Object Oriented Paradigm :

The Object-Oriented Paradigm is a programming paradigm that focuses on organizing and designing software using objects, classes, and the principles of Object-Oriented Programming (OOP).

It's a way of thinking about software development that emphasizes modeling real-world entities and their interactions. The Object-Oriented Paradigm is a fundamental shift from traditional procedural programming and has become widely adopted due to its benefits in terms of code organization, reusability, and maintainability.

OOP promotes modular, organized, and maintainable code by encouraging the encapsulation of data and behaviors into well-defined classes and by enabling code reuse through inheritance and polymorphism. **Java, with its classes, objects, access modifiers, and other features, provides a strong platform for practicing OOP principles.**

1.class : A class is a blueprint or template for creating objects. It defines the structure and behavior that the objects of the class will have. It contains data members (variables-fields) and methods (functions) that define the characteristics and actions of the objects.

class ClassName {

// Fields or attributes (variables)

// These define the properties of objects created from this class

dataType fieldName1;

dataType fieldName2;

// ...

// Constructors

// These are special methods used to initialize objects

```
    public ClassName() {
        // Constructor code
    }
    // Methods
    // These define the behaviors of objects created from this class
    returnType methodName1(parameterType parameterName1, parameterType parameterName2, ...) {
        // Method code
    }
    returnType methodName2(parameterType parameterName1, parameterType parameterName2, ...) {
        // Method code
    }
    // ...}
```

components of a class definition:

Class Declaration: The public class ClassName line declares the class. The class name should start with an uppercase letter and follow Java naming conventions.

Fields (Attributes): These are variables that represent the properties of the objects created from the class. Fields define the state of the objects.

Constructors: Constructors are special methods with the same name as the class. They are used to initialize objects when they are created. Constructors do not have a return type, not even void.

Methods: These are functions that define the behavior of objects created from the class. Methods perform specific actions or computations.

2.Object : objects are instances of classes. Once you've defined a class, you can create multiple objects based on that class. Each object is a separate instance with its own set of fields and can call the methods defined in the class.

Key points to remember:

Declaration and Instantiation: To create an object, you first declare a variable of the class type and then use the new keyword followed by the class constructor to create the object.

Accessing Fields and Methods: You can access fields and methods of an object using the dot notation (objectName.fieldName or objectName.methodName()).

Each Object is Independent: Objects created from the same class are independent of each other. Changing the state of one object does not affect the state of other objects.

Constructor: The constructor is called when an object is created. It initializes the object's state.

Memory Allocation: When you create an object, Java allocates memory for it on the heap.

Object Reference: The variable you use to refer to an object actually holds a reference to the object in memory. It's important to understand that you're working with references rather than direct memory addresses.

Class Methods vs. Instance Methods: Some methods in a class can be declared as static, which means they belong to the class itself and not to any specific instance. Instance methods, on the other hand, operate on specific instances of the class.

Remember that classes define the structure and behavior of objects, and objects are the actual instances that hold data and execute methods.

3.Constructors :

In Java, a constructor is a special type of method that is automatically invoked when an object of a class is created. The primary purpose of a constructor is to initialize the newly created object, setting its initial state and performing any necessary setup tasks. Constructors have the same name as the class and do not have a return type, not even void.

Here are the key points to know about constructors in Java:

Constructor Naming: Constructors have the same name as the class they belong to. This is how Java identifies them as constructors.

No Return Type: Unlike regular methods, constructors do not have a return type, not even void.

Overloading: A class can have multiple constructors, each with different parameter lists. This is known as constructor overloading.

Default Constructor: If a class does not explicitly define any constructors, Java provides a default constructor with no parameters. It initializes fields to their default values (0 for numeric types, null for reference types, etc.).

Initializing Fields: Constructors are used to initialize the fields of an object to desired values.

4. this Keyword: Inside a constructor, the this keyword refers to the object being created. It is often used to distinguish between instance variables and constructor parameters with the same name.

Chaining Constructors: Constructors can call other constructors using the **this() keyword** to avoid duplicate initialization code. This is known as constructor chaining.

5.new keyword : In Java, the new keyword is used to create new instances (objects) of classes. It is followed by a constructor call and creates a new object on the heap memory. Here's how the new keyword is used:

ClassName objectName = new ClassName();

6.static fields

In Java, a static field (also known as a class variable) is a field that belongs to the class itself, rather than to any particular instance of the class. This means that the value of a static field is shared among all instances of the class. You define a static field using the static keyword in the field declaration.

key points about static fields:

Shared Value: All instances of the class share the same value for a static field. If the value of a static field is changed, it will be reflected in all instances of the class.

Accessing Static Fields: You can access static fields using the class name followed by the field name (e.g., ClassName.staticField). You can also access static fields using an instance of the class, but this is generally considered bad practice.

Initialization: Static fields are initialized when the class is loaded, which happens when the class is first referenced in the code.

Memory Allocation: Static fields are stored in the memory associated with the class, not with individual instances of the class.

Static fields are commonly used for constants, configuration settings, counters, and other data that should be shared among all instances of a class.

7.instance fields :

In Java, an instance field (also known as an instance variable) is a field that belongs to a specific instance (object) of a class. Each object created from the class has its own set of instance fields, which are used to store individual state or data for that object.

key points about instance fields:

Per-Object Data: Each instance of a class has its own set of instance fields. Changes to instance fields in one object do not affect the values in other objects.

Accessing Instance Fields: Instance fields are accessed using the dot notation, which involves using the object reference followed by the field name (e.g.,

objectName.instanceField).

Initialization: Instance fields are usually initialized in constructors or methods. They are not automatically initialized by Java and will have default values (0 for numeric types, null for reference types) until explicitly assigned a value.

Memory Allocation: Memory for instance fields is allocated when an object is created, and each instance of the class has its own set of instance fields.

Instance Methods Access: Instance methods can access both instance fields and static fields (class variables) of the same class directly.

8.instance initializer block :

An instance initializer block is a block of code enclosed in curly braces {} and is not associated with any specific method or constructor. It is executed when an instance of the class is created, just before the constructor is called. This allows you to initialize instance fields or perform other tasks that are common to all constructors of the class.

9. static initializer block :

It is similar to an instance initializer block, but it's marked with the static keyword. It's executed when the class is loaded into memory, Static initializer blocks are used to perform class-level initialization tasks, like setting up static fields or loading resources.

instance method vs static method :

In Java, both instance methods and static methods are used to define behaviors within a class. However, they have distinct differences in how they are defined, accessed, and utilized. Here's a breakdown of the differences between instance methods and static methods:

Instance Methods:

Definition: Instance methods are defined within a class without the static keyword. They are associated with an instance of the class and operate on its specific data.

Access: Instance methods can access instance fields and other instance methods directly without the need for a reference to the instance. They can also access static fields and static methods through the class name.

Usage: Instance methods are used when you need to perform actions or calculations that depend on the specific instance's state.

Invocation: To call an instance method, you need to create an instance of the class and then use the instance to call the method using the dot notation (objectName.methodName()).

Static Methods:

Definition: Static methods are defined with the static keyword within a class. They are associated with the class itself, not with instances, and operate on class-level data.

Access: Static methods can access only static fields and other static methods directly. They cannot access instance fields or instance methods without a reference to a specific instance.

Usage: Static methods are used when you have utility methods that don't require access to instance-specific data, or when you need to perform operations at the class level.

Invocation: Static methods are called using the class name followed by the method name (ClassName.methodName()).

Static methods can be called using the class name without creating instances, whereas instance methods require an instance to be created first.

Copy Constructor: A copy constructor is a constructor in a class that takes an object of the same class as a parameter and creates a new object by copying the contents of the provided object. It's used to create a new instance that's a copy of an existing instance. The copy constructor is often used to ensure deep copy behavior