**1.Constructors:**

A constructor in Java is a special method used to initialize objects. It is called when an object of a class is created.

Constructors have the same name as the class and do not have a return type.

**Types of Constructors:**

**Default Constructor:** A constructor with no parameters. If you don't define any constructor in your class, Java provides a default constructor automatically.

Ex:

class MyClass {

MyClass() { // Default constructor

System.out.println("Default constructor called");

}

}

**Parameterized Constructor**: A constructor that takes arguments to initialize an object with specific values.

Ex:

class MyClass {

int x;

MyClass(int value) { // Parameterized constructor

x = value;

}

}

**2.Types of Fields (Instance Variables)**

Fields, also known as instance variables, are variables declared within a class but outside of any method.

**Types of Fields:**

Instance Variables: Belong to an instance of a class. Each object has its own copy.

Ex:

class MyClass {

int instanceVar; // Instance variable

}

**Static Variables:** Belong to the class rather than any instance. Shared among all instances of the class.

Ex:

class MyClass {

static int staticVar; // Static variable

}

**Local Variables**: Declared within a method and are only accessible within that method.

Ex:

class MyClass {

void myMethod() {

int localVar = 5; // Local variable

}

}

**Final Variables:** Variables declared with the `final` keyword that cannot be changed once initialized.

Ex:

class MyClass {

final int finalVar = 10; // Final variable

}

**3.Types of Methods**

Methods in Java define the behavior of an object. They are blocks of code that perform a specific task.

**Types of Methods:**

**Instance Methods**: Operate on instances of a class (objects). They can access instance variables.

Ex:

class MyClass {

void instanceMethod() {

// Code

}

}

**Static Methods:** Belong to the class and can be called without creating an instance of the class. They can only access static variables directly.

Ex:

class MyClass {

static void staticMethod() {

// Code

}

}

**Abstract Methods:** Declared without a body in an abstract class, meant to be overridden in a subclass.

Ex:

abstract class MyClass {

abstract void abstractMethod(); // Abstract method

}

**Final Methods:** Cannot be overridden by subclasses.

Ex:

class MyClass {

final void finalMethod() {

// Code

}

}

**Synchronized Methods:** Control access to a method by multiple threads to prevent thread interference.

Ex:

class MyClass {

synchronized void synchronizedMethod() {

// Code

}

}

**4.this` Keyword:**

The `this` keyword refers to the current instance of a class. It is often used to resolve naming conflicts between instance variables and parameters or to pass the current instance to another method or constructor.

Ex:

class MyClass {

int x;

MyClass(int x) {

this.x = x; // 'this.x' refers to the instance variable, 'x' refers to the parameter

}

}

**5.this() Method:**

The this() method call is used within a constructor to invoke another constructor in the same class. This is useful for constructor chaining, where one constructor calls another to reuse code.

Ex:

class MyClass {

int x;

MyClass() {

this(10); // Calls the parameterized constructor

}

MyClass(int x) {

this.x = x;

}

}

**Object-Oriented Programming (OOP) principles:**

1.**Class:** A blueprint or template.

Ex: Dog class defines what a dog is like (name, breed) and what it can do (bark).

2.**Object:** A specific instance of a class.

Ex: myDog is an object made from the Dog class with its own name and breed.

3.**Encapsulation:**

Encapsulation is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

**4.Inheritance:**

Inheritance allows a new class (child or subclass) to inherit properties and methods from an existing class (parent or superclass).

Inheritance represents the IS-A relationship which is also known as a *parent-child* relationship.

**Types of Inheritance:**

**Single Inheritance:** A class inherits from one superclass.

**Multilevel Inheritance:** A class inherits from another class, and the child class is further inherited by another class.

Ex:

class Vehicle { }

class Car extends Vehicle { }

class SportsCar extends Car { }

**Hierarchical Inheritance:** Multiple classes inherit from a single superclass.

class Vehicle { }

class Car extends Vehicle { }

class Bike extends Vehicle { }

**Multiple Inheritance (via interfaces):** Java does not support multiple inheritance with classes directly to avoid complexity and ambiguity. However, it allows a class to implement multiple interfaces, which is a way to achieve multiple inheritance

**Hybrid Inheritance:** This is a combination of two or more types of inheritance.

**5.Polymorphism:**

It is a concept by which we can perform a single action in different ways.

Polymorphism can be achieved through method overloading and method overriding.

**6.Abstraction:** Abstraction is a process of hiding the implementation details and showing only functionality to the user. In Java, abstraction is achieved using abstract classes and interfaces.