Unit 3

Object Oriented Programming Concepts

Fundamentals of Classes:

- 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**

```java
public class Employee {

    //Fields, data members, Instance variables
    private int employeeId;
    private String employeeName;
    private String employeeAddress;

    //Constructors
    public Employee(int employeeId, String employeeName, String employeeAddress) {
        this.employeeId = employeeId;
        this.employeeName = employeeName;
        this.employeeAddress = employeeAddress;
    }
    //Methods
    public String showEmployee() {
        return "Id : "+employeeId + "\nName : "+employeeName + "\nAddress : "+employeeAddress;
    }


    public static void main(String[] args) {
        //Creating a class instance
        Employee employee = new Employee(10, "Hari Dahal", "Baneshwor 31, Kathmandu");

        //Calling methods
        String employeeDetails = employee.showEmployee();
        System.out.println(employeeDetails);
    }
}
```

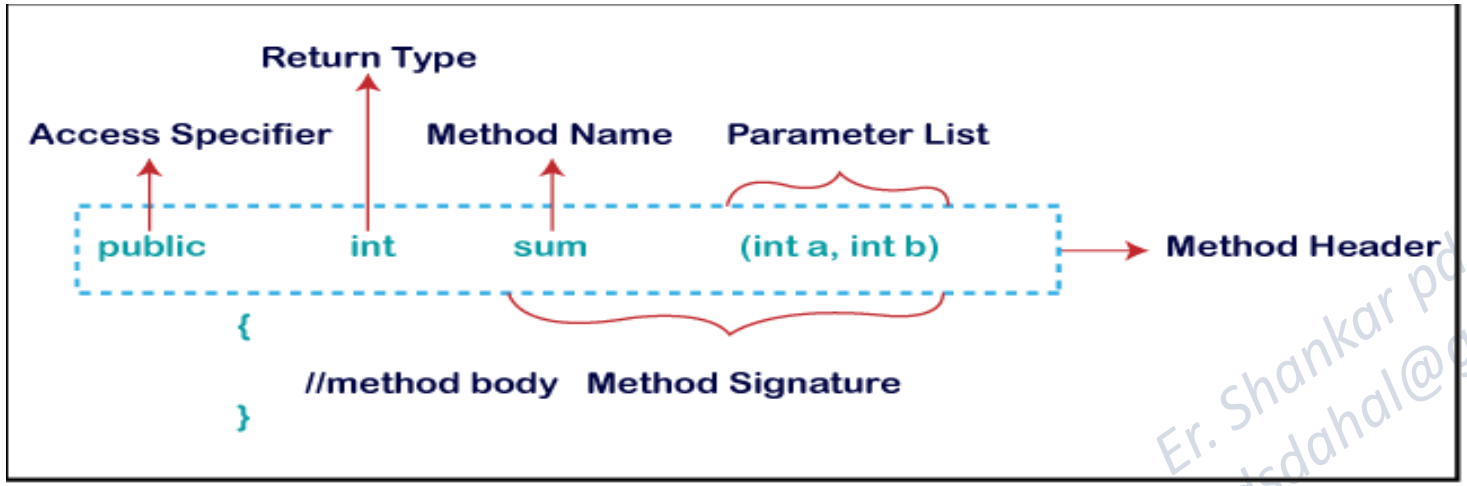
Fig. General form of class

Method 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.

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.



1. **Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.
2. **Return Type:** It is a data type that the method returns. If the method does not return anything, we use void keyword.
3. **Method Name:** It is a unique name that is used to define the name of a method.
4. **Method Body:** It contains all the actions to be performed. It is enclosed within the pair of curly braces.
5. **Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method.

Pass by Value and Pass by Reference in Java :

❖ Pass by Value and Pass by reference is the two ways by which we can pass a value to the variable in a function.

1. Pass by Value:

➢ When we pass only the value part of a variable to a function as arguments, it is referred to as pass by value.

➢ In below figure only the value part of the variable is passed i.e. a copy of the existing variable is passed instead of passing the origin address of the variable. Hence, any changes done to the value of the copy will not have any impact on the value of the original copy. **Java supports pass-by-value.**

➢ For primitive types, Java makes a copy of the variables and sends it to the function.

2. Pass by Reference:

- When the reference to the location of a variable is passed to a function as arguments, then it is called pass by reference.
- Java does not support pass-by-reference.

Handles Pass by Reference in Java:

- Instead of passing only the value part of a variable, Java passes a reference to the original object for **non-primitive**.

- Any modification made to the referenced object will reflect changes in the original object.

Java OOPs

- As the name suggests, OOPs refers to languages that uses objects in programming.
- Object-oriented programming is a model that provides different types of concepts, such as inheritance, abstraction, polymorphism, etc. These concepts aim to implement real-world entities in programs.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- **Procedural programming** is about writing procedures or methods that perform operations on the data, while **object-oriented programming** is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

1. OOP is faster and easier to execute.
2. OOP provides a clear structure for the programs.
3. OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.
4. OOP makes it possible to create full reusable applications with less code and shorter development time.

Note : The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

Object :

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Create Object in Java :

1. Using new keyword:

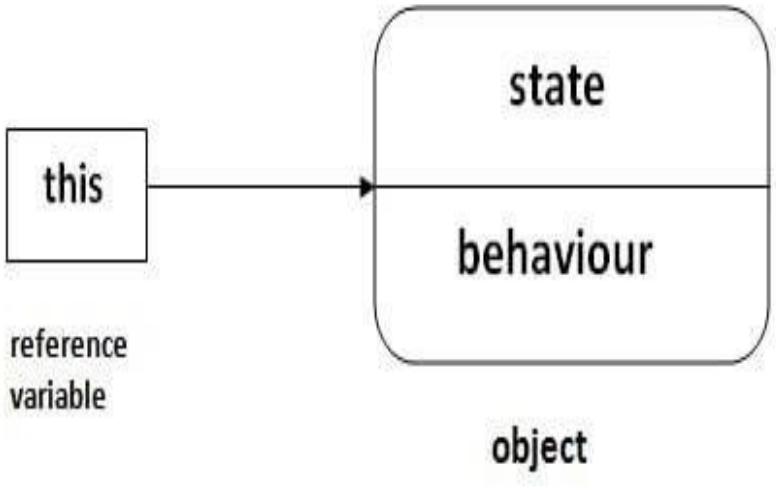
- Using the **new** keyword is the most popular way to create an object or instance of the class.
- When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created **object** and also returns the **reference** of that object to that memory.

- The syntax for creating an object is:

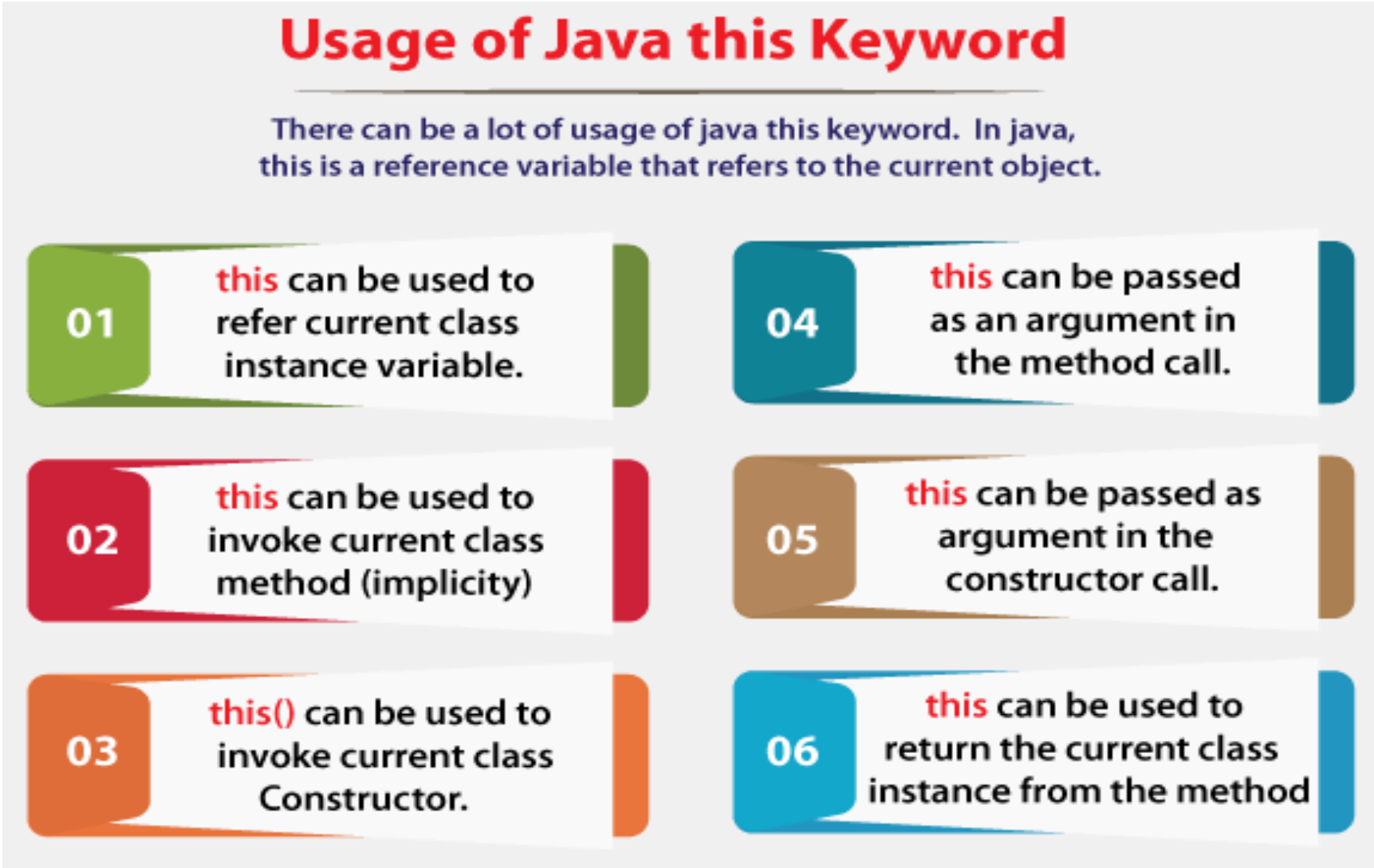
```
ClassName object = new ClassName();
```

this keyword:

- In Java, **this** is a **reference variable** that refers to the current object.
- The most common use of **this keyword** is to eliminate the confusion between class attributes and parameters with the same name



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



Constructors in Java:

- A constructor in Java is a **special method** that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes.
- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.

Note: *It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.*

Characteristics of Constructor:

1. Constructor name must be the same as its class name.
2. The constructor should not have any return type even void also because if there is a return type then JVM would consider as a method, not a constructor.
3. Whenever we create an object/instance of a class, the constructor will be automatically called by the JVM (Java Virtual Machine).

If we don't define any constructor inside the class, Java compiler automatically creates a default constructor at compile-time and assigns default values for all variables declared in the class.

The default values for variables are as follows:

a. Numeric variables are set to 0.
b. Strings are set to null.
c. Boolean variables are set to false.

4. Java constructor may or may not contain parameters. Parameters are local variables to receive value (data) from outside into a constructor.

5. A constructor is automatically called and executed by JVM at the time of object creation. JVM first allocates the memory for variables (objects) and then executes the constructor to initialize instance variables.

6. It is called and executed only once per object. This means that when an object of a class is created, constructor is called. When we create second object then the constructor is again called during the second time.

Types of Constructors in Java:

In Java, constructors can be divided into 3 types:

1. **No-Arg Constructor :**

❖ As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

2. Parameterized Constructor:

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

3. Default Constructor:

❖ If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

Characteristics/Basic building blocks/Principles of Object-Oriented programming language in Java

1. Encapsulation :

- Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.

2. Abstraction :

- Data abstraction is the process of hiding certain details and showing only essential information to the user.

3. Polymorphism :

- It is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms.

4. Inheritance :

- It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.

1. Encapsulation

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

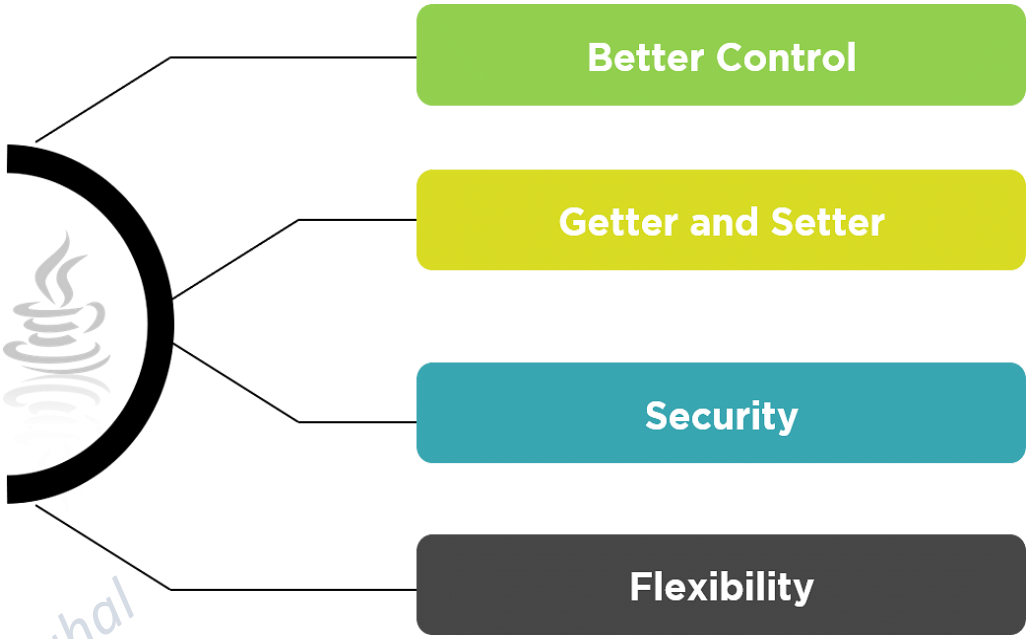
To achieve encapsulation in Java :

i. Declare the variables of a class as private.
ii. Provide public setter and getter methods to modify and view the variables values.

Let's understand the access modifiers in Java 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

Need for Encapsulation in Java :



1. Better Control:
Encapsulation provides ultimate control over the data members and data methods inside the class.

2. Getter and Setter:
The standard IDEs provide in-built support for 'Getter and Setter' methods, which increases the programming pace.

3. Security:
Encapsulation prevents access to data members and data methods by any external classes. The encapsulation process improves the security of the encapsulated data.

4. Flexibility:
Changes made to one part of the code can be successfully implemented without affecting any other part of the code.

Getter and Setter Methods

1. Getter
The method capable of accessing and retrieving an instance of a private variable is known as a getter method.

2. Setter
The method that is capable of modifying or setting to an instance of a private variable is the setter method.

2. Abstraction :

- Abstraction is the process of selecting data to show only the relevant information to the user.
- In simple terms, abstraction "**displays**" only the relevant attributes of objects and "**hides**" the unnecessary details.
- In Java, abstraction is implemented using an abstract class and interface.
- We can achieve 100% abstraction using interfaces.
- Java provides a non-access modifier "**abstract**" for implementing abstraction. This abstract modifier can be used with classes and methods but not variables.

Abstract Class:

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
- An abstract class can have abstract methods (methods without body) as well as non-abstract methods or concrete methods (methods with the body). A non-abstract class cannot have abstract methods.
- if a class has at least one abstract method, then the class **must** be declared abstract.
- Abstract classes cannot be instantiated, but they can be subclassed.
- Abstract class can not be declared as a final class.
- Abstract class can inherit another class using extends keyword and implement an interface.
- There is always a default constructor in an abstract class, it can also have a parameterized constructor.
- The abstract class can also contain final and static methods.

Syntax:

```
abstract class ClassName
{
//class body
}
```

Abstract Methods:

- Abstract methods are methods with no implementation and without a method body. They do not contain any method statement.
- An abstract method is declared with an abstract keyword.
- The declaration of an abstract method must end with a semicolon **;**
- The child classes which inherit the abstract class must provide the implementation of these inherited abstract methods.

Syntax:
access-specifier abstract **return**-type method-name();

Differences:

Abstraction	Encapsulation
Abstraction in Object Oriented Programming solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction in Programming is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Data Abstraction in Java allows focusing on what the information object must contain.	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

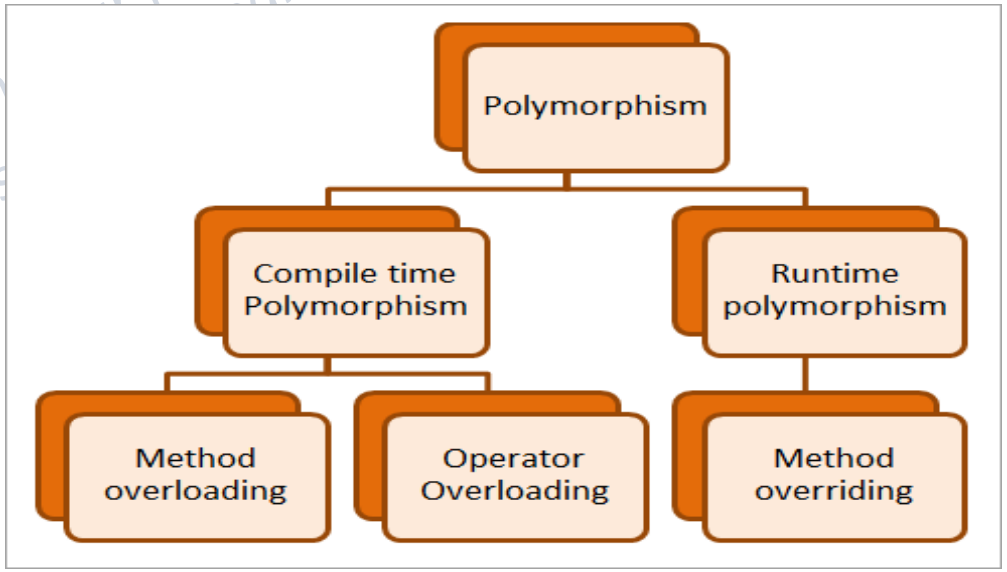
3. Polymorphism :

- The word polymorphism means having many forms.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- In Java, polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method.

1. Compile-time polymorphism :

- Compile-time polymorphism is also known as "**Static polymorphism**".
- As the name suggests, the compile-time polymorphism is performed at compile-time.
- This type of polymorphism is achieved by **function overloading** or **operator overloading**.

- ***Note: Java doesn't support the Operator Overloading.***



A. Method Overloading:

- When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**.

- There are two ways to overload the method in java:

i. By changing number of arguments
ii. By changing the data type

Note: In Java, Method Overloading is not possible by changing the return type of the method only.

2. Runtime polymorphism:

- **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- In this process, an overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the object being referred to by the reference variable.
- Runtime polymorphism in Java is achieved by using "**method overriding**".

Upcasting:

- If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

Rules of Runtime Polymorphism:

- Methods of child and parent class must have the same name.
- Methods of child and parent class must have the same parameter.
- IS-A relationship is mandatory (inheritance).

Limitations of Runtime Polymorphism:

- One cannot override the private methods of a parent class.
- One cannot override Final methods.
- One cannot override static methods.

A. Method Overriding:

- Method overriding is a technique by which a method in the parent class is redefined or overridden in the child class.
- When the method is overridden in a class, the dynamic method dispatch technique resolves the overridden method call at runtime and not at compile time.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Compile Time Polymorphism	Run time Polymorphism
In Compile time Polymorphism, the call is resolved by the compiler.	In Run time Polymorphism, the call is not resolved by the compiler.
It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Method overriding is the runtime polymorphism having the same method with same parameters or signature but associated withcompared, different classes.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution because the method that needs to be executed is known early at the compile time.	It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime.
Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.
Inheritance is not involved.	Inheritance is involved.

Java Recursion :

- In Java, a method that calls itself is known as a recursive method. And this process is known as recursion.
- This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Nested Classes :

- The Java programming language allows you to define a class within another class. Such a class is called a *nested class*.
- For e.g.

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```


There are two types of nested classes :

1. Non-static nested class (**Inner class**)
2. Static nested class

1. Non-static nested class (Inner class) :

- A non-static nested class is a class within another class.
- It has access to members of the enclosing class (outer class). It is commonly known as inner class.
- The **inner class** is associated with the **object of the outer class**. So, the inner class is treated like other variables and methods of the outer class.
- The **inner class** is associated with the outer class object or instance, so we **can't declare static variables** inside the inner class.
- Since the inner class exists within the outer class, you must instantiate the outer class first, in order to instantiate the inner class.

2. Static Nested Class :

- In Java, we can also define a static class inside another class. Such class is known as static nested class.
- Unlike inner class, a static nested class cannot access the member variables of the outer class. It is because the **static nested class** doesn't require you to create an instance of the outer class.