OOPS

1. Explain with example the concepts of overloading , overriding and dynamic method dispatch in java .

   Ans:

   a. Method Overloading:

      Method Overriding occurs when a **subclass provides a specific implementation** of a method that is already defined in its **superclass**.

   Key Points:

- Must have the **same method name, return type, and parameters**.
- Enables **runtime polymorphism**.
- Access modifier in the child class method should be **same or more visible**.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
```

   b. Method Overloading:

      Method Overloading means having **multiple methods with the same name** in the same class but with **different parameters** (number, type, or order).

   Key Points:

- Happens at **compile time** (Compile-time polymorphism).
- Improves **code readability** and reusability.

Example:

```
class MathOperation {
   int add(int a, int b) {
      return a + b;
   }

   double add(double a, double b) {
      return a + b;
   }
}
```

c. Dynamic Method Dispatch:

Dynamic Method Dispatch is a mechanism by which a **call to an overridden method** is resolved at **runtime** rather than compile time.

Key Points:

- Achieved using **method overriding**.
- Enables **runtime polymorphism**.
- Requires a **superclass reference** pointing to a **subclass object**.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
        }
    }

class Cat extends Animal {
    void sound() {
        System.out.println("Cat meows");
        }
    }
public class Main {
   public static void main(String[] args) {
      Animal obj = new Cat();              // Superclass reference, subclass object
      obj.sound();                         // Calls Cat's version at runtime
      }
}
```

2. With help of a neat diagram explain life cycle of Thread.

Ans:

1. Newborn State :

- A thread is in this state when it is **created** using the Thread class.
- Example: Thread t = new Thread();
- The thread is not yet started.

---

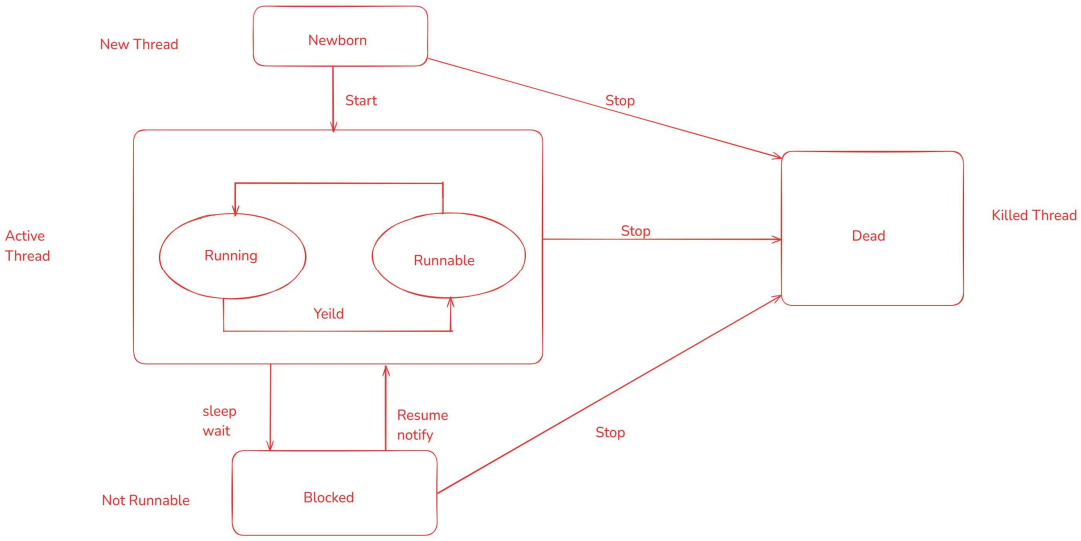
2. Runnable State:

- When the start() method is called, the thread moves to the **Runnable** state.
- In this state, the thread is **ready to run** but **waiting for CPU** to allocate time for execution.
- Example: t.start();

---

3. Running State:

- The thread is in the **Running** state when the **CPU selects it for execution**.
- Only one thread can be in the Running state at a time per processor core.



4. Blocked or Not Runnable State:

- A thread moves to this state when it is **inactive temporarily**.
- This can happen due to:
  - sleep()
  - wait()
- It will move back to **Runnable** state when notified or when the sleep/wait time ends.

---

5. Dead State :

- A thread enters this state when:
  - It **completes execution**,
  - Or it is **forcefully stopped** using stop().
- A dead thread **cannot be restarted**.

3. Discuss some of the properties of OOP.
   Ans:

   Features of OOPS:

   A. Encapsulation:

      i. Encapsulation is the **binding of data and methods** that operate on that data into a single unit (class).

      ii. It **protects data** from direct access by using access modifiers (private, public, etc.).

Example:

```
class Student {
    private int marks;
    public void setMarks(int m) {
        marks = m;
    }
    public int getMarks() {
        return marks;
    }
}
```

   B. Inheritance:

      i. Inheritance allows a class (**subclass**) to **inherit properties and methods** from another class (**superclass**).

      ii. Promotes **code reusability**.

Example:

```
class Animal {
    void eat() { System.out.println("Eating"); }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Barking");
```

```
    }
}
```

C. Polymorphism:

i. Polymorphism means **many forms**. A method behaves differently based on the object.

ii. It is of two types:
   a. Compile-time (Method Overloading)
   b. Run-time (Method Overriding)

Example:

```
void show(int a) {}
void show(String s) {}
```

D. Abstraction:

i. Abstraction means **hiding internal implementation** and showing only necessary details.
ii. Achieved using **abstract classes** and **interfaces**.

Example:

```
abstract class Shape {
  abstract void draw();
    }
```

E. Class and Object:

i. **Class** is a blueprint/template for creating objects.
ii. **Object** is an instance of a class containing state (variables) and behavior (methods).

Example:

```
class Car {
   void run() { System.out.println("Car is running"); }
   }
  Car c = new Car(); // object creation
```

4. What is Multithreading? What will happen if we call the run() method directly without calling start() method in Thread?

   Ans:

   Multithreading:
   Multithreading is a **feature of Java** that allows the execution of **two or more threads concurrently**.
   Each thread runs independently and shares the same memory, improving the performance of programs, especially on **multi-core processors**.

   ✅ **What happens if we call run() method directly instead of start()?**

   If we call the run() method **directly**, it will behave like a **normal method call**, and **no new thread will be created**.

   Instead:

- Code inside run() will execute in the **current thread**, not in a separate thread.
- So, it **won't achieve multithreading**.

5. What is applet?
   Ans:

   An **Applet** is a **small Java program** that is **embedded in a web page** and runs in a **browser** using the **Java Plugin**.
6. What is super keyword?
   Ans:

   The super keyword in Java is a reference variable used to refer to the **immediate parent class** object. It is mainly used in inheritance to access members (methods, variables, constructors) of the parent class from a subclass.
   Example use case:

   Access parent class variables:
   When a subclass has a variable with the same name as a parent class variable, super can be used to refer to the parent class variable.

```
class Parent {
   int x = 10;
}

class Child extends Parent {
   int x = 20;

   void display() {
      System.out.println(x);
      System.out.println(super.x);     } }
```

7. What is constructor?
   Ans:

   A **constructor** in Java is a special method that is called automatically when an object of a class is created. Its main purpose is to initialize the object.

- It has the same name as the class.
- It does **not** have a return type (not even void).

```
Example:
class Car {
   String model;

   // Constructor
   Car(String modelName) {
      model = modelName;
   }
}

public class Main {
   public static void main(String[] args) {
      Car myCar = new Car("Tesla");
      System.out.println(myCar.model);  // Output: Tesla
   }
}
```

8. Describe class variable.
   Ans:

   A **class variable** in Java is a variable that is declared with the static keyword inside a class but **outside any method or constructor**.
   Key points:

- It is **shared among all objects** of the class.
- Only **one copy** exists regardless of how many objects are created.
- It belongs to the **class itself**, not to any specific instance.

```
Example:
class Student {
   static int schoolCode = 101;                  // class variable
   String name;

   Student(String n) {
      name = n;
   }
}
```

9. What are access modifiers?
   Ans:

In Java, **access modifiers** are keywords used to define the **visibility (access level)** of classes, methods, constructors, and variables.

Access Modifier	Same Class	Same Package	Subclass (Different Package)	Other Packages
`private`	✅ Yes	❌ No	❌ No	❌ No
default (no modifier)	✅ Yes	✅ Yes	❌ No	❌ No
`protected`	✅ Yes	✅ Yes	✅ Yes	❌ No
`public`	✅ Yes	✅ Yes	✅ Yes	✅ Yes

10. How to create own package in java .
    Ans:

How to Create Your Own Package in Java :
In Java, a **package** is a namespace that organizes a set of related classes and interfaces. It helps avoid class name conflicts and makes it easier to maintain and locate classes.

Steps to Create a Package:

1. Use the package keyword at the **top** of your Java file.
2. Save the file in a folder with the **same name** as the package.
3. Compile and run using the appropriate directory structure.

Example:
1. Creating the Package university:
File: university/Student.java

```
package university;

public class Student {
    public void display() {
        System.out.println("Welcome to the university package!");
    }
}
```

2. Using the Package:
File: Main.java

```
import university.Student;

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.display();
    }
}
```

Compilation and Execution:
javac university/Student.java
javac Main.java
java Main
Output:
Welcome to the university package!

---

11. What do you mean by constructor overloading?
    Ans:

    Constructor Overloading in Java
    Definition:
    Constructor overloading means defining **multiple constructors** in the same class with **different parameters** (number or type). It allows creating objects in different ways.

    Example:

```
class Student {
  Student() {
    System.out.println("Default constructor");
  }

  Student(String name) {
    System.out.println("Name: " + name);
  }
}
```

12. Describe Inheritance in java.
    Ans:

    Inheritance in Java
    Definition:
    Inheritance is a fundamental concept in Object-Oriented Programming (OOP) where one class (**child/subclass**) inherits the **properties and behaviors** (fields and methods) of another class (**parent/superclass**). It promotes **code reusability** and **method overriding**.

Types of Inheritance in Java:

1. **Single Inheritance** – One subclass inherits from one superclass.
2. **Multilevel Inheritance** – A subclass inherits from a class that itself inherits from another class.
3. **Hierarchical Inheritance** – Multiple subclasses inherit from a single superclass.
   (*Note: Java does not support multiple inheritance with classes.*)

Example:

```
class Parent {
  void display() {
```

```
        System.out.println("Parent class");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("Child class");
    }
}
```

13. Describe Compiletime and Runtime errors.
    Ans:

    Compile-Time Error
    Definition:
    Errors that occur **when you compile** the program. These are usually **syntax** or **semantic** errors caught by the compiler before the program runs.
    Examples:

- Missing semicolon
- Undeclared variable
- Wrong method signature

    Example Code:

```
int x = "hello";  // Error: incompatible types
```

    Fix: Correct the code before running.

    Run-Time Error
    Definition:
    Errors that occur **while the program is running**. These are usually caused by **illegal operations** or **unexpected inputs**.
    Examples:

- Dividing by zero
- Accessing array out of bounds
- Null pointer access

    Example Code:

```
int a = 10 / 0;  // Run-time error: ArithmeticException
```

14. Describe Interface.
    Ans:
    Interface in Java
    Definition:
    An **interface** is a **reference type** in Java that is similar to a class but only contains **abstract methods** (methods without body) and **constants**. It defines a **contract** that classes can implement.

    Key Points:

- Interfaces contain **method declarations** but no method implementations (except default and static methods in newer Java versions).
- A class **implements** an interface using the implements keyword.
- Supports **multiple inheritance** since a class can implement multiple interfaces.
- All methods in an interface are **implicitly public and abstract**.

Example:

```
interface Animal {
   void sound();  // abstract method
}

class Dog implements Animal {
   public void sound() {
      System.out.println("Bark");
   }
}
```

15. How to throw exception in java ?
    Ans:

    How to Throw Exception in Java
    In Java, you can **throw an exception explicitly** using the throw keyword followed by an instance of Throwable (usually an Exception).

    Syntax:
    throw new ExceptionType("Error message");
16. Describe Exception handling in java.
    Ans:

    Exception Handling in Java
    Definition:
    Exception handling in Java is a mechanism to handle runtime errors so that the normal flow of the program is maintained. It helps in detecting and managing errors gracefully without crashing the program.

    Why Exception Handling?

- To handle unexpected errors at runtime.
- To maintain normal flow of application.
- To separate error-handling code from regular code.

Key Keywords:

Key Keywords:

Keyword	Purpose
`try`	Block of code to monitor for exceptions.
`catch`	Handles the exception thrown in `try`.
`finally`	Executes code regardless of exception.
`throw`	Used to explicitly throw an exception.
`throws`	Declares exceptions that a method can throw.

Example:

```
try {
   int result = 10 / 0;     // Causes ArithmeticException
} catch (ArithmeticException e) {
   System.out.println("Cannot divide by zero.");
} finally {
   System.out.println("Execution completed.");
}
```

17. Write difference between class and abstract class.
    Ans:

Aspect	Class	Abstract Class
Definition	A blueprint for objects with concrete implementation.	A class that **cannot be instantiated** and may contain abstract methods.
Instantiation	Can create objects directly.	Cannot create objects directly.
Methods	Can have fully implemented methods only.	Can have both abstract (without body) and concrete methods.
Purpose	Used to create objects with complete behavior.	Used to provide a common base with some methods to be implemented by subclasses.
Abstract methods	Not allowed.	Allowed and may contain one or more.
Usage	Used when full implementation is ready.	Used when some methods need to be implemented by subclasses.
Keyword	`class`	`abstract class`