**Data Abstraction Using Interface in Java**

In Java, **interfaces** are another way to achieve **data abstraction**. An interface defines **what should be done** but does **not specify how it should be done**. The actual implementation is left to the classes that implement the interface.

**Example: Data Abstraction Using an Interface**

Let's take a **Vehicle** example where a user can start a vehicle **without knowing how the engine starts internally**.

// Interface providing Data Abstraction

interface Vehicle {

void start(); // Abstract method (no implementation)

void stop();

}

// Concrete class implementing the interface

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car starts with a key or push button.");

}

@Override

public void stop() {

System.out.println("Car stops by applying brakes.");

}

}

// Concrete class implementing the interface

class Bike implements Vehicle {

@Override

public void start() {

System.out.println("Bike starts with a self-start or kick.");

}

@Override

public void stop() {

System.out.println("Bike stops by pressing the brake lever.");

}

}

// Main class

public class DataAbstractionWithInterface {

public static void main(String[] args) {

Vehicle myCar = new Car();

myCar.start(); // Car starts with a key or push button.

myCar.stop(); // Car stops by applying brakes.

System.out.println("------------------");

Vehicle myBike = new Bike();

myBike.start(); // Bike starts with a self-start or kick.

myBike.stop(); // Bike stops by pressing the brake lever.

}

}

**Why is This Data Abstraction?**

1. **Hides Internal Details:**
   * The **user does not know how start() works internally** for different vehicles.
2. **Defines Behavior Without Implementation:**
   * The Vehicle interface **only declares methods** (start(), stop()) but doesn’t define how they work.
3. **Different Implementations:**
   * Car and Bike provide **different implementations** of start() and stop().

**Summary**

* **Interfaces provide data abstraction** by **hiding implementation details**.
* **Only necessary details (methods) are exposed** to the user.
* The actual **implementation is handled by concrete classes**.