# Ex1

|  |  |
| --- | --- |
| Write a JAVA program to demostrate Polymorphism property in OOP  Ans1:  Sure, here's a simple Java program that demonstrates the concept of polymorphism using inheritance and method overriding:  class Animal {      void makeSound() {          System.out.println("Animal makes a sound");      }  }  class Dog extends Animal {      @Override      void makeSound() {          System.out.println("Dog barks");      }  }  class Cat extends Animal {      @Override      void makeSound() {          System.out.println("Cat meows");      }  }  public class PolymorphismDemo {      public static void main(String[] args) {          Animal animal1 = new Dog();          Animal animal2 = new Cat();          animal1.makeSound(); // Output: Dog barks          animal2.makeSound(); // Output: Cat meows      }  }  In this program:   1. We have a base class **Animal** with a method **makeSound()**. 2. We have two derived classes **Dog** and **Cat**, both of which override the **makeSound()** method to provide their own implementation. 3. In the **PolymorphismDemo** class, we create instances of **Dog** and **Cat**, but we store them in reference variables of type **Animal**. 4. When we call the **makeSound()** method on these references, the appropriate overridden method from the subclass is invoked, demonstrating runtime polymorphism.   The concept of polymorphism here is that even though we are using references of the base class type (**Animal**), the actual method that gets executed depends on the type of object that the reference is pointing to (**Dog** or **Cat**). This allows for more flexibility and extensibility in the code by allowing different subclasses to provide their own implementations of methods from the base class. | Viết chương trình JAVA để minh họa thuộc tính Đa hình trong OOP  Ans1:  Chắc chắn rồi, đây là một chương trình Java đơn giản thể hiện khái niệm đa hình bằng cách sử dụng tính kế thừa và ghi đè phương thức:    Trong chương trình này:   1. Chúng tôi có một lớp cơ sở **Animal**với một phương thức **makeSound()**. 2. Chúng ta có hai lớp dẫn xuất **Dog**và **Cat**, cả hai đều ghi đè **makeSound()**phương thức để cung cấp cách triển khai của riêng chúng. 3. Trong **PolymorphismDemo**lớp, chúng ta tạo các thể hiện của **Dog**và **Cat**, nhưng chúng ta lưu trữ chúng trong các biến tham chiếu kiểu **Animal**. 4. Khi chúng ta gọi **makeSound()**phương thức trên các tham chiếu này, phương thức được ghi đè thích hợp từ lớp con sẽ được gọi, thể hiện tính đa hình trong thời gian chạy.   Khái niệm về tính đa hình ở đây là mặc dù chúng ta đang sử dụng các tham chiếu của loại lớp cơ sở ( **Animal**), nhưng phương thức thực sự được thực thi phụ thuộc vào loại đối tượng mà tham chiếu đang trỏ tới ( **Dog**hoặc **Cat**). Điều này cho phép mã linh hoạt hơn và khả năng mở rộng bằng cách cho phép các lớp con khác nhau cung cấp các triển khai phương thức của riêng chúng từ lớp cơ sở. |