Inheritance in Java

1. Vehicle Management System:

Problem Statement: Design a vehicle management system where there are multiple types of vehicles: **Cars**, **Bikes**, and **Trucks**. All these vehicles share some common attributes (like `speed`, `fuel`, `capacity`) and behaviors (like `start()`, `stop()`), but each has unique features, such as `loadCapacity` for trucks, `typeOfBike` for bikes, etc.

- **Explanation**:
  - Create a base class `Vehicle` that contains the common attributes and methods.
  - Create subclasses `Car`, `Bike`, and `Truck` that inherit from `Vehicle`.
  - Each subclass can have additional attributes and behaviors specific to that type of vehicle.

2. Employee Management System:

Problem Statement: Develop an Employee Management System for a company. The company has different types of employees: **Full-Time Employees**, **Part-Time Employees**, and **Contract Employees**. All employees share common details like `name`, `id`, and `salary`, but the calculation of salary varies for each type of employee.

- **Explanation**:
  - Define a base class `Employee` with common fields and methods.
  - Create subclasses `FullTimeEmployee`, `PartTimeEmployee`, and `ContractEmployee` that extend the `Employee` class.
  - Override the salary calculation method for each subclass.

3. Banking System:

Problem Statement: Create a banking system where there are multiple types of accounts, such as **Savings Account** and **Current Account**. Both accounts share common features like `accountNumber`, `balance`, and methods like `deposit()` and `withdraw()`. However, savings accounts have an additional feature of interest calculation, while current accounts have an overdraft limit.

- **Explanation**:
  - Create a base class `BankAccount` that contains shared properties like `accountNumber` and `balance`.
  - Define subclasses `SavingsAccount` and `CurrentAccount` that extend the `BankAccount` class.
  - Each subclass will have its own specific attributes and methods.

4. Online Shopping System:

Problem Statement: Create an online shopping system where different types of users can place orders: **Guest Users** and **Registered Users**. All users share common details like `name`, `email`, and `cart`. However, registered users can have additional benefits like discounts and loyalty points, while guest users do not.

- **Explanation**:
    - Create a base class `User` with shared attributes and methods.
    - Subclass `GuestUser` and `RegisteredUser` to handle specific behaviors like applying discounts or using loyalty points.

5. Library Management System:

Problem Statement: In a library management system, there are two types of users: **Students** and **Teachers**. Both users can borrow books, but students can borrow up to 5 books at a time, while teachers can borrow up to 10 books. All users have common attributes like `name` and `ID`, and a method to `borrowBook()`.

- **Explanation**:
    - Create a base class `LibraryUser` with common attributes.
    - Subclass `Student` and `Teacher` to define the borrowing limits and other specific rules.

Key Concepts Demonstrated:

- **Inheritance**: The child classes (`Car`, `FullTimeEmployee`, `SavingsAccount`, etc.) inherit common properties and behaviors from the parent classes (`Vehicle`, `Employee`, `BankAccount`, etc.).
- **Method Overriding**: Child classes can provide their own specific implementation of the inherited methods.
- **Reusability**: The parent class code is reused by child classes, reducing redundancy.

Has-A relationship

The **Has-A relationship** in Java, also known as **composition**, is when one class contains a reference to another class as one of its attributes. It's typically used to model situations where one object is made up of or uses another object.

real-time example problem statements:

1. Car and Engine Example

- **Problem Statement**: Create a `Car` class that contains an `Engine`. The `Car` class should have attributes like `brand` and `model`, and the `Engine` class should have attributes like `engineType` and `horsePower`. Demonstrate the **Has-A relationship** by making the `Car` class contain an object of the `Engine` class.
- **Explanation**: A car "has-an" engine, meaning the `Car` class uses the `Engine` class as one of its fields.

2. Library and Books Example

- **Problem Statement**: Create a `Library` class that contains multiple `Book` objects. The `Library` class should have attributes like `libraryName` and `location`, and the `Book` class should have attributes like `title`, `author`, and `isbn`. Demonstrate the **Has-A relationship** by making the `Library` class contain a list of `Book` objects.
- **Explanation**: A library "has" books, meaning the `Library` class has a collection of `Book` objects.

3. Person and Address Example

- **Problem Statement**: Create a `Person` class that contains an `Address`. The `Person` class should have attributes like `name` and `age`, and the `Address` class should have attributes like `street`, `city`, and `zipCode`. Demonstrate the **Has-A relationship** by making the `Person` class contain an object of the `Address` class.
- **Explanation**: A person "has-an" address, meaning the `Person` class has an `Address` object as a field.

4. Company and Employee Example

- **Problem Statement**: Create a `Company` class that contains multiple `Employee` objects. The `Company` class should have attributes like `companyName` and `location`, and the `Employee` class should have attributes like `employeeName`, `designation`, and `salary`. Demonstrate the **Has-A relationship** by making the `Company` class contain a list of `Employee` objects.
- **Explanation**: A company "has" employees, meaning the `Company` class has a collection of `Employee` objects.

5. Computer and Monitor Example

- **Problem Statement**: Create a `Computer` class that contains a `Monitor`. The `Computer` class should have attributes like `processor` and `ram`, and the `Monitor` class should have attributes like `resolution` and `size`. Demonstrate the **Has-A relationship** by making the `Computer` class contain an object of the `Monitor` class.
- **Explanation**: A computer "has-a" monitor, meaning the `Computer` class has a `Monitor` object as one of its fields.

6. Mobile Phone and Battery Example

- **Problem Statement**: Create a `MobilePhone` class that contains a `Battery`. The `MobilePhone` class should have attributes like `brand` and `model`, and the `Battery` class should have attributes like `capacity` and `type`. Demonstrate the **Has-A relationship** by making the `MobilePhone` class contain an object of the `Battery` class.
- **Explanation**: A mobile phone "has-a" battery, meaning the `MobilePhone` class contains a `Battery` object.

7. Student and Course Example

- **Problem Statement**: Create a `Student` class that contains a list of `Course` objects. The `Student` class should have attributes like `studentName` and `rollNumber`, and the `Course` class should have attributes like `courseName` and `credits`. Demonstrate the **Has-A relationship** by making the `Student` class contain a list of `Course` objects.
- **Explanation**: A student "has" courses, meaning the `Student` class contains a collection of `Course` objects.

8. Flight and Pilot Example

- **Problem Statement**: Create a `Flight` class that contains a `Pilot`. The `Flight` class should have attributes like `flightNumber` and `destination`, and the `Pilot` class should have attributes like `name` and `licenseNumber`. Demonstrate the **Has-A relationship** by making the `Flight` class contain a `Pilot` object.
- **Explanation**: A flight "has-a" pilot, meaning the `Flight` class has a `Pilot` object as a field.