**Compiling and executing java program:**

class HelloWorld

{

    public static void main(String args[])

    {

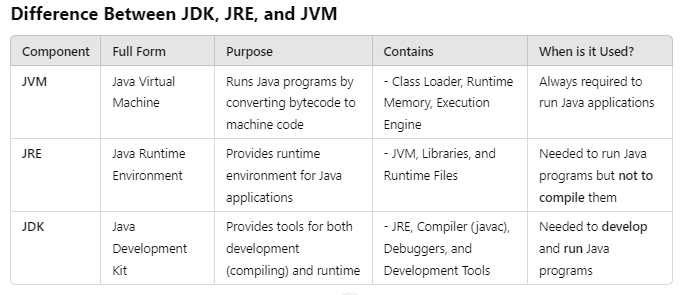
        System.out.println("Hello World");

    }

}

**compiling :** javac HelloWorld.java // After compiling .class file will be created

**Executing :** java HelloWorld



**What is JVM?**  
JVM (Java Virtual Machine) is responsible for **loading and executing Java programs(.class files)**.

**JVM Specification:**  
The JVM specification is a document that defines the requirements for a JVM implementation.  
This ensures all implementations are **interoperable**.

**Popular JVM Implementations:**

* Oracle JVM (HotSpot)
* OpenJDK
* IBM OpenJ9
* SAP JVM

**Development of JVM:**  
Oracle JVM is developed using **C/C++**.

**Different JVM's for Different OS:**

https://www.oracle.com/java/technologies/downloads/#jdk23-linux

The **four pillars of Object-Oriented Programming (OOP)** are:

1)Encapsulation

2)Abstraction

3)Inheritance

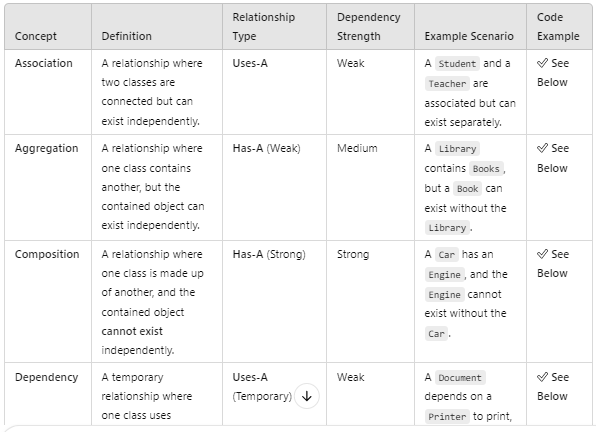
4)Polymorphism

**Encapsulation :**

· Binding of data and corresponding methods into a single unit is called Encapsulation .

· If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Note : We will discuss Abstraction, inheritance and Polymorphism in next sessions



## **1. Association Example: Student-Teacher Relationship**

🔹 **Objects are related but can exist independently.**

class Student {

String name;

Student(String name) {

this.name = name;

}

}

class Teacher {

String name;

Teacher(String name) {

this.name = name;

}

void teaches(Student student) {

System.out.println(this.name + " teaches " + student.name);

}

}

public class Main {

public static void main(String[] args) {

Student s = new Student("Alice");

Teacher t = new Teacher("Mr. John");

t.teaches(s);

}

}

✅ **Output:**

Mr. John teaches Alice

📌 **Key Point:** **Objects (Student & Teacher) exist independently but interact with each other.**

## **2. Aggregation Example: Library & Books**

🔹 **Library has Books, but Books can exist without Library.**

java

CopyEdit

import java.util.List;

class Book {

String title;

Book(String title) {

this.title = title;

}

}

class Library {

String name;

List<Book> books;

Library(String name, List<Book> books) {

this.name = name;

this.books = books;

}

void showBooks() {

System.out.println(name + " Library has:");

for (Book book : books) {

System.out.println(book.title);

}

}

}

public class Main {

public static void main(String[] args) {

Book b1 = new Book("Java Programming");

Book b2 = new Book("Data Structures");

Library library = new Library("Central", List.of(b1, b2));

library.showBooks();

}

}

✅ **Output:**

Central Library has:

Java Programming

Data Structures

📌 **Key Point:** **If the Library is deleted, Books can still exist separately.**

## **3. Composition Example: Car & Engine**

🔹 **Car has an Engine, and Engine cannot exist without Car.**

class Engine {

void start() {

System.out.println("Engine started.");

}

}

class Car {

private Engine engine;

Car() {

this.engine = new Engine(); // Strong dependency

}

void startCar() {

engine.start();

System.out.println("Car started.");

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.startCar();

}

}

✅ **Output:**

nginx

CopyEdit

Engine started.

Car started.

📌 **Key Point:** **If Car is destroyed, Engine is also destroyed.**

## **4. Dependency Example: Document & Printer**

🔹 **A Document depends on a Printer to print, but the Printer exists separately.**

class Printer {

void print(String message) {

System.out.println("Printing: " + message);

}

}

class Document {

void printDocument(Printer printer) {

printer.print("Document is being printed...");

}

}

public class Main {

public static void main(String[] args) {

Printer printer = new Printer();

Document doc = new Document();

doc.printDocument(printer);

}

}

✅ **Output:**

Printing: Document is being printed...

📌 **Key Point:** **The Document uses Printer, but Printer is independent.**

## **5. Message Passing Example: Person Greeting**

🔹 **Objects interact via method calls (message passing).**

class Person {

void sayHello(String name) {

System.out.println("Hello, " + name);

}

}

public class Main {

public static void main(String[] args) {

Person p = new Person();

p.sayHello("John"); // Passing a message (method call)

}

}

✅ **Output:**

Hello, John

📌 **Key Point:** **Method calls act as "messages" between objects.**

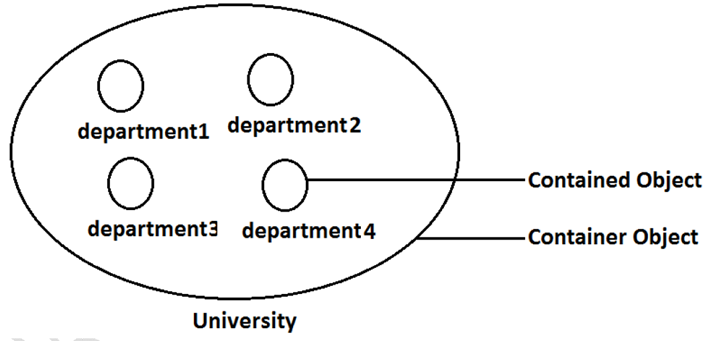
**Composition vs Aggregation:**

**Composition:**

Without existing container object if there is no chance of existing contained objects then the relationship between container object and contained object is called composition which is a strong association.

**Example:**

University consists of several departments whenever university object destroys automatically all the department objects will be destroyed that is without existing university object there is no chance of existing dependent object hence these are strongly associated and this relationship is called composition.



**Aggregation :**

Without existing container object if there is a chance of existing contained objects such type of relationship is called aggregation. In aggregation objects have weak association.

**Example:**

Within a department there may be a chance of several professors will work whenever we are closing department still there may be a chance of existing professor object without existing department object the relationship between department and professor is called aggregation where the objects having weak association.

