

# Multiple Inheritance

In this lesson, you'll learn what multiple inheritance is and how it can be implemented in Java.

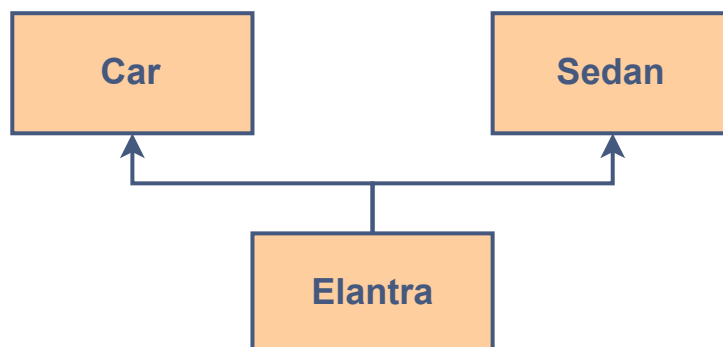
## WE'LL COVER THE FOLLOWING ^

- What Is Multiple Inheritance?
- How to Implement
- An Example
- Interface vs Abstract Class

## What Is Multiple Inheritance? #

When a class is derived from more than a single base class, i.e. when a class has more than one immediate parent classes, it is an instance of **Multiple Inheritance**. **Example:**

- A Hyundai **Elantra** IS A **Car**
- A Hyundai **Elantra** IS A **Sedan** as well



## How to Implement #

As mentioned earlier, in Java, a class can't extend from more than one class. So the question arises, *“how can we implement multiple inheritance?”*

The answer to the above question is **Interfaces**. In Java, *multiple inheritance* can be implemented using interfaces

can be implemented using interfaces.

A class can **implement** more than one interfaces and an interface can **extend** from more than one interfaces.

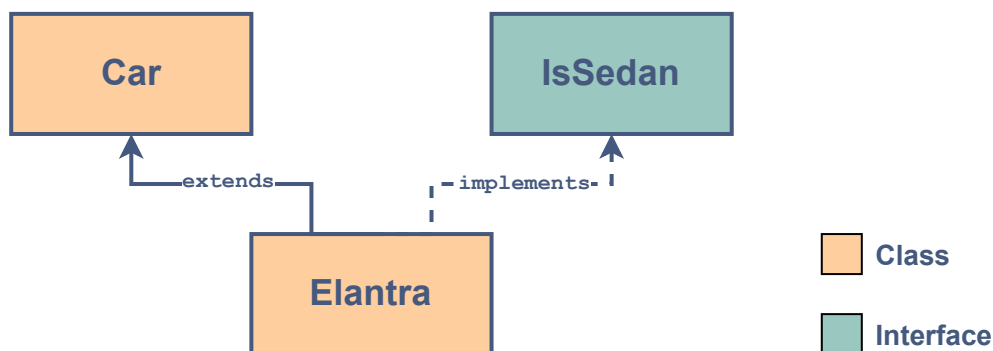
So in this way, we can achieve multiple inheritance in Java.

## An Example #

Let's implement the example of Elantra given at the start of the lesson. This example can be implemented using:

- A base *class* named **Car**
- An *interface* named **IsSedan**
- An **Elantra** class derived from **Car** and implementing **IsSedan**

The above illustration then becomes:



Below is the implementation:

```
class Car { // Base class

    private int model; // Common features of all cars
    private String manufacturer;

    public Car(int model, String manufacturer) { // Constructor
        this.model = model;
        this.manufacturer = manufacturer;
    }

    public void printDetails() {

        System.out.println("The model of " + getClass().getSimpleName() + " is: " + model);
        System.out.println("The manufacturer of " + getClass().getSimpleName() + " is: " + manufa
    }

} // End of Car class
```

```

interface IsSedan { // Interface for sedans

    int bootSpace = 420; // Sedans have boot space

    void bootSpace(); // Every sedan must implement this
} // End of IsSedan interface

class Elantra extends Car implements IsSedan { // Elantra is a Car and is a Sedan also

    private String variant; // Elantra's data member

    public Elantra(int model, String variant) { // Constructor
        super(model, "Hyundai"); // Calling the parent constructor with already known manufacture
        this.variant = variant;
    }

    @Override
    public void bootSpace() { // Implementation of the interface method
        System.out.println("The bootspace of Elantra is: " + IsSedan.bootSpace + " litres");
    }

    @Override
    public void printDetails() { // Overriding the parent class's inherited method
        super.printDetails(); // Calling the method from parent class
        System.out.println("The variant of Elantra is: " + variant); // printing the data member
    }

} // End of Elantra class

class Main {

    public static void main(String[] args) {

        Elantra sport = new Elantra(2019, "Sport"); //creating Sports variant Elantra
        Elantra eco = new Elantra(2018, "Eco"); //creating Eco variant Elantra

        sport.printDetails();
        sport.bootSpace();

        System.out.println();

        eco.printDetails();
        eco.bootSpace();
    }

}

```



Now that we've implemented multiple inheritance, let's take a look at the differences between an interface and an abstract class.

## Interface vs Abstract Class #

Interfaces and abstract classes are both used to achieve abstraction but with some of the key differences:

Interfaces	Abstract Classes
Can have abstract method(s) only.	Can have concrete (non-abstract) & abstract method(s)
Support multiple inheritance	Don't support multiple inheritance
All members are <code>public</code>	Can have <code>private</code> , <code>protected</code> and <code>public</code> members
All data members are <code>static</code> and <code>final</code>	Can have non-static and non-final members too
Can't have constructors	Constructors can be defined