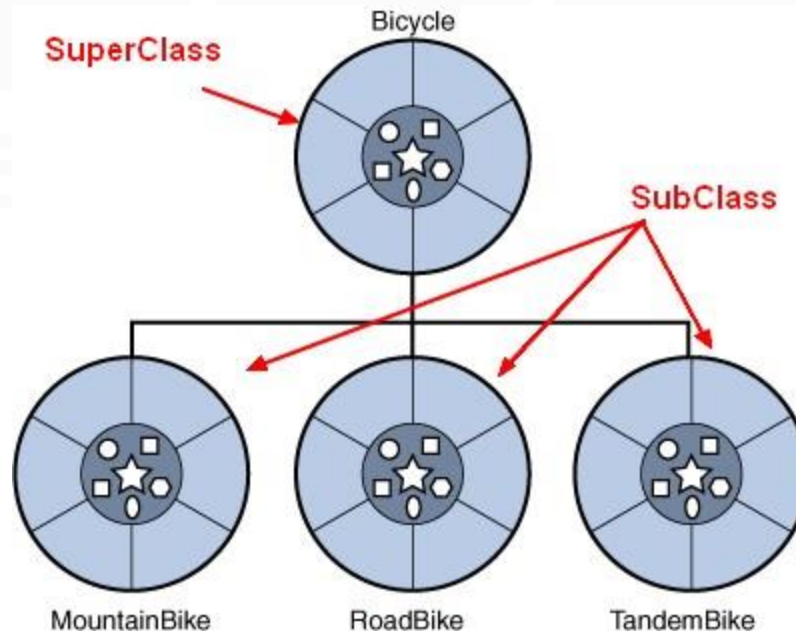


Bài 10

Kế thừa và đa hình

- Giới thiệu về kế thừa
- Phương thức overriding
- Đa hình
- Từ khóa Abstract



- Trong thế giới thực có nhiều loài động thực vật có nhiều **đặc điểm tương tự**.
- Chúng có thể **phân loại** theo nhóm như ăn cỏ, ăn thịt hoặc ăn tạp...
- Theo thuyết tiến hóa thì chúng có **sự kế thừa**, phát triển hoặc triệt tiêu. Và, theo cách phân loại trên chúng được **nhóm và phân loại** theo các lớp con.

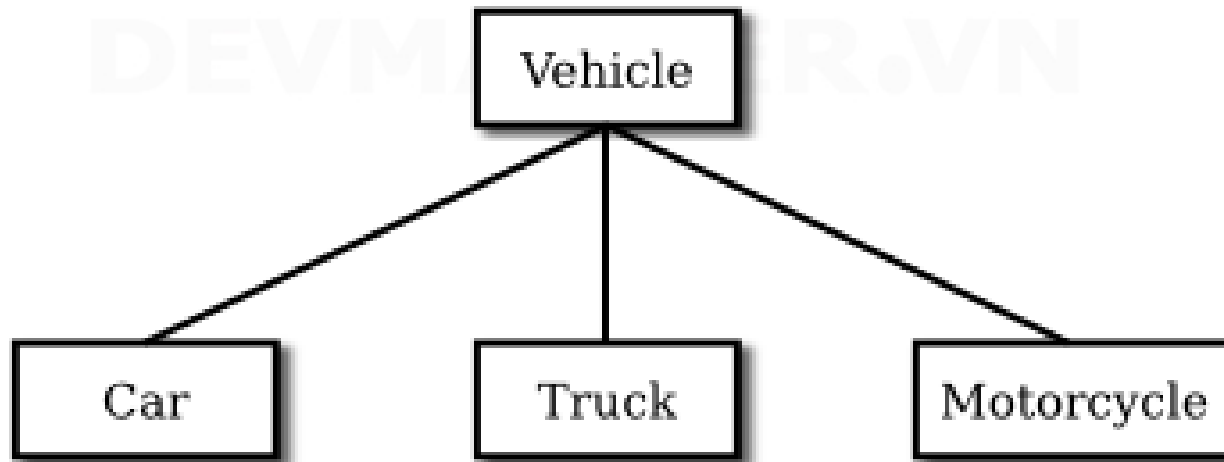
- Tương tự như vậy, Java cung cấp khái niệm về kế thừa để tạo **lớp con** từ lớp cụ thể.
- Ngoài ra, như con tắc kè có thể thay đổi màu sắc tùy môi trường, con người cũng có vai trò sống khác nhau như: cha, con, anh, em, bạn....
- Điều này có nghĩa là hành xử khác trong môi trường khác. Đây chính là khái niệm **đa hình** trong Java

Trong cuộc sống hàng ngày ta thường gặp những đối tượng có một hoặc nhiều mối liên hệ với nhau.

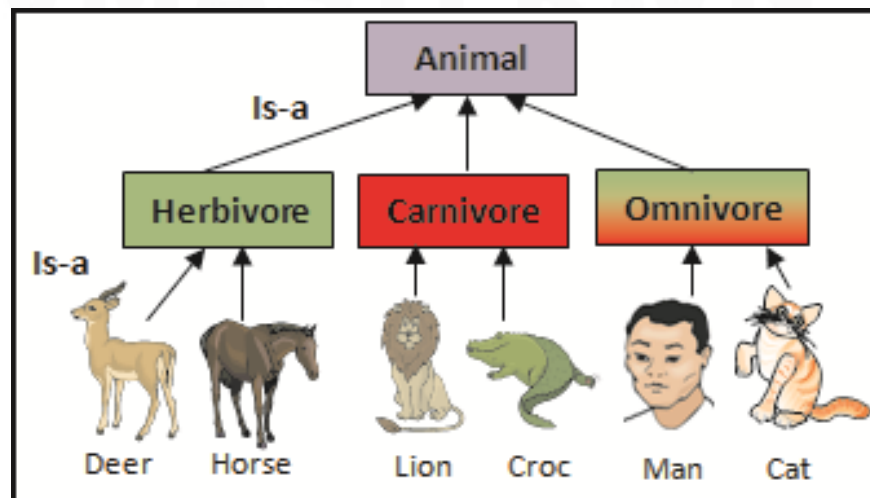
Ví dụ, ô tô có 4 bánh, 4 bánh là phương tiện, phương tiện là máy móc.

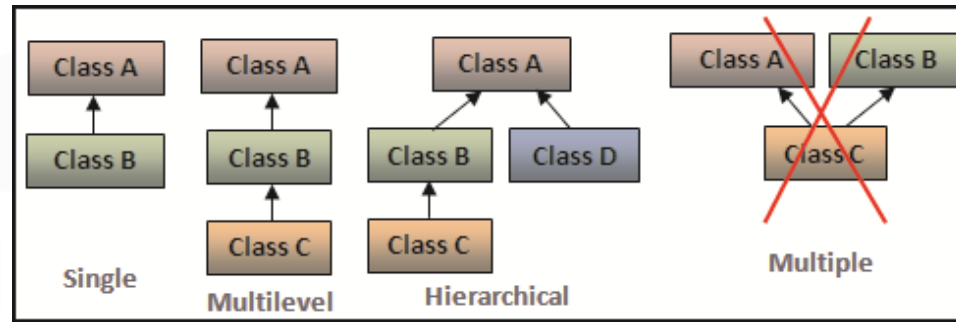
Tương tự vậy, nhiều đối tượng có thể xác định mối quan hệ với nhau. Tất cả chúng có điểm chung nào đó.

Ví dụ các phương tiện cần phải có BKS, số bánh, dung tích xi-lanh, nhiên liệu tiêu thụ....



- Hình vẽ dưới thể hiện mối quan hệ **thừa kế**:
- Nai là một động vật ăn cỏ, động vật ăn cỏ là một con vật.
- Các thuộc tính chung của lớp động vật ăn cỏ có thể lưu trữ ở **lớp động vật ăn cỏ**.
- Tương tự vậy, thuộc tính chung của thú có thể lưu ở **lớp thú vật**.





- Lớp chỉ kế thừa từ một và chỉ một lớp cha.

Single Inheritance

- Lớp con có thể kế thừa từ lớp mà lớp đó lại là con của lớp khác.

Multilevel Inheritance

- Lớp cha có thể có nhiều con ở các cấp độ khác nhau.

Hierarchical Inheritance

- Lớp con có nguồn gốc từ hơn một lớp cha.

Multiple Inheritance

Bên trong lớp con, ta có thể tác động tới các thành phần thừa kế như là ẩn, thay thế, tăng cường với các thành phần mới như:

Các trường và phương thức có thể gọi trực tiếp.

Có thể khai báo trường với tên trùng với tên ở lớp cha, điều này dẫn tới ẩn đi trường ở lớp cha khi sử dụng đối tượng tạo từ lớp con.

Có thể khai báo trường mới ở lớp con, thành phần mới này là mở rộng và không có ở lớp cha.

Có thể viết phương thức trùng tên với phương thức ở lớp cha, điều này gọi là overriding (ghi đè)

Một phương thức `static` mới có thể khai báo ở lớp con trùng tên với phương thức ở lớp cha. Điều này dẫn tới phủ định phương thức ở lớp cha.

Một phương thức khai báo ở lớp con đương nhiên không thể truy cập từ lớp cha.

Constructor của lớp con có thể gọi constructor của lớp cha, hoặc ngầm bằng cách dùng từ khóa `super`.

Từ khóa `extends` được sử dụng để tạo lớp con. Một lớp chỉ có thể kế thừa trực tiếp từ một lớp.

Syntax

```
public class <class1-name> extends <class2-name>
{
  ...
  ...
}
```

Ví dụ: Tạo class **Vehicle**

```
package session10;
public class Vehicle {

    // Declare common attributes of a vehicle
    protected String vehicleNo; // Variable to store vehicle number
    protected String vehicleName; // Variable to store vehicle name
    protected int wheels; // Variable to store number of wheels
    /**
     * Accelerates the vehicle
     *
     * @return void
     */
    public void accelerate(int speed) {
        System.out.println("Accelerating at:" + speed + " kmph");
    }
}
```

Tạo class **FourWheeler** kế thừa **Vehicle**:

```
package session10;
class FourWheeler extends Vehicle{

    // Declare a field specific to child class
    private boolean powerSteer; // Variable to store steering information

    public FourWheeler(String vId, String vName, int numWheels, boolean pSteer){

        // Attributes inherited from parent class
        vehicleNo = vId;
        vehicleName = vName;
        wheels = numWheels;

        // Child class' own attribute
        powerSteer = pSteer;
    }
}
```

Tạo class **FourWheeler** kế thừa **Vehicle**:

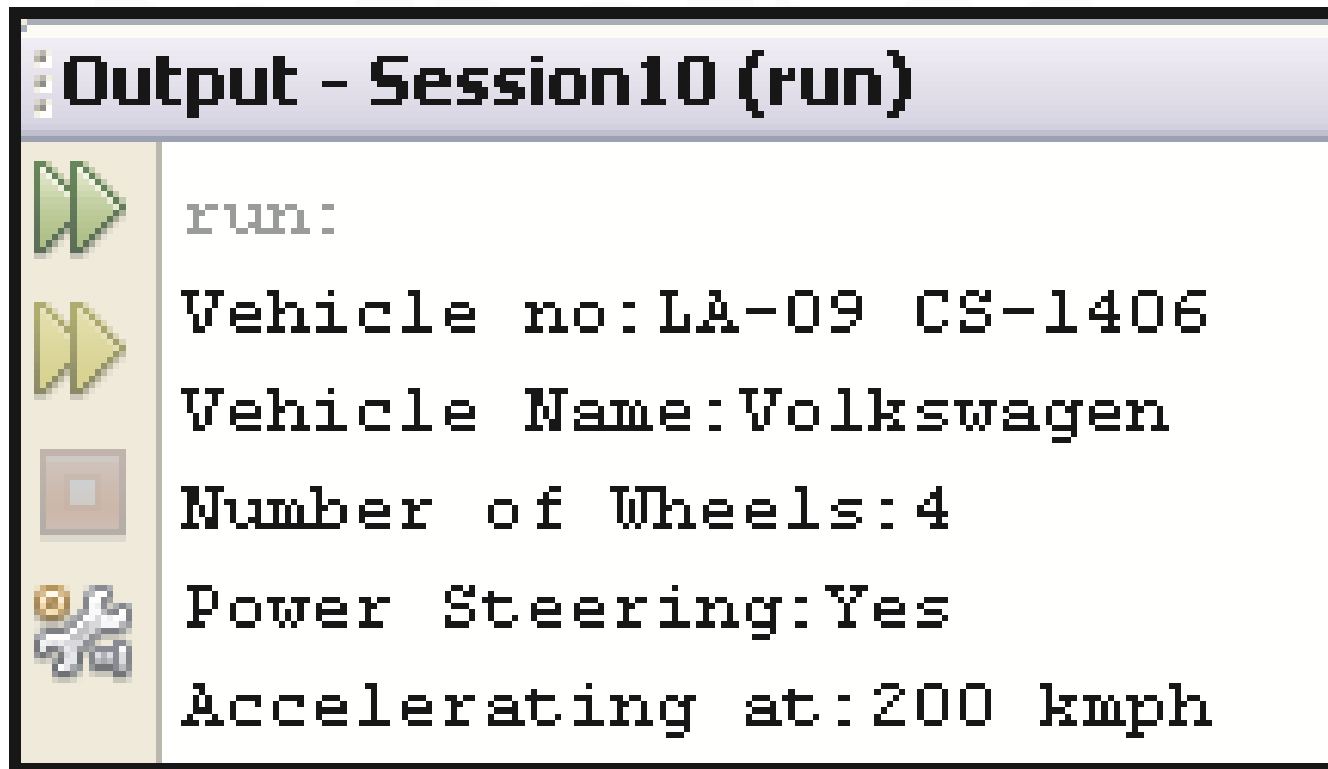
```
public void showDetails() {  
  
    System.out.println("Vehicle no:" + vehicleNo);  
    System.out.println("Vehicle Name:" + vehicleName);  
    System.out.println("Number of Wheels:" + wheels);  
  
    if(powerSteer == true)  
        System.out.println("Power Steering:Yes");  
    else  
        System.out.println("Power Steering:No");  
    }  
}  
  
/**  
 * Define the TestVehicle.java class  
 */  
public class TestVehicle {
```

Tạo class **FourWheeler** kế thừa **Vehicle**:

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {

    // Create an object of child class and specify the values
    FourWheeler objFour = new FourWheeler("LA-09 CS-1406", "Volkswagen",
    4, true);
    objFour.showDetails(); // Invoke the child class method
    objFour.accelerate(200); // Invoke the inherited method
}
}
```

Chạy chương trình:



The image shows a screenshot of an IDE's output window. The window has a title bar that reads "Output - Session10 (run)". On the left side of the window, there is a vertical toolbar with four icons: a green double arrow pointing right, a yellow double arrow pointing right, a brown square, and a silver wrench and screwdriver. The main area of the window contains the following text in a monospaced font:

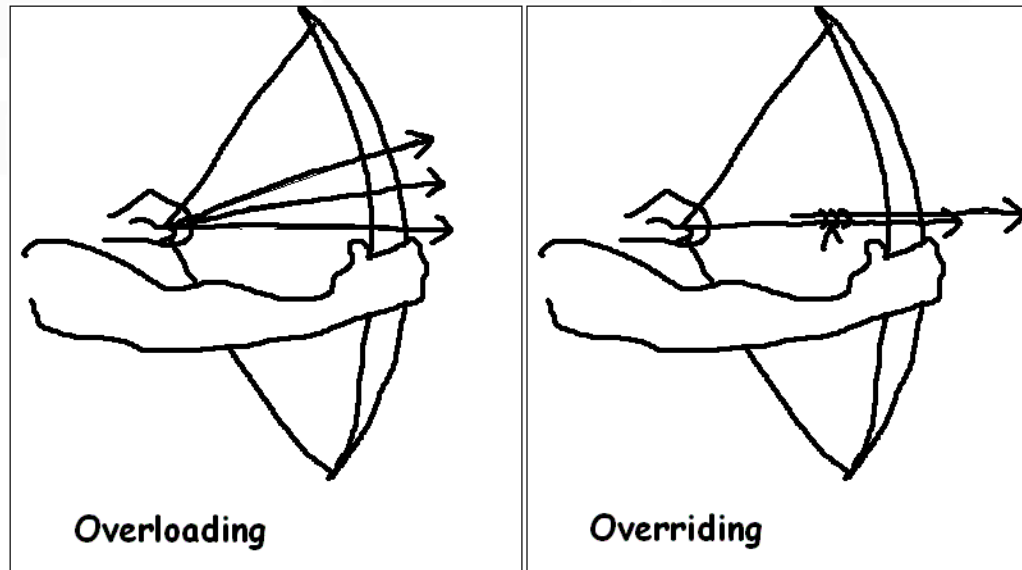
```
run:  
Vehicle no:LA-09 CS-1406  
Vehicle Name:Volkswagen  
Number of Wheels:4  
Power Steering:Yes  
Accelerating at:200 kmph
```

- Java cho phép tạo phương thức có **tên giống** với phương thức ở lớp cha (cả kiểu trả về).
- Điều này được gọi là **ghi đè**.
- **Phương thức ghi đè** cho phép một lớp kế thừa có thể thay đổi hành vi từ lớp cha khi cần thiết.
- **Quy tắc** ghi đè:

Phương thức ghi đè cần trùng tên, kiểu, số lượng tham số cũng như kiểu dữ liệu trả về tương tự với phương thức ở lớp cha nó.

Phương thức ghi đè phải có mô tả truy cập lớn hơn hoặc bằng so với phương thức ở cha của nó.

- Ghi chú **@override** được đặt ở đầu phương thức.
- Chú thích này cung cấp thêm thông tin chương trình, và không ảnh hưởng đến hoạt động của mã.
- **Overriding** khác với **Overloading**.



Đa hình - polymorph là sự kết hợp của 2 từ cụ thể là “poly” có nghĩa là nhiều và “morph” có nghĩa là hình thức.

Vì vậy, đa hình là đề cập đến đối tượng cụ thể có nhiều hình thức khác nhau

Nguyên tắc này có thể áp dụng cho các lớp con của một lớp, chúng có hành vi cụ thể của mình đồng thời cũng có chức năng tương tự ở lớp cha.

Khái niệm ghi đè là một ví dụ về đa hình trong lập trình hướng đối tượng, trong đó cùng một phương thức nhưng ứng xử khác nhau ở lớp con và lớp cha.

Ví dụ

```
class Bike{  
    void run(){System.out.println("running");}  
}  
class Splender extends Bike{  
    void run(){System.out.println("running safely with 60km");}  
  
    public static void main(String args[]){  
        Bike b = new Splender();//upcasting  
        b.run(),  
    }  
}
```

Lớp CHA

Lớp CON

Khởi tạo đối tượng kiểu lớp CHA từ constructor lớp con

Test it Now

Output:running safely with 60km.

Java cung cấp từ khóa `abstract` để tạo lớp cha như là một hình thức tổng quát và sẽ được thừa hưởng ở tất cả lớp con của nó.

Các phương thức đó của lớp cha như là ràng buộc hoặc tiêu chuẩn mà các lớp con có thể thực hiện theo cách riêng của nó.

Abstract method

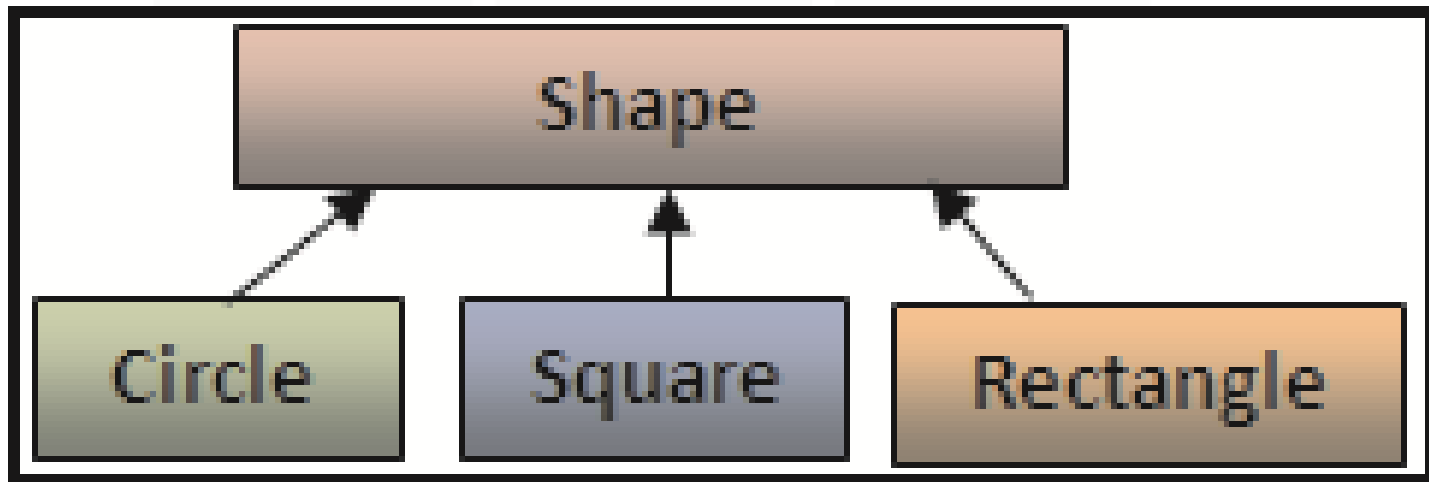
Một phương thức `abstract` được khai báo với từ khóa `abstract` và không có phần thực thi, cũng như không có thân.

Syntax

```
abstract <return-type> <method-name> (<parameter-list>);
```

Abstract class

Một lớp abstract có thể chứa phương thức abstract.



Ví dụ: Tạo lớp **Shape**

```
package session10;  
abstract class Shape {  
    private final float PI = 3.14F; // Variable to store value of PPI  
  
    public float getPI(){  
        return PI;  
    }  
  
    abstract void calculate(float val);  
}
```

Tạo lớp **Circle** kế thừa **Shape** và override **calculate()**

```
package session10;
/**
 * Define the child class Circle.java
 */
class Circle extends Shape{
    float area; // Variable to store area of a circle

    /**
     * Implement the abstract method to calculate area of circle
     *
     * @param rad a float variable storing value of radius
     * @return void
     */
    @Override
    void calculate(float rad){
        area = getPI() * rad * rad;
        System.out.println("Area of circle is:" + area);
    }
}
```

Tạo lớp **Rectangle** kế thừa **Shape** và override **calculate()**

```
/**
 * Define the child class Rectangle.java
 */
class Rectangle extends Shape{

    float perimeter; // Variable to store perimeter value
    float length=10; // Variable to store length

    @Override
    void calculate(float width){

        perimeter = 2 * (length+width);
        System.out.println("Perimeter of the Rectangle is:"+ perimeter);
    }
}
```

Tạo lớp **Calculator**

```
package session10;
public class Calculator {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        Shape objShape; // Declare the Shape object
        String shape; // Variable to store the type of shape

        if(args.length==2) { // Check the number of command line arguments
            //Retrieve the value of shape from args[0]
            shape = args[0].toLowerCase(); // converting to lower case
            switch(shape){

                // Assign reference to Shape object as per user input
                case "circle": objShape = new Circle();
                objShape.calculate(Float.parseFloat(args[1]));
                break;
```


Tạo lớp **Calculator**

```
        case "rectangle": objShape = new Rectangle();
        objShape.calculate(Float.parseFloat(args[1]));
        break;
    }
}
else{
    // Error message to be displayed when arguments are not supplied
    System.out.println("Usage: java Calculator <shape-name> <value>");
}
}
```

- ✓ **Thừa kế** là một tính năng trong Java cho phép các lớp có thể kế thừa từ một lớp khác.
- ✓ Một phương thức ở lớp con có khai báo tương tự, cùng kiểu trả về như phương thức ở lớp cha gọi là **ghi đè**.
- ✓ **Đa hình** đề cập đến một đối tượng có thể có **nhiều hình thức** khác nhau.
- ✓ Một phương thức **trừu tượng** là phương thức khai báo với từ khóa **abstract**, không có phần thực hiện, không có thân.
- ✓ Một lớp trừu tượng chứa phương thức trừu tượng.
- ✓ Trừu tượng – abstract cung cấp **khuôn khổ hành vi** được xác định trước và có thể được sửa đổi ở lớp kế thừa.



Thank for watching!

