{
2.   void sum(int a,int b){System.out.println(a+b);}
3.   void sum(double a,double b){System.out.println(a+b);}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   obj.sum(10.5,10.5);
8.   obj.sum(20,20);
9.
10.  }
11. }
```

```
Output:21.0
       40
```

OVERLOAING IS NOT POSSIBLE BY CHANGING THE RETURN TYPE OF METHOD

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:

```
1. class Calculation{
2.   int sum(int a,int b){System.out.println(a+b);}
3.   double sum(int a,int b){System.out.println(a+b);}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   int result=obj.sum(20,20); //Compile Time Error
8.
9.   }
10. }
```

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

Can we overload main() method?

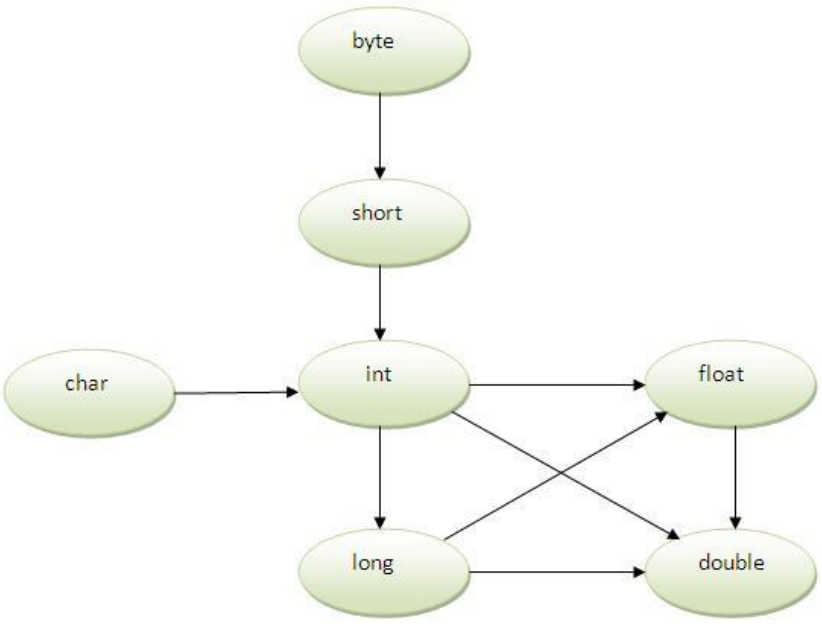
Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
1. class Simple{
2.   public static void main(int a){
3.   System.out.println(a);
4.   }
5.
6.   public static void main(String args[]){
7.   System.out.println("main() method invoked");
8.   main(10);
9.   }
10. }
```

```
Output:main() method invoked
       10
```

METHOD OVERLOADING AND TYPEPROMOTION

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



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

```
1. class Calculation{
2.   void sum(int a,long b){System.out.println(a+b);}
3.   void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   obj.sum(20,20);//now second int literal will be promoted to long
8.   obj.sum(20,20,20);
9.
10.  }
11. }
```

```
Output:40
       60
```

EXAMPLE OF METHOD OVERLOADING WITH TYPEPROMOTION IF MATCHING FOUND

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

```
1. class Calculation{
2.   void sum(int a,int b){System.out.println("int arg method invoked");}
3.   void sum(long a,long b){System.out.println("long arg method invoked");}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   obj.sum(20,20);//now int arg sum() method gets invoked
8.   }
9. }
```

```
Output:int arg method invoked
```

EXAMPLE OF METHOD OVERLOADING WITH TYPEPROMOTION IN CASE AMBIGUITY

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

```
1. class Calculation{
2.   void sum(int a,long b){System.out.println("a method invoked");}
3.   void sum(long a,int b){System.out.println("b method invoked");}
4.
5.   public static void main(String args[]){
6.   Calculation obj=new Calculation();
7.   obj.sum(20,20);//now ambiguity
8.   }
9. }
```

```
Output:Compile Time Error
```

2.4.CONSTRUCTOR IN JAVA

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

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 constructor

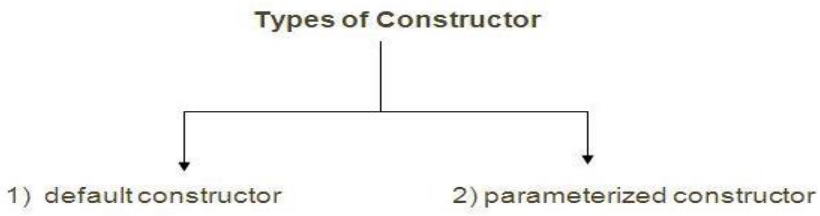
There are basically two rules defined for the constructor.

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

Types of constructors

There are two types of constructors:

1. default constructor (no-arg constructor)
2. parameterized constructor



2.4.1. DEFAULT CONSTRUCTOR

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

Syntax of default constructor:

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

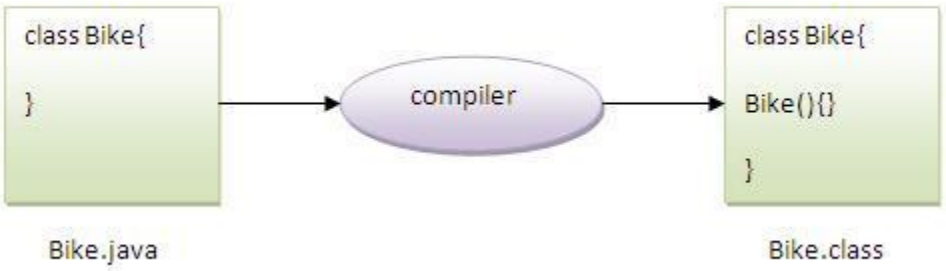
EXAMPLE OF DEFAULT CONSTRUCTOR

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

```
1. class Bike{
2.
3. Bike(){System.out.println("Bike is created");}
4.
5. public static void main(String args[]){
6. Bike b=new Bike();
7. }
8. }
```

```
Output: Bike is created
```

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



Que)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

```
1. class Student{
2. int id;
3. String name;
4.
5. void display(){System.out.println(id+" "+name);}
6.
7. public static void main(String args[]){
8. Student s1=new Student();
```

```
9.  Student s2=new Student();
10. s1.display();
11. s2.display();
12. }
13. }
```

```
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.

2.4.2. PARAMETERIZED CONSTRUCTOR

A constructor that have parameters is known as parameterized constructor.

Why 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.

```
1.  class Student{
2.     int id;
3.     String name;
4.
5.     Student(int i,String n){
6.     id = i;
7.     name = n;
8.     }
9.     void display(){System.out.println(id+" "+name);}
10.
11.    public static void main(String args[]){
12.    Student s1 = new Student(111,"Karan");
13.    Student s2 = new Student(222,"Aryan");
14.    s1.display();
15.    s2.display();
16.   }
17. }
```

```
Output:111 Karan
       222 Aryan
```

2.5. CONSTRUCTOR OVERLOADING

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

```
1.  class Student{
2.     int id;
3.     String name;
4.     int age;
5.     Student(int i,String n){
6.     id = i;
7.     name = n;
8.     }
9.     Student(int i,String n,int a){
10.    id = i;
11.    name = n;
12.    age=a;
13.    }
14.    void display(){System.out.println(id+" "+name+" "+age);}
15.
16.    public static void main(String args[]){
17.    Student s1 = new Student(111,"Karan");
18.    Student s2 = new Student(222,"Aryan",25);
19.    s1.display();
20.    s2.display();
21.   }
22. }
```

```
Output:111 Karan 0
       222 Aryan 25
```

What is the difference between constructor and method ?

There are many differences between constructors and methods. They are given below.

Constructor	Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.

Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

COPYING 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. 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 constructor.

```
1.  class Student{
2.     int id;
3.     String name;
4.     Student(int i,String n){
5.     id = i;
6.     name = n;
7.     }
8.
9.     Student(Student s){
10.    id = s.id;
11.    name =s.name;
12.    }
13.    void display(){System.out.println(id+" "+name);}
14.
15.    public static void main(String args[]){
16.    Student s1 = new Student(111,"Karan");
17.    Student s2 = new Student(s1);
18.    s1.display();
```

```
19.    s2.display();
20.   }
21. }
```

```
Output:111 Karan
       111 Karan
```

Copying the values of one object to another 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.

```
1.  class Student{
2.     int id;
3.     String name;
4.     Student(int i,String n){
5.     id = i;
6.     name = n;
7.     }
8.     Student(){}
9.     void display(){System.out.println(id+" "+name);}
10.
11.    public static void main(String args[]){
12.    Student s1 = new Student(111,"Karan");
13.    Student s2 = new Student();
14.    s2.id=s1.id;
15.    s2.name=s1.name;
16.    s1.display();
17.    s2.display();
18.   }
19. }
```

```
Output:111 Karan
       111 Karan
```

2.6 ACCESS MODIFIERS IN JAVA

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

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

There are 4 types of java access modifiers:

1. private
2. default

3. protected
4. 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.

```
1. class A{
2. private int data=40;
3. private void msg(){System.out.println("Hello java");}
4. }
5.
6. public class Simple{
7.  public static void main(String args[]){
8.    A obj=new A();
9.    System.out.println(obj.data);//Compile Time Error
10.   obj.msg();//Compile Time Error
11.   }
12. }
```

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:

```
1. class A{
2. private A(){}//private constructor
3. void msg(){System.out.println("Hello java");}
4. }
```

```
5. public class Simple{
6.  public static void main(String args[]){
7.    A obj=new A();//Compile Time Error
8.  }
9. }
```

Note: A class cannot be private or protected except nested class.

---

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.

```
1. //save by A.java
2. package pack;
3. class A{
4.   void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4. class B{
5.   public static void main(String args[]){
6.    A obj = new A();//Compile Time Error
7.    obj.msg();//Compile Time Error
8.   }
9. }
```

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.

```
1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4.
5. class B extends A{
6.   public static void main(String args[]){
7.    B obj = new B();
8.    obj.msg();
9.   }
10. }
Output:Hello
```

4) PUBLIC ACCESS MODIFIER

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

Example of public access modifier

```
1. //save by A.java
2.
```

```
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2.
3. package mypack;
4. import pack.*;
5.
6. class B{
7.   public static void main(String args[]){
8.    A obj = new A();
9.    obj.msg();
10.  }
11. }
```

```
Output:Hello
```

UNDERSTANDING ALL JAVA ACCESS MODIFIERS

Let's understand the access modifiers by a simple table.

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