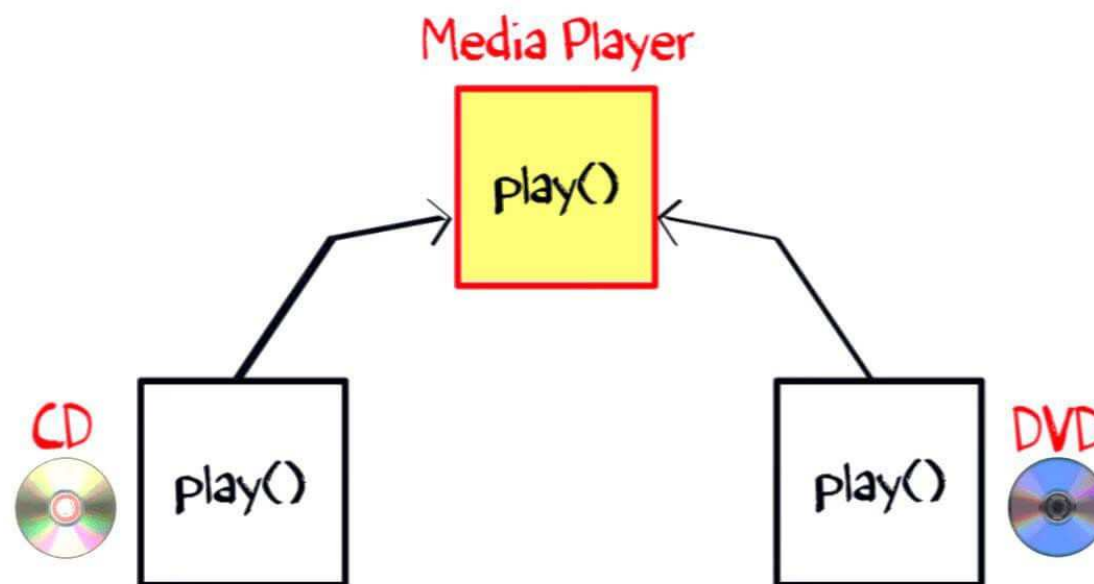


# **Bài 9**

## **Interface và lớp**

# Mục tiêu

- Mô tả về Interface
- Mục đích và cách sử dụng Interface
- Abstraction vs Interface
- Lớp lồng (nested)
- Lớp ẩn danh (anonymous)



Interface – Nested Class

# Giới thiệu

---

- **Đa kế thừa** không được hỗ trợ trong Java.
- Tuy nhiên, có một số trường hợp bắt buộc đối tượng cần **thừa kế** thuộc tính từ **nhiều lớp** để tránh **dư thừa** và **mã nguồn phức tạp**.
- Để giải quyết điều này, Java cung cấp một giải pháp dưới hình thức là **Interface**.
- Ngoài ra, Java cũng cung cấp khái niệm về **lớp lồng** nhau để một số loại chương trình dễ **quản lý**, **an toàn** hơn, **ít phức tạp** hơn.

# Giới thiệu

---

Interface trong Java là một **ràng buộc các quy định tiêu chuẩn** để theo đó phải được thực hiện ở phần tử thực hiện nó.

Một lớp nếu chấp nhận **ràng buộc** này thì buộc phải **thực hiện**.

# Giới thiệu

---

Lớp và Interface có điểm giống nhau ở:

Interface có thể có nhiều phương thức.

Một interface lưu trữ trong file có phần mở rộng là .java và có tên phải trùng với tên file tương tự như Java class.

Mã bytecode của interface được lưu trữ trong .class file.

Interfaces được lưu trữ trong packages và bytecode file lưu trữ trong thư mục có cấu trúc khớp với tên package.

# Giới thiệu

**Interface** và **class** khác nhau như sau:



# Mục đích

---

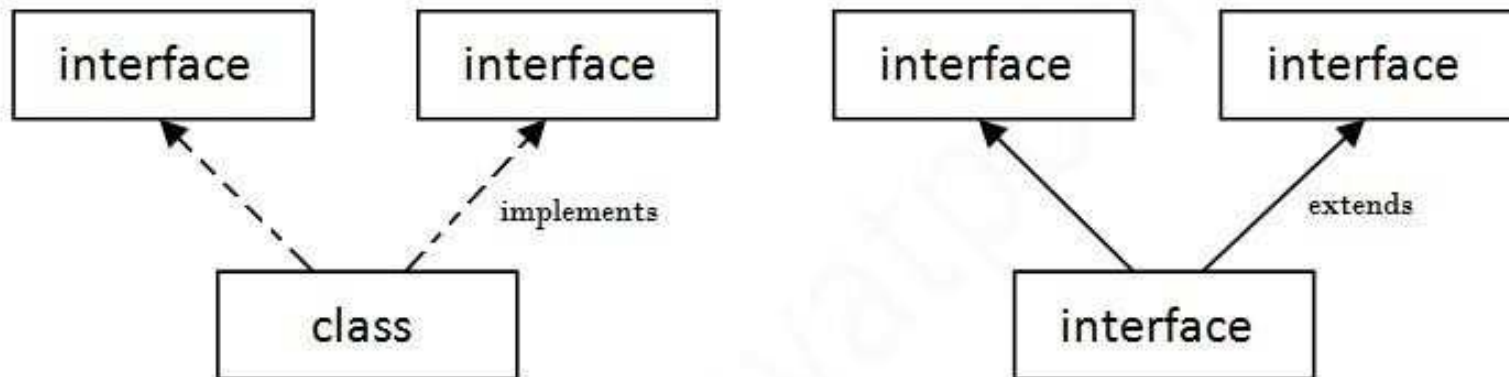
Mục đích của Interface là:

- Đưa ra **"ràng buộc"** cách phần mềm sẽ tương tác cho các nhóm lập trình viên khác nhau.
- Mỗi nhóm có thể **tự do phát triển** mã của mình mà **không cần phải hiểu** về mã của nhóm khác.
- Interface không thuộc về **hệ thống phân cấp** lớp mặc dù nó có tương tác với lớp.
- Giải quyết vấn đề **đa kế thừa** trong lập trình OOP.
- Kế thừa chỉ có thể từ **một** lớp nhưng thực thi (implement) lại từ **nhều** Interface.

# Sử dụng

## Syntax

```
<visibility> interface <interface-name> extends <other-interfaces, ... >
{
// declare constants
// declare abstract methods
}
```

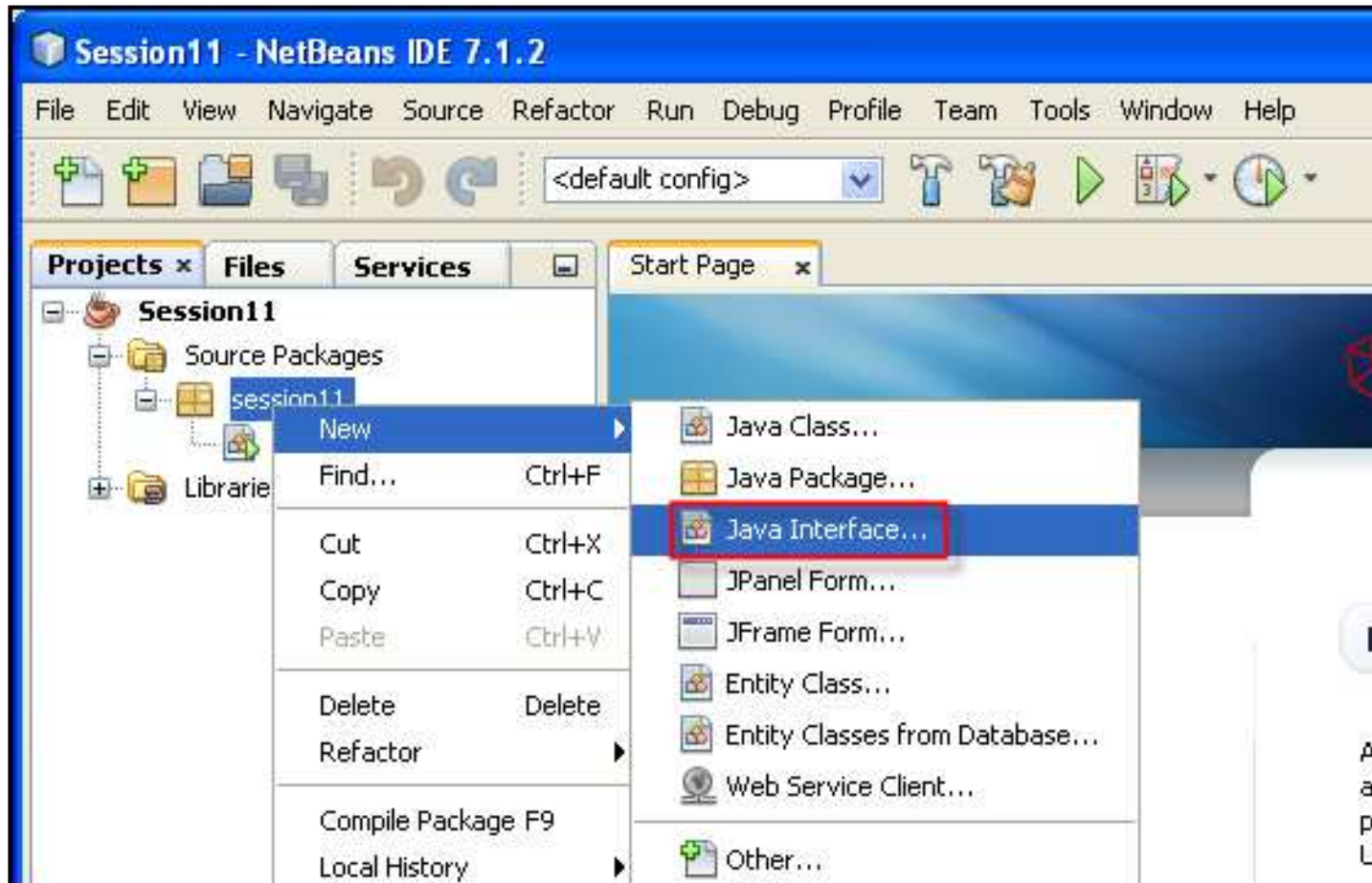


**Multiple Inheritance in Java**



# Sử dụng

Tạo interface **IVehicle**:



Interface – Nested Class

# Sử dụng

## Tạo interface **IVehicle**:

```
package session11;
public interface IVehicle {

    // Declare and initialize constant
    static final String STATEID="LA-09"; // variable to store state ID

    /**
     * Abstract method to start a vehicle
     * @return void
     */
    public void start();

    /**
     * Abstract method to accelerate a vehicle
     * @param speed an integer variable storing speed
     * @return void
     */
    public void accelerate(int speed);
}
```

# Sử dụng

---

Tạo interface **IVehicle**:

```
/**
 * Abstract method to apply a brake
 * @return void
 */
public void brake();

/**
 * Abstract method to stop a vehicle
 * @return void
 */
public void stop();
}
```

# Sử dụng

---

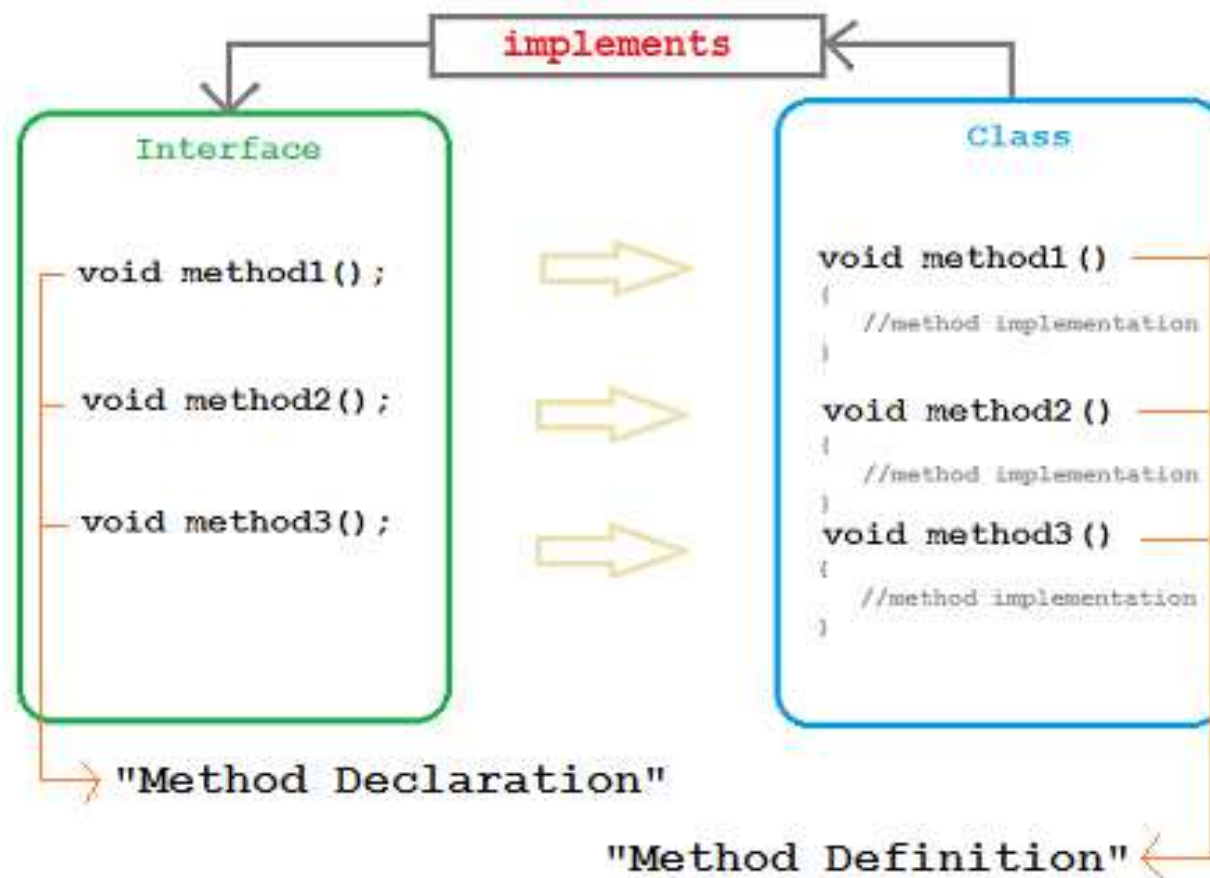
Tạo class thực thi interface:

## Syntax

```
class <class-name> implements <Interface1>,...  
{  
    // class members  
    // overridden abstract methods of the interface(s)  
}
```

# Sử dụng

Tất cả phương thức trừu tượng ở interface phải được *@override*:



# Sử dụng

Tạo class **TwoWheeler**:

```
package session11;
class TwoWheeler implements IVehicle {

    String ID; // variable to store vehicle ID
    String type; // variable to store vehicle type

    /**
     * Parameterized constructor to initialize values based on user input
     *
     * @param ID a String variable storing vehicle ID
     * @param type a String variable storing vehicle type
     */
    public TwoWheeler(String ID, String type){
        this.ID = ID;
        this.type = type;
    }

    /**
     * Overridden method, starts a vehicle
     */
}
```

# Sử dụng

```
* @return void
*/
@Override
public void start() {
    System.out.println("Starting the "+ type);
}

/**
 * Overridden method, accelerates a vehicle
 * @param speed an integer storing the speed
 * @return void
 */
@Override
public void accelerate(int speed) {
    System.out.println("Accelerating at speed:"+speed+ " kmph");
}

/**
 * Overridden method, applies brake to a vehicle
 *
 * @return void
```

# Sử dụng

```
*/  
@Override  
public void brake() {  
    System.out.println("Applying brakes");  
}  
  
/**  
 * Overridden method, stops a vehicle  
 *  
 * @return void  
 */  
@Override  
public void stop() {  
    System.out.println("Stopping the "+ type);  
}  
  
/**  
 * Displays vehicle details  
 *  
 * @return void  
 */  
public void displayDetails(){
```



# Sử dụng

```
        System.out.println("Vehicle No.: "+ STATEID+ " "+ ID);
        System.out.println("Vehicle Type.: "+ type);
    }
}

/**
 * Define the class TestVehicle.java
 *
 */
public class TestVehicle {

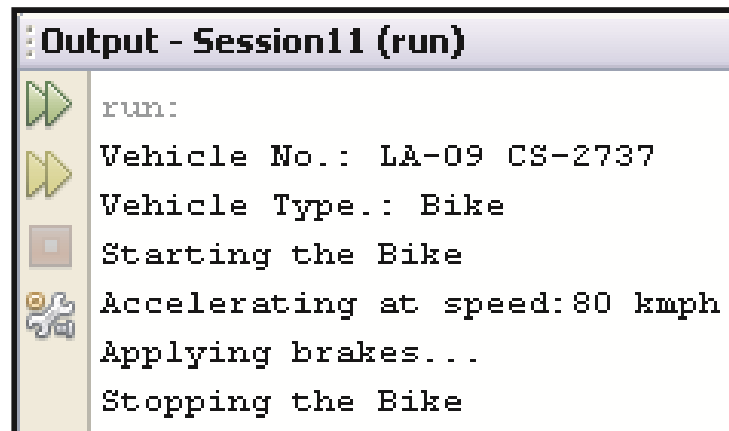
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args){

        // Verify the number of command line arguments
        if(args.length==3) {

            // Instantiate the TwoWheeler class
            TwoWheeler objBike = new TwoWheeler(args[0], args[1]);
```

# Sử dụng

```
// Invoke the class methods
objBike.displayDetails();
objBike.start();
objBike.accelerate(Integer.parseInt(args[2]));
objBike.brake();
objBike.stop();
}
else {
    System.out.println("Usage: java TwoWheeler <ID> <Type> <Speed>");
}
}
```



The screenshot shows the 'Output - Session11 (run)' window. It contains a list of messages with corresponding icons: a green play button for 'run:', a yellow play button for 'Vehicle No.: LA-09 CS-2737' and 'Vehicle Type.: Bike', a red square for 'Starting the Bike', a gear icon for 'Accelerating at speed:80 kmph', and a wrench icon for 'Applying brakes...'. The final message 'Stopping the Bike' is at the bottom without an icon.

```
Output - Session11 (run)
run:
Vehicle No.: LA-09 CS-2737
Vehicle Type.: Bike
Starting the Bike
Accelerating at speed:80 kmph
Applying brakes...
Stopping the Bike
```

# Sử dụng

---

Thực thi nhiều interface:

- Tạo thêm interface IManufacturer

```
package session11;  
public interface IManufacturer {  
  
    /**  
     * Abstract method to add contact details  
     * @param detail a String variable storing manufacturer detail  
     * @return void  
     */  
    public void addContact(String detail);  
  
    /**  
     * Abstract method to call the manufacturer
```

# Sử dụng

```
* @param phone a String variable storing phone number
* @return void
*/
public void callManufacturer(String phone);

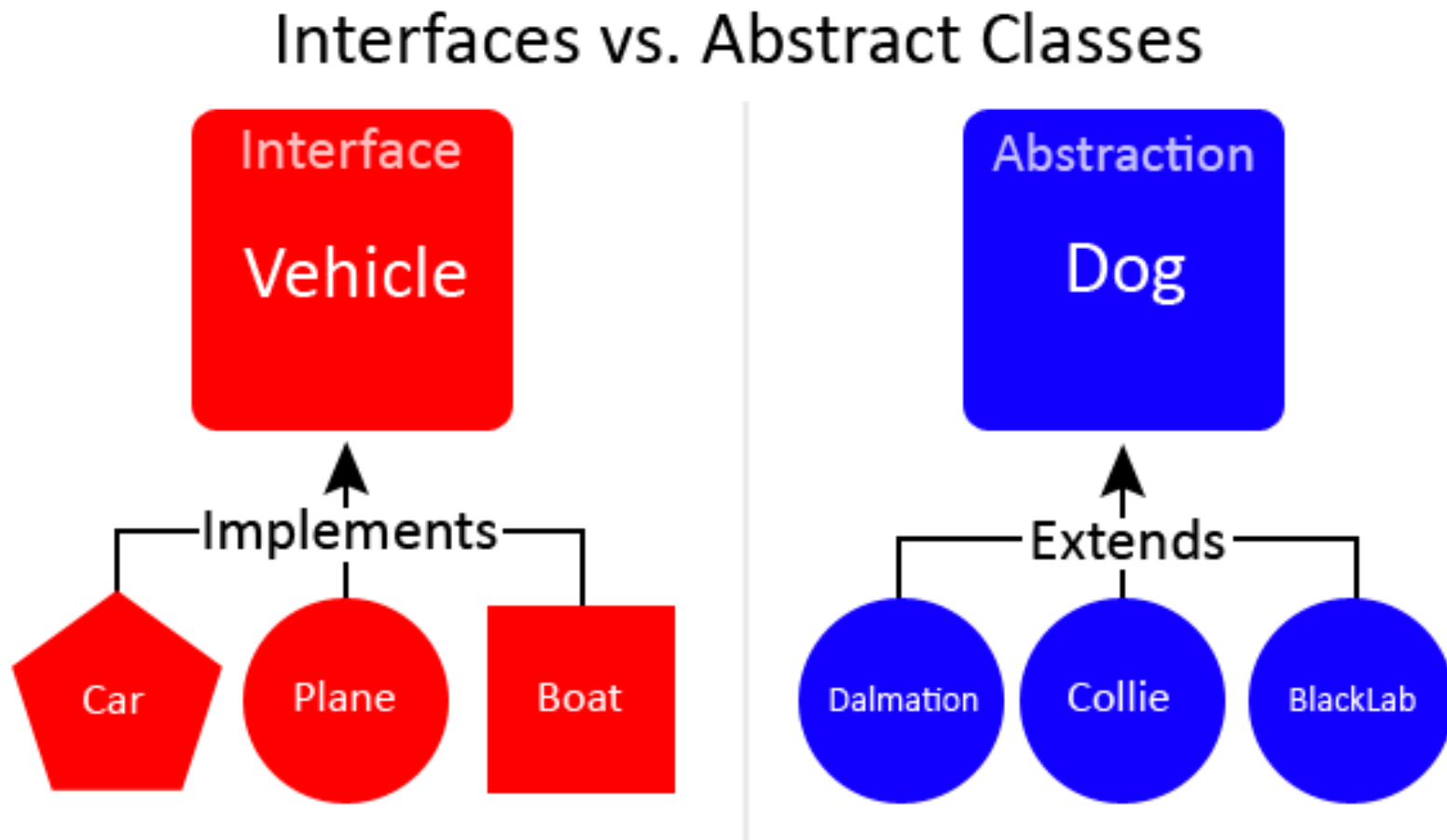
/**
 * Abstract method to make payment
 * @param amount a float variable storing amount
 * @return void
 */
public void makePayment(float amount);
}
```

```
package session11;
class TwoWheeler implements IVehicle, IManufacturer {

    String ID; // variable to store vehicle ID
    String type; // variable to store vehicle type
```

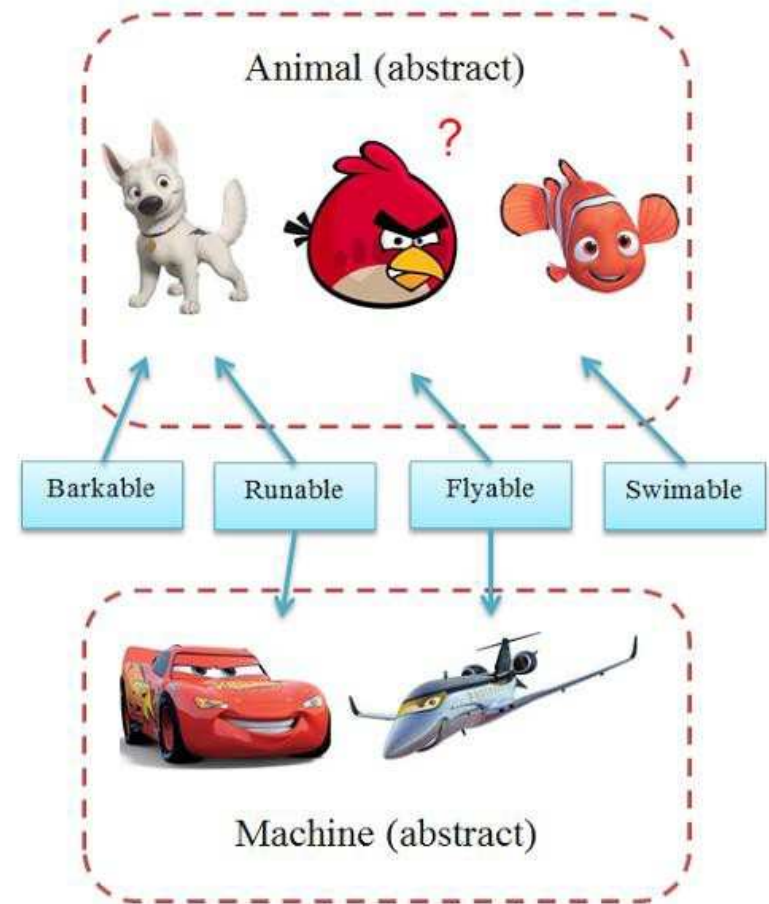
# Abstraction vs Interface

Tùy tình huống mà có thể sử dụng Interface hoặc Abstract class:



# Abstraction vs Interface

Tùy tình huống mà có thể sử dụng Interface hoặc Abstract class:



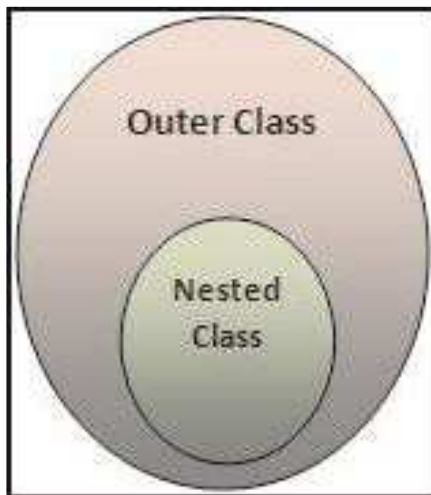
# Abstraction vs Interface

Sự khác nhau liệt kê như sau:

Abstract Class	Interface
Một <code>abstract class</code> có thể có đồng thời phương thức <code>abstract</code> và phương thức có thân.	Interface chỉ có phương thức <code>abstract</code> .
Một <code>abstract class</code> có thể có biến không phải <code>final</code> .	Biến trong interface mặc định hiểu ngầm là có bổ từ <code>final</code> .
Một <code>abstract class</code> có thể có các thành phần với các mô tả truy cập như <code>private</code> , <code>protected</code> ....	Thành phần trong interface buộc là <code>public</code> theo mặc định.
Một <code>abstract class</code> kế thừa sử dụng từ khóa <code>extends</code> .	Một interface thì thực thi sử dụng từ khóa <code>implements</code> .
Một <code>abstract class</code> có thể kế thừa một class khác và thực thi nhiều interfaces.	Một interface có thể mở rộng từ một hoặc nhiều interfaces.

# Lớp lồng (nested)

Trong class có thể khai báo thêm class bên trong chính nó

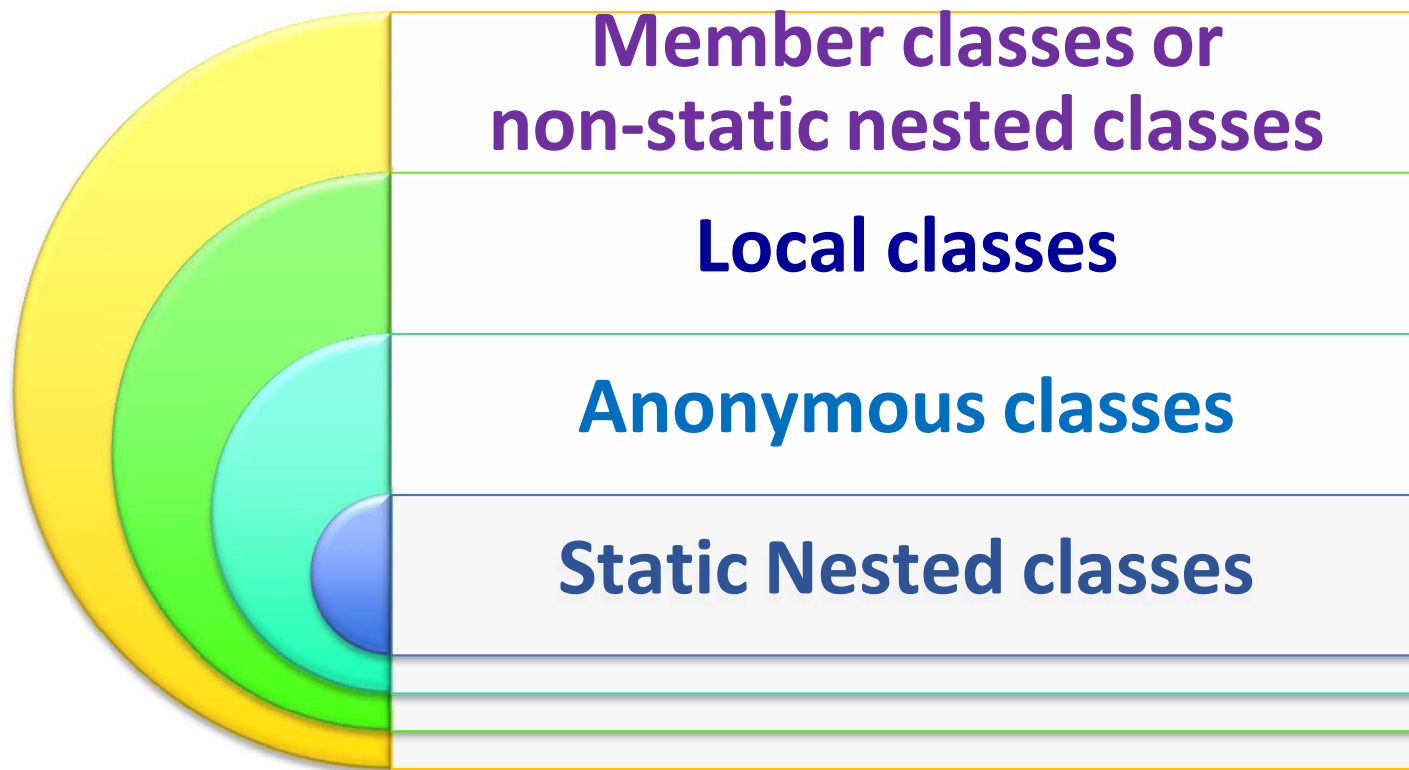


```
class Outer{  
    ...  
    class Nested{  
        ...  
    }  
}
```

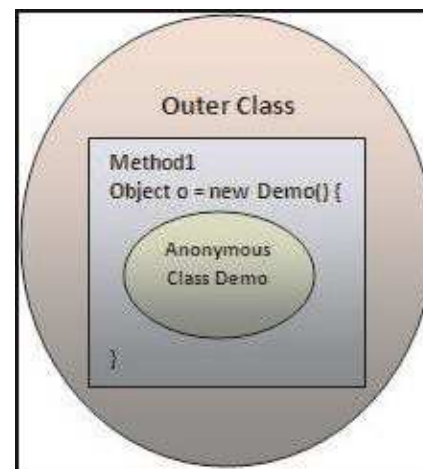
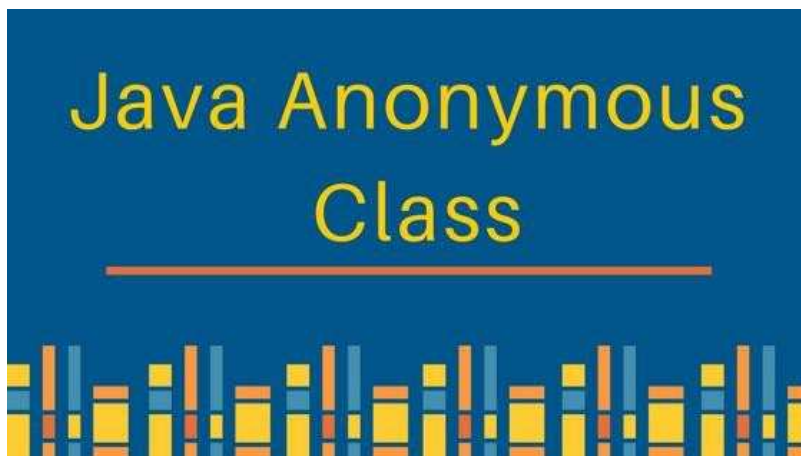


# Lớp lồng (nested)

- Nhóm các lớp có chung tính logic.
- Tăng tính bao gói
- Tăng khả năng đọc và bảo trì mã nguồn



# Lớp nặc danh (anonymous)



Một lớp được khai báo không có tên ở bên trong khối lệnh của phương thức chính là lớp Anonymous.

Một anonymous class thì không có tên, vì vậy nó chỉ có thể được truy cập tại thời điểm nó được định nghĩa.

Không thể sử dụng với từ khóa `extends` và `implements` keywords cũng như các bổ từ truy cập `public`, `private`, `protected`, và `static`.

# Lớp nặc danh (anonymous)

---

Lớp nặc danh không định nghĩa constructor, `static` với trường, phương thức, hoặc lớp.

Nó không thể implement anonymous interfaces bởi vì một interface không thể implemented mà không có tên.

Không có tên, không có constructor nhưng có thể khởi tạo thành cá thể.

Truy cập lớp nặc danh tương tự như lớp local.

# Tóm tắt bài học

---

- ✓ **Interface** định nghĩa các **ràng buộc** mà theo đó những lớp thực thi phải làm theo.
- ✓ Có thể **thực thi nhiều interface** bằng cách đặt tên chúng phân cách bởi dấu “,”.
- ✓ Java cho phép định nghĩa class bên trong một class khác (**inner** class).
- ✓ Lớp định nghĩa bên trong khối lệnh và không có tên gọi là lớp nặc danh (**anonymous**).