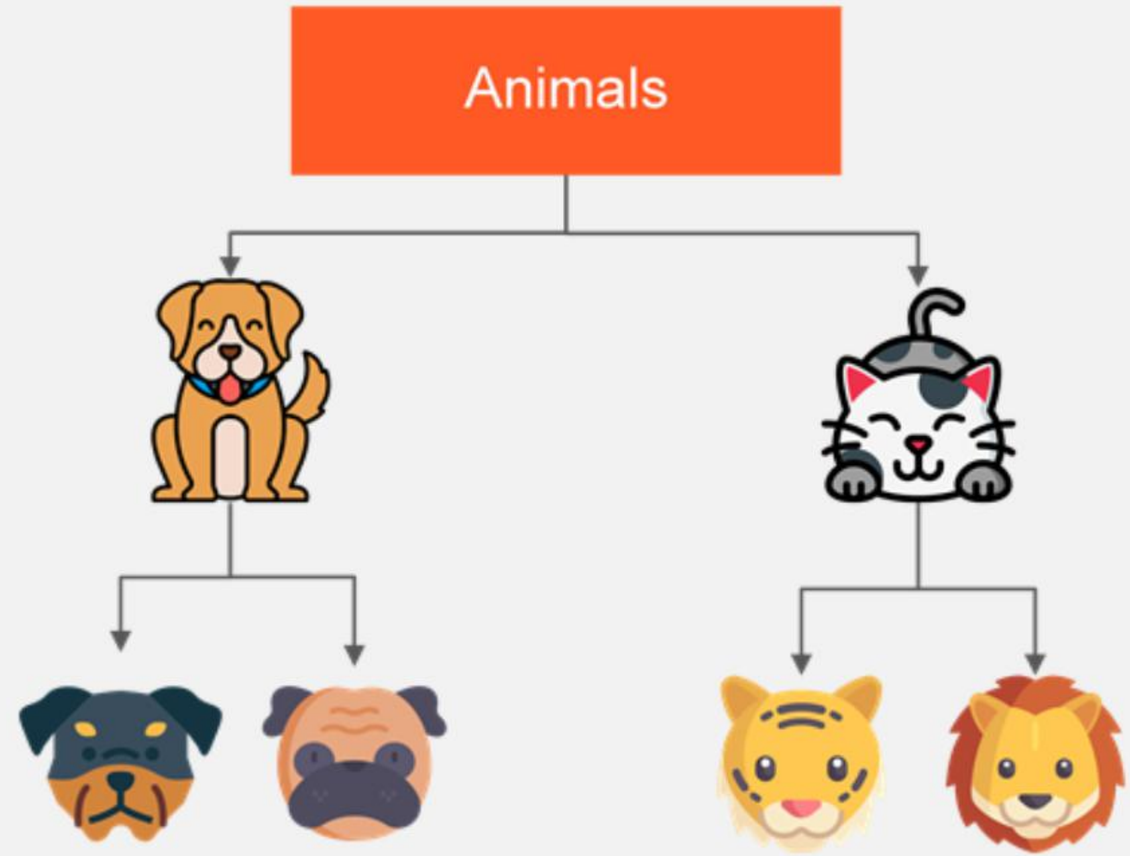
Animal Species

Observe the animal classification.

- How do you think the different species of the dog family are related?
- What features do they share?
- Which feature is unique in each one of them?



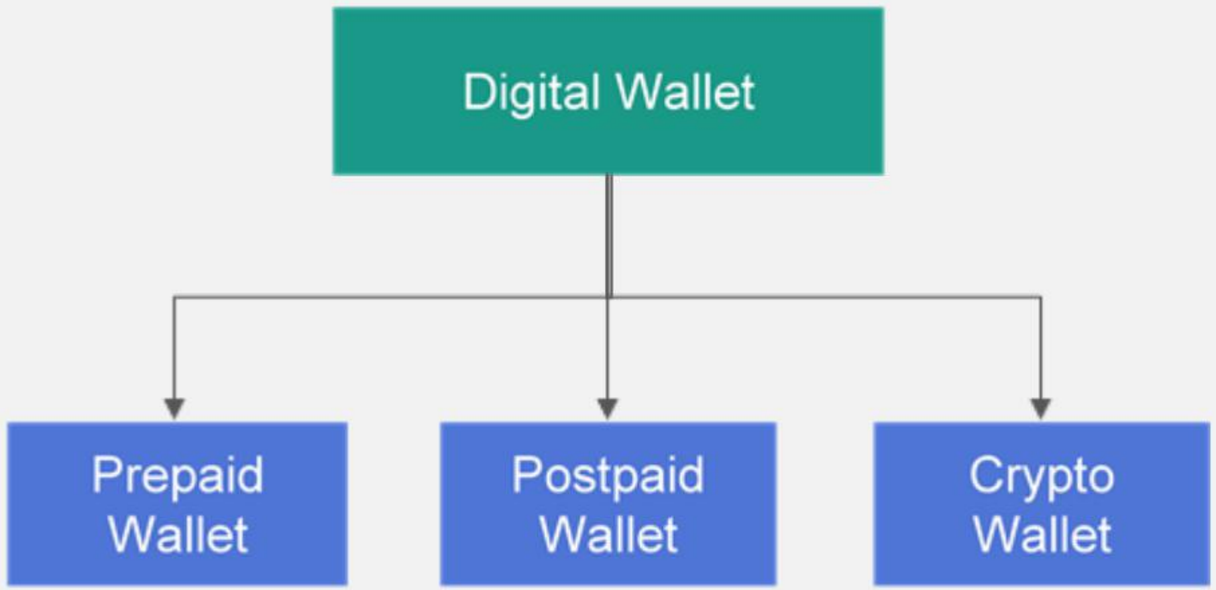
Book Formats

Books come in various formats like paper back, e-books, audio books etc.

- What common characteristics do they have?
- How are they different?

Digital Wallet

- What are the common features among the different types of digital wallets that we use ?



Think and Tell

- Is there a hierarchy in all the examples?
- Are all the elements in the tree structure related?
- Is there a relationship between the animals in the hierarchy?
- Is a dog a type of animal?
- Is an e-book a type of book?
- Is a crypto wallet a type of digital wallet?

In OOP terms, an animal is an object, a book is an object, a digital wallet is an object.

Learning Objectives

- Explore the relationship between objects
- Explain the types of Inheritance
- Implement Inheritance
- Explore Constructors in Inheritance
- Implement Inheritance using abstract classes
- Explore the Object Class

Implement Inheritance

Exploring Relationship Between Objects

Relationship Between Objects – Composition

- Java objects can be related to one another just as in the real world.
- When an object contains another object, it's called a composition.
- The Java object Employee can have an address.

- The has-a relationship is referred to as a Composition relationship between objects.

Implementing Composition Relationship Between Classes

- Create a new class called `Address`.
- An Employee can have an Address.

```java
public class Address {
    int houseNo;
    String streetName;
    String state;
    String city;
    String zipCode;
    String country;

    public Address(int houseNo, String streetName, String state,
                   String city, String zipCode, String country) {
        this.houseNo = houseNo;
        this.streetName = streetName;
        this.state = state;
        this.city = city;
        this.zipCode = zipCode;
        this.country = country;
    }
}
```

```java
public class Employee {
    String employeeName;
    int employeeCode;
    int age;
    String dob;
    double salary;
    Address address;

    public Employee(String employeeName, int employeeCode, int age,
                    String dob, double salary, Address address) {
        this.employeeName = employeeName;
        this.employeeCode = employeeCode;
        this.age = age;
        this.dob = dob;
        this.salary = salary;
        this.address = address;
    }
}
```

The Employee Class

- **Declare the `Address` as an attribute of the `Employee` class.**
- **The `address` is also an instance variable of the class Employee just like `employee Name` or `age`.**
- **The address can be a parameter of the constructor.**

Composition – Object Initialization

- **Steps to initialize the values of the `Employee` object that has a composition relationship with Address:**
  - **Declare an `Address` object and initialize its values by making a call to the `Address` parameterized constructor.**
  - **Pass the `Address` object when initializing the `Employee` object.**

```
Address address = new Address( houseNo: 12, streetName: "Marble Drive",
        state: "Kentucky", city: "Evansville", zipCode: "567-60", country: "USA");

Employee tom = new Employee( employeeName: "Tom", employeeCode: 103, age: 23, dob: "08/09/1999", salary: 2000,address);
```

Employee Details

A startup company wants to manage the personal details of its Employees like name, age, DOB, and address. The address contains the house number, street name, city, country, and zip code.

Tasks

1. Model the `Employee` and `Address` class.
2. Establish the Composition relationship between the `Employee` and `Address` class.

Click here for the solution.

Demonstrate the program on the IntelliJ IDE.

DEMO

Relationship Between Objects – Inheritance

- Java objects can have a hierarchical relationship to one another just like real-world objects.
- In an organization, a Manager or Programmer can be the Employee of an organization.
- A Manager is-a Employee.
- The is-a relationship between objects is called as Inheritance relationship.
- In this case, we can say that the Manager class can be derived from the Employee class.

Quick Check

Which objects listed are in an is-a relationship ?

1. Student – Teacher
2. BMW – Car
3. Bus – Vehicle
4. Laptop – Hard Drive
5. Thermometer – Temperature
6. Pelican – Bird
7. Lion – Animal
8. Television – Mobile
9. Football – Game

Quick Check – Solution

Which objects listed are in an is-a relationship ?

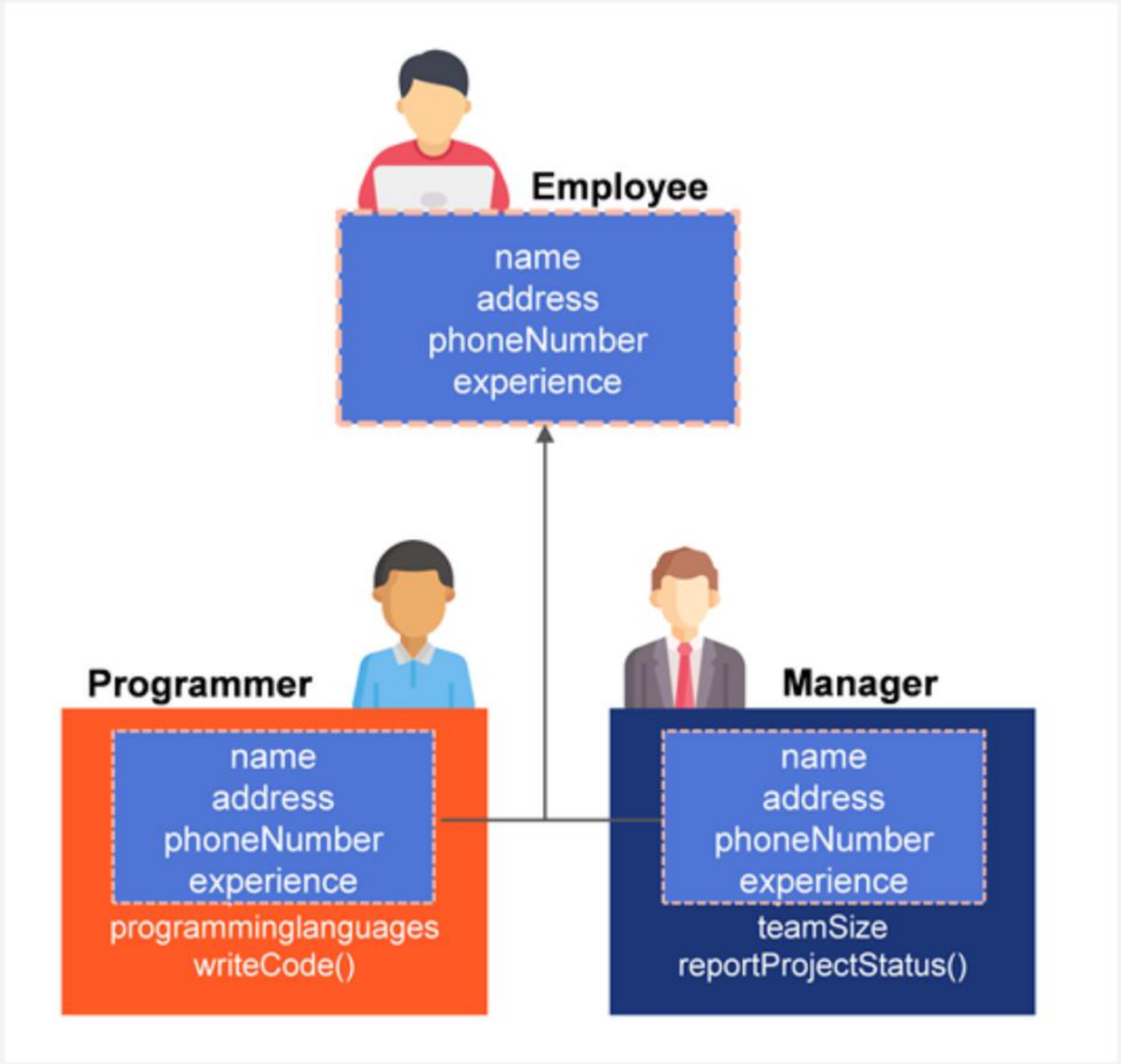
1. Student – Teacher
2. BMW – Car
3. Bus – Vehicle
4. Laptop – Hard Drive
5. Thermometer – Temperature
6. Pelican – Bird
7. Lion – Animal
8. Television – Mobile
9. Football – Game

Common Properties

- The Manager and Programmer are Employees of an organization.
- A few attributes and behaviors of both are shown in the image.
- There are few common attributes for both.
- When building a relationship using inheritance, we can have
  - The common attributes and behaviors can be inside the class called `Employee`.
  - Make the `Programmer` and `Manager` class inherit from `Employee`.


```
Class Programmer {
     String name;
     String address;
     String phoneNumber;
     float experience;
     String[] programminglanguages;
     void writeCode()   {}
}
```


```
Class Manager {
     String name;
     String address;
     String phoneNumber;
     float experience;
     int teamSize;
     void reportProjectStatus()   {}
}
```



Common Properties

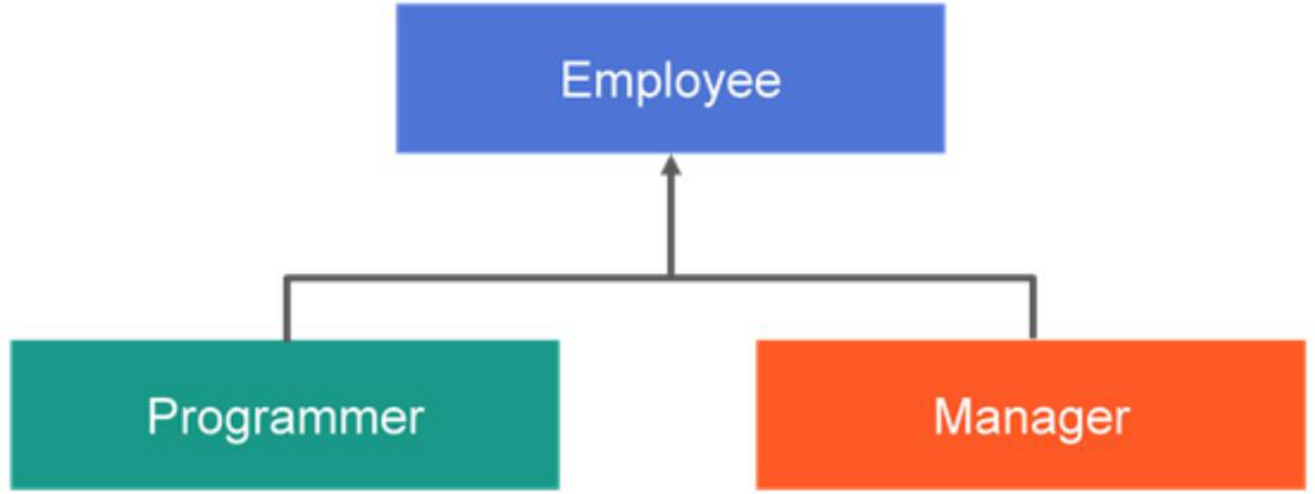
- Identify the common attributes and behavior and build a top class called Employee.
- The Programmer and Manager class inherit the common attributes and behavior from Employee.
- The specific attributes and behavior can be modelled in their respective classes.

The Need for Inheritance

- The common attributes and behavior can be defined in the super class, so that all subclasses will inherit them. Thus, inheritance enables code reusability.
- A sub class can use a piece of code, such as methods implemented by the super class without rewriting the code.
- This allows a sub class to duplicate any public or protected variables, objects, and methods defined in another class for its own use automatically.

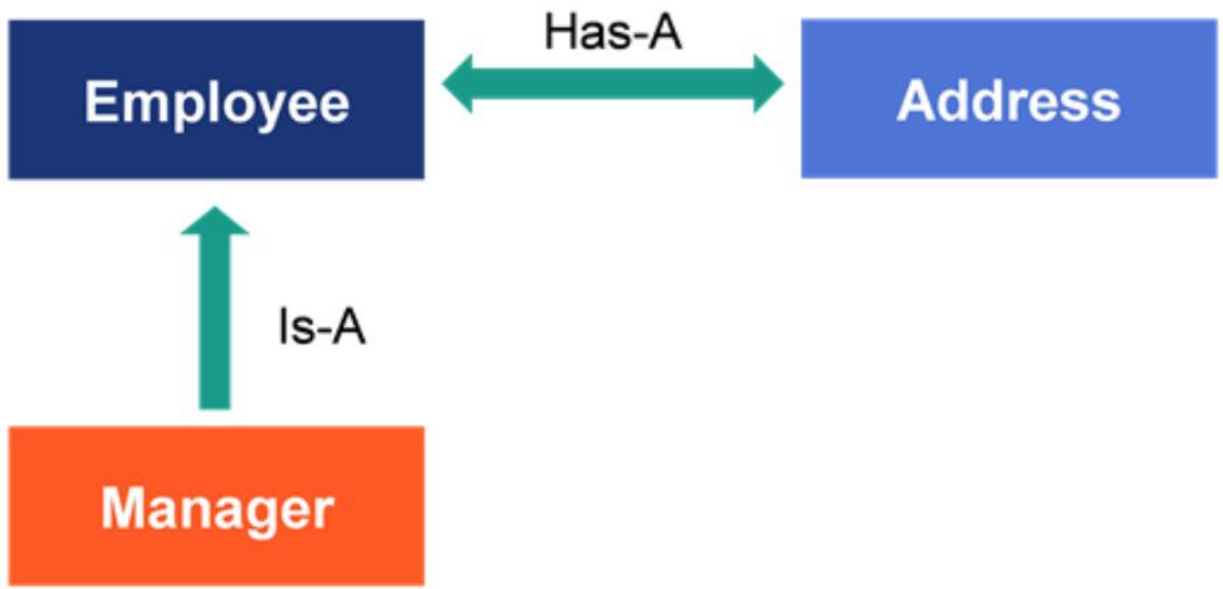
Inheritance Terminology

- The top class in an inheritance hierarchy is called a super class, base class, or parent class.
- The class that derives from the super class is called a sub class, derived class, or child class.
- `Employee` is the super class, `Programmer` and `Manager` are the sub classes.



Composition and Inheritance

- A simple illustration of what composition and inheritance relationships look like.



Explain the Types of Inheritance

Types of Inheritance

- There are many types of inheritance, we will discuss a few of them.

Single level inheritance

Multilevel inheritance

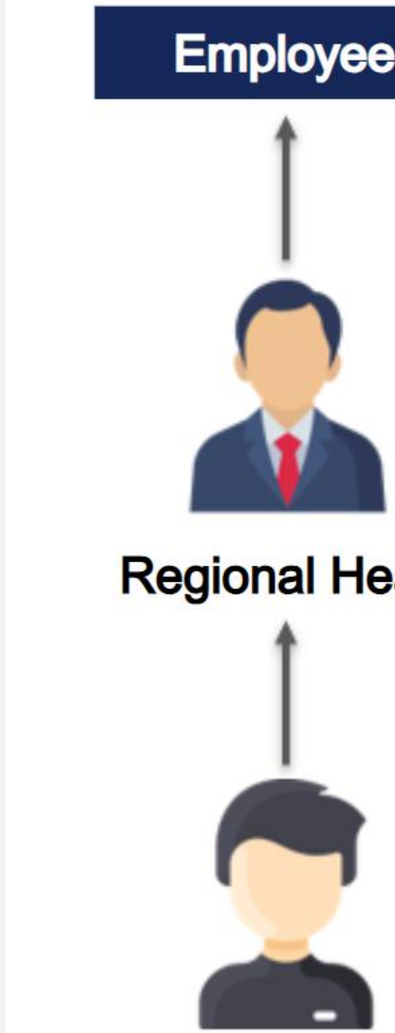
Hierarchical inheritance

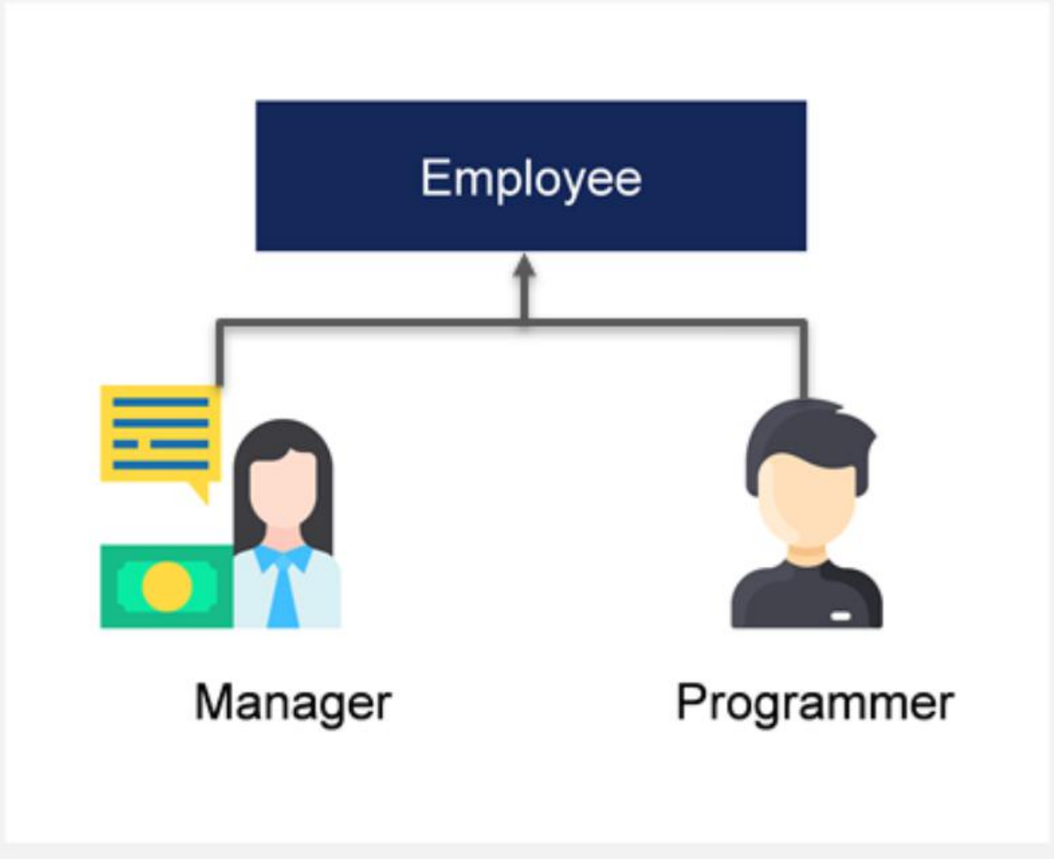
Single Level Inheritance

- When a single subclass derives the functionality of the existing superclass, it is called a single level inheritance.
- The Employee is the super class, and the Manager class is the sub class.

Multilevel Inheritance

- When a subclass inherits the properties of another subclass, it is called a multilevel inheritance.
- The Regional Delivery head class inherits from the Regional Head class which in turn inherits from the Employee class.





Hierarchical Inheritance

- In hierarchical inheritance, one or more subclasses are derived from a single superclass.
- The Employee class is the super class. The Manager and Programmer class derive their properties from the Employee.

Implement Inheritance

Implementing Inheritance

- **Inheritance can be implemented in a Java program using the `extends` keyword.**
- **A subclass that inherits from a super class must be specified by using the `extends` keyword**
  - `<subclass name> extends <superclass name>`
- **If we say that a `Manager` class derives its properties from an `Employee` class, we can represent it as shown below.**

```
class Employee{

}
class Manager extends Employee{

}
```

Implement Inheritance

- The super class Employee consists of the attributes name, employee code, DOB, age and salary.
- The sub class Manager inherits all the properties.
- Any specific property like a manager will have a team size, these can be specified in the sub class Manager.

Super class - Employee

```
public class Employee {
    private String employeeName;
    private String employeeCode;
    private String dob;
    private int age;
    private double salary;

}
```

Sub class - Manager

```
public class Manager extends Employee{
    int teamSize;
    int teamRatings;

}
```

Employee Information

A startup company wants to manage the details of its Employees like name, age, DOB, address, and salary. The Employees of the company have different designations. A Manager will have common attributes and specific attributes like team size and ratings by the team that need to be captured.

Tasks:

1. Model the `Employee` class as the super class
2. Model the `Manager` class as a sub class.
3. Create an implementation class where the object of the `Manager` class can be created, and values assigned.
4. Display the details of the manager.

Click here for the solution.

Demonstrate the program on the IntelliJ IDE

DEMO

Explore Constructors in Inheritance

Constructors in Inheritance

- The constructors cannot be inherited in Java.
- Each class has its own constructors.
- Java does not allow constructors to be inherited since constructing the sub class object might be different from the super class object.
- But Java allows the constructors of the super class to be called in the sub class using the `super` keyword.
- Using the `super` keyword allows the sub class to reuse the initialization code in the constructor for the common attributes shared with the super class.

The `super` Keyword

- The super keyword can be used to:
  - Refer to the immediate parent or super class instance variable in the sub class.
  - Invoke a parent or super class method from the sub class.
  - Invoke the immediate parent or super class constructor from the sub class.

Invoke the Constructor of the `super` Class

- `Employee` **is the super class and consists of all the common attributes of the sub class** `Manager`**.**
- **The** `Manager` **class calls the** `Employee` **class constructor using the** `super` **keyword.**
- **The common attributes are passed to the constructor call and the variables are initialized.**

Note: The parameters in the call using super in the sub class must match the parameters in the super class constructor.

```java
public class Employee {
    private String employeeName;
    private String employeeCode;
    private String dob;
    private int age;
    private double salary;

    public Employee(String employeeName,
                    String employeeCode, String dob,
                    int age, double salary) {
        this.employeeName = employeeName;
        this.employeeCode = employeeCode;
        this.dob = dob;
        this.age = age;
        this.salary = salary;
    }
```

```java
public class Manager extends Employee{
    private int teamSize;
    private int teamRatings;

    public Manager(String employeeName, String employeeCode,
                   String dob, int age, double salary,
                   int teamSize, int teamRatings) {
        super(employeeName, employeeCode, dob, age, salary);
        this.teamSize = teamSize;
        this.teamRatings = teamRatings;
    }
```

Quick Check

The `super` keyword is used to _____.

1. invoke the sub class constructor from the super class
2. invoke the super class constructor from the sub class
3. call the sub class variables in the super class
4. call the sub class methods in the super class

Quick Check – Solution

The `super` keyword is used to _____.

1. invoke the sub class constructor from the super class
2. invoke the super class constructor from the sub class
3. call the sub class variables in the super class
4. call the sub class methods in the super class

Employee Information

The Employees of a company has different designations based on which they will get a yearly increment. A Manager will have common attributes and specific attributes like team size and ratings by the team that are crucial to the yearly increment. A Regional Head will have common attributes of the Employee and specific attributes like region name, region performance, review ratings.

Tasks:

1. Use constructors to initialize common attributes in the super class and sub classes.
2. Model the calculate increment behavior of the employee.

DEMO

Employee Information (contd.)

3. The conditions for yearly increment:
   A manager gets a 12.5 % hike if the ratings by team is > 8
   A regional head gets a 10.5% hike if the region performance is >8 and review ratings > 7
4. Use the super keyword in the constructor and to call the variables and methods of the super class.

Click here for the solution.
Demonstrate the program on the IntelliJ IDE

DEMO

Implement Inheritance Using Abstract Classes

Think and Tell

- The salary of the employee of an organization is captured in the `Employee` class.
- An increment must be calculated for all employees of the company.
- The `calculateIncrement` method is implemented in the `Manager` and `RegionalHead` class.
- If the programmer of the application forgets to implement `calculateIncrement` method in the `Manager` class. The managers in the company may not get a pay hike.
- How can such situations be avoided?
- Can we give what needs to be implemented by the sub classes?

What Is Abstraction?

- Abstraction in OOP is the property in which only the functionality of a class or method is made visible to a user, however, the implementation is hidden.
- In the super class, the method definition can be made, and the sub class can implement it as per the requirement.
- This allows the implementation of a class or method to change without affecting the way it is called or used.
- It reduces the efforts and programming complexity involved in developing a Java application.
- There are two ways to apply abstraction in Java:
  - By using abstract classes
  - By using interfaces

Abstract Class

- An abstract class is a class that is declared abstract.
- Abstract classes cannot be instantiated.
  - For example - An object of the abstract class cannot be created, but they can be sub classed.
- It is declared using the `abstract` keyword.
- This class can have abstract methods (methods without body) as well as concrete methods (methods with body).
- Other classes can inherit an abstract class; however, any abstract methods in the class must be explicitly implemented by the derived class.

Implementing Inheritance Using Abstract Classes

- **The super class** `Employee` **can be declared as abstract with an abstract method** `calculateIncrement()`.

```
public abstract class Employee {
    private String employeeName;
    private String employeeCode;
    private String dob;
    private int age;
    private double salary;
    public abstract double calculateIncrement(float hikePercentage);
}
```

- **Any class that inherits from the** `Employee` **class must implement the** `calculateIncrement` **method.**

Rules to Write Abstract Classes

- An abstract class can contain:
  - Implemented methods
  - Constructors with common attributes of the sub classes
  - An abstract method

Note: Having an abstract method in an abstract class is optional.

```java
public abstract class Employee {
    String employeeCode;
    String employeeName;

    public Employee(String employeeCode, String employeeName, double salary) {
        this.employeeCode = employeeCode;
        this.employeeName = employeeName;
        this.salary = salary;
    }

    double salary;

    void displayDetails(){
        System.out.println(employeeCode);
        System.out.println(employeeName);
        System.out.println(salary);
    }

    public abstract double calculateIncrement(float hikePercentage);
}
```

Constructor

Implemented method

Abstract method

Quick Check

An abstract class must contain an abstract method.

1. True

2. False

Quick Check – Solution

An abstract class must contain an abstract method.

1. True
2. False

Employee Information – Abstract Class

The Employees of a company has different designations based on which they will get a yearly increment. A Manager will have common attributes and specific attributes like team size and ratings by the team that are crucial to the yearly increment. A Regional Head will have common attributes of the Employee and specific attributes like region name, region performance, review ratings.

DEMO

Employee Information – Abstract Class (contd.)

Tasks:

1. **Declare the super class `Employee` as abstract, define an abstract method `calculateIncrement`.**
2. **Use constructors to initialize common attributes in the super class and sub classes.**
3. **Model the calculate increment behavior of the employee.**
4. **The conditions for yearly increment:**
   A manager gets a 12.5 % hike if the ratings by team is > 8
   A regional head gets a 10.5% hike if the region performance is >8 and review ratings > 7
5. **Use the super keyword in the constructor.**

Click here for the solution.

Demonstrate the program on the IntelliJ IDE

DEMO

Explore the Object Class

The Object Class

- The `Object` class of the `java.lang` package is the parent class for all the classes in Java.
- It is the topmost class in the class hierarchy.
- All classes implicitly inherit the Object class.
- The following methods of the Object class that are used extensively:
  - `protected Object clone() throws CloneNotSupportedException`
  - `protected void finalize() throws Throwable`
  - `public final Class getClass()`
  - `public String toString()`
  - `public boolean equals(Object obj)`
  - `public int hashCode()`

The `toString()` Method

- The `toString` method is used to return a string representation of an object.
- If any object is printed as shown below, the `toString()` method is internally invoked by the Java compiler.

```
Author author = new Author();
System.out.println(author);
```

- The `toString` method can be overridden to give our own implementation in the `Author` class as below,

```
@Override
public String toString() {
    return "Author{" +
            "authorName='" + authorName + '\'' +
            ", authorPenName='" + authorPenName + '\'' +
            '}';
}
```

- Note: More on overriding will be discussed in later Sprints.