

# JAVA



What you learn ?

## Java

- |                    |                  |
|--------------------|------------------|
| ➤ Paradigm         | ➤ OOP Concepts   |
| ➤ Types of Classes | ➤ Constructor    |
| ➤ this keyword     | ➤ static keyword |

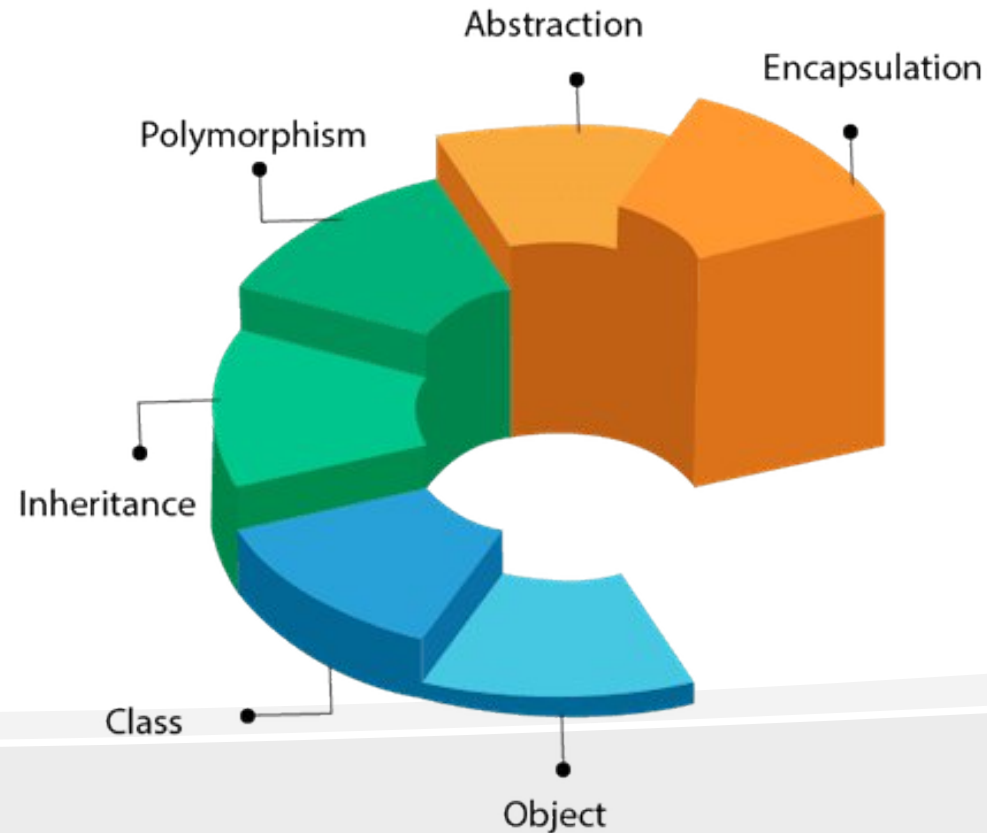
- 
- **In programming, a paradigm refers to a particular style or approach to solving problems and designing software applications. It is a fundamental way of thinking about and organizing code.**

- **Procedural programming:** This paradigm involves writing code that follows a series of steps, with an emphasis on the functions or procedures that carry out those steps.
- **Object-oriented programming:** This paradigm focuses on creating objects that encapsulate data and behavior. Objects can interact with each other through methods, which are functions that belong to a specific object.
- **Functional programming:** This paradigm emphasizes the use of functions that do not have side effects and do not modify data. Instead, functions take inputs and produce outputs based on those inputs.
- **Event-driven programming:** This paradigm involves programming software that responds to user events, such as mouse clicks or keyboard input.

# OOP concept

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

OOPs (Object-Oriented Programming System)



## ➤ What Is an Object?

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner.

## ➤ What Is a Class?

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior.

## ➤ What Is Inheritance?

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

## ➤ What is Abstraction

Abstraction refers to the process of hiding implementation details of a class or method, and only exposing relevant information to the users of that class or method. This allows for a simpler and more organized design, and helps prevent users from making unintended changes to the internal workings of a class or method.

Abstraction can be achieved through two main mechanisms in Java: abstract classes and interfaces.



## What Is an Interface?

An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it.

## ➤ What Is a Package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.

## ➤ What is Polymorphism ?

- Polymorphism refers to the ability of an object to take on many forms or types. Polymorphism allows objects of different classes to be treated as if they are of the same class, as long as they implement the same methods.
- There are two main types of polymorphism in Java:
  - **Compile-time polymorphism:** Also known as method overloading, this type of polymorphism occurs when a class has multiple methods with the same name but different parameters. The compiler determines which method to call based on the number and types of arguments passed to it.
  - **Runtime polymorphism:** Also known as method overriding, this type of polymorphism occurs when a subclass overrides a method of its superclass with its own implementation. The overridden method must have the same name, return type, and parameter list as the original method.

# Example of object and class



## Object and Class Example: main within the class

```
//Defining a Student class.  
class Student{  
    //defining fields  
    int id;//field or data member or instance variable  
    String name;  
    //creating main method inside the Student class  
    public static void main(String args[]){  
        //Creating an object or instance  
        Student s1=new Student();//creating an object of  
Student  
        System.out.println(s1.id);//accessing member thro  
ugh reference variable  
        System.out.println(s1.name);  
    }  
}
```

## Object and Class Example: main outside the class

```
//Java Program to demonstrate having the main m  
ethod in  
//another class  
//Creating Student class.  
class Student{  
    int id;  
    String name;  
}  
//Creating another class TestStudent1 which conta  
ins the main method  
class TestStudent1{  
    public static void main(String args[]){  
        Student s1=new Student();  
        System.out.println(s1.id);  
        System.out.println(s1.name);  
    }  
}
```




# Types of Classes



- **Concrete classes:** These are standard classes that can be instantiated to create objects. They may have instance variables, methods, and constructors, and can be extended by other classes.
- **Abstract classes:** These are classes that cannot be instantiated directly, but are intended to be extended by other classes. They often contain abstract methods, which must be implemented by any subclass.
- **Final classes:** These are classes that cannot be extended by other classes. They are often used to ensure that a class's implementation cannot be modified or overridden.

# Types of Classes

- 
- **Inner classes:** These are classes that are defined within another class, and can access its private members. They can be static or non-static, and can have their own members and methods.
  - **Anonymous classes:** These are a type of inner class that are defined and instantiated at the same time. They are often used to provide a one-time implementation of an interface or abstract class.
  - **Static classes:** These are a type of nested class that have the static modifier. They can be accessed without creating an instance of their containing class.
  - **Local classes:** These are classes that are defined within a method or block of code. They can access the variables of the enclosing method, but are not accessible outside of it.

# Constructor



- A constructor is a special method that is used to initialize an object. Every class has a constructor either implicitly or explicitly.
- If we don't declare a constructor in the class then JVM builds a default constructor for that class. This is known as default constructor.
- A constructor has same name as the class name in which it is declared. Constructor must have no explicit return type. Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.
- A constructor is called automatically when an object of the class is created using the new keyword. The main purpose of a constructor is to ensure that an object of the class is properly initialized before it is used.

# Types of constructor in java



- Default constructor:** A default constructor is a constructor that takes no arguments. If a class does not have any constructors defined, a default constructor is automatically created by the Java compiler. The default constructor initializes all instance variables to their default values (e.g., null for object references, 0 for numeric types).
- **If a class contain a constructor with no parameter then it is known as default constructor defined by user. In this case JVM does not create default constructor.**
  - **Parameterized constructor:** A parameterized constructor is a constructor that takes one or more arguments. It allows objects of a class to be created with specific initial values for their instance variables. A class can have multiple parameterized constructors, each with a different combination of parameters.

# Example of constructor



## No-arg constructor-

```
public class Main {  
    int x; // Create a class attribute  
    // Create a class constructor for the Main class  
    public Main() {  
        x = 5; // Set the initial value for the class attribute  
    }  
    public static void main(String[] args) {  
        Main myObj = new Main(); // Create an object of  
        class Main (This will call the constructor)  
        System.out.println(myObj.x); // Print the value of  
    }  
}  
// Outputs 5
```

## Parameterized constructor-

```
public class Main {  
    int x;  
  
    public Main(int y) {  
        x = y;  
    }  
  
    public static void main(String[] args) {  
        Main myObj = new Main(5);  
        System.out.println(myObj.x);  
    }  
}  
  
// Outputs 5
```

# This keyword

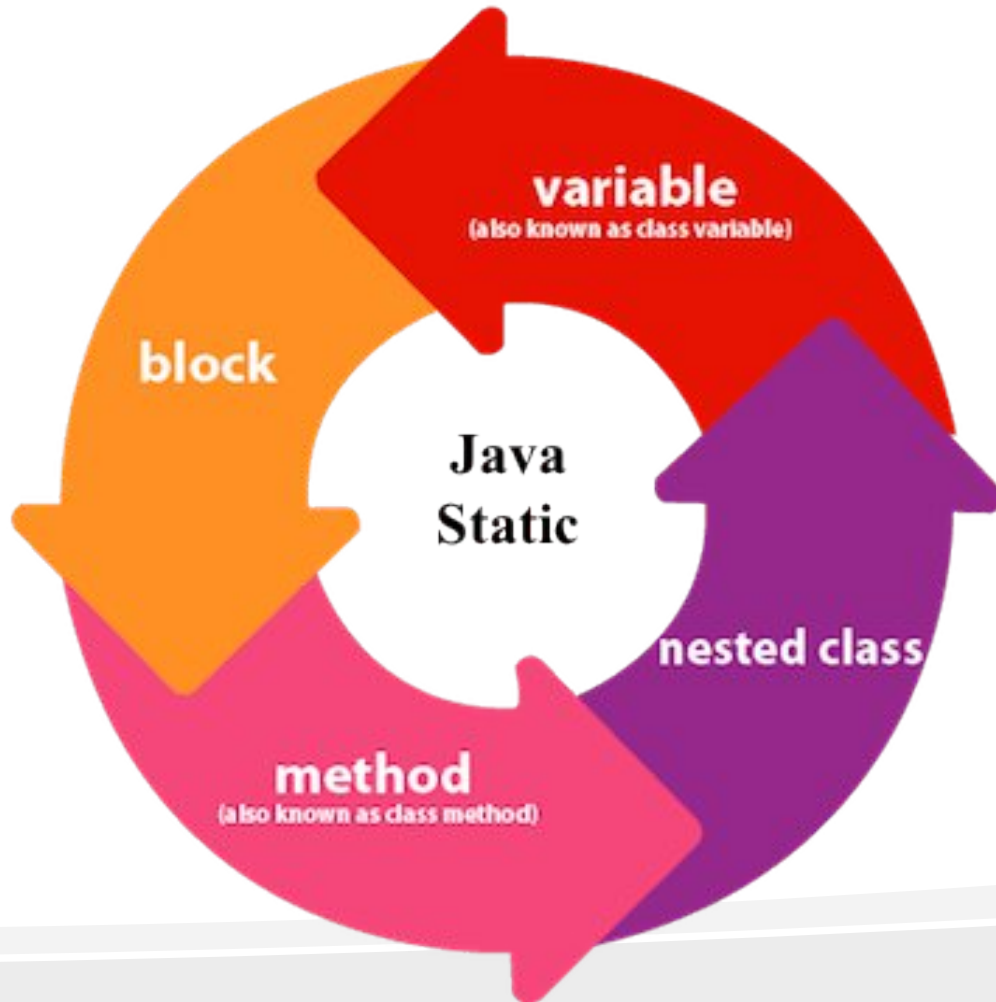
**In Java, this is a keyword which is used to refer current object of a class.**

- **To refer to an instance variable of the current object:** When this is used followed by a dot and an instance variable name, it refers to the instance variable of the current object.
- **To call a constructor from another constructor:** When a constructor needs to call another constructor of the same class, this() can be used. It must be the first statement in the constructor body
- **To pass the current object as an argument to another method:** When this is passed as an argument to a method, it passes a reference to the current object
- **To call instance method from another instance method**

# Static keyword



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



# Static keyword



- the static keyword is used to create class-level variables and methods that can be accessed without creating an object of the class. Here are some key features of static variables and methods:
- Static variables: A static variable is a class-level variable that is shared by all instances of the class. It can be accessed without creating an object of the class.
- Static methods: A static method is a class-level method that can be called without creating an object of the class.



# Static keyword



- Here are some common use cases for static variables and methods:
- To keep track of information that is shared by all instances of a class (such as a count of the number of objects created).
- To create utility methods that do not require an object of the class to be created (such as math functions or string formatting methods).
- To create constants that are shared by all instances of a class (such as mathematical or physical constants).
- It's important to note that static variables and methods belong to the class itself, not to any specific instance of the class. This means that they can be accessed and modified by any method or object of the class. However, because they are shared by all instances of the class, changes made to static variables or methods will be visible to all objects of the class.

# Object class in JAVA



The **Object** class is the parent class of all other classes. It is at the top of the class hierarchy in Java. Every class in Java implicitly or explicitly extends the Object class, and therefore, inherits its methods.

The Object class has a few important methods that are inherited by all other classes. Some of these methods include:

- toString():** This method returns a string representation of the object. It is often overridden in child classes to provide a meaningful string representation of the object.
- equals(Object obj):** This method checks if the current object is equal to another object. It returns true if the two objects are equal, and false otherwise.
- hashCode():** This method returns a hash code value for the object. It is used in hash tables, which are data structures that allow for quick lookup of objects.
- finalize() :** method can be overridden in a class to perform any necessary cleanup before the object is garbage collected.

# equals() method in JAVA



The equals() method is defined in the Object class in Java. By default, it uses the == operator for comparison. However, equals() method can be overridden to provide custom logic to compare two objects.

When we want to compare two objects based on some logic, we need to override the equals() method in the corresponding class of those objects.

Thus equals() methods compare two entities and return true if they are logically the same. Since equals() is a method defined in the Object class thus the default implementation of the equals() method compares the object references or the memory location where the objects are stored in the heap.

Thus by default the equals() method checks the object by using the “==” operator.



# Thanks

**Anirudha Gaikwad**