NGÔN NGỮ LẬP TRÌNH JAVA

Giảng viên: TS. Vũ Hữu Tiến
Email: tienvh@ptit.edu.vn
Mobile: 0939919396

GIỚI THIỆU MÔN HỌC

- Nội dung môn học:
  - **Chương 1:** Tổng quan về lập trình hướng đối tượng và ngôn ngữ lập trình Java
  - **Chương 2:** Lớp và đối tượng trong Java
  - **Chương 3:** Tính kế thừa và đa hình trong Java
  - **Chương 4:** Xử lý nhập/ xuất trong Java
  - **Chương 5:** Xử lý ngoại lệ trong Java
  - **Chương 6:** Lập trình đa luồng trong Java
  - **Chương 7:** Lập trình giao diện trong Java

MULTIMEDIA

KẾ THỪA

- **Kế thừa (inheritance)**: là một mối quan hệ giữa các lớp với nhau trong Java.
- Kế thừa cho phép chia sẻ các phương thức của lớp cha với lớp con.
- Lớp con có thể bổ sung thêm phương thức mới hoặc gán chồng (override) phương thức đã có ở lớp cha.

KẾ THỪA

- **Một số thuật ngữ:**
  - **superclass**, **base class**, **parent class**: lớp cha
  - **subclass**, **derived class**, **child class**: lớp con
  - **extend**, **inherit**, **derive**: kế thừa

```
public class Student extends Person {

}
```

KẾ THỪA

- **Mối quan hệ**
  - Mối quan hệ bao gồm "Has-a": Khi một đối tượng chứa một đối tượng khác.

```
public class Person {
        protected String name;
}
```

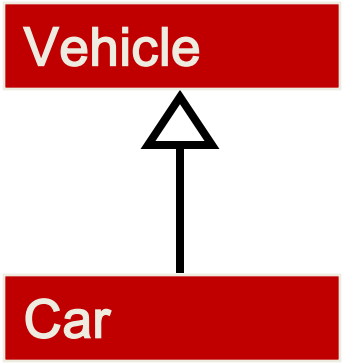
  - Mối quan hệ “là-một” – “is-a”: Khi một đối tượng kế thừa một đối tượng khác.
    - Đối tượng Student “là-một” Person

```
public class Student extends Person {

}
```

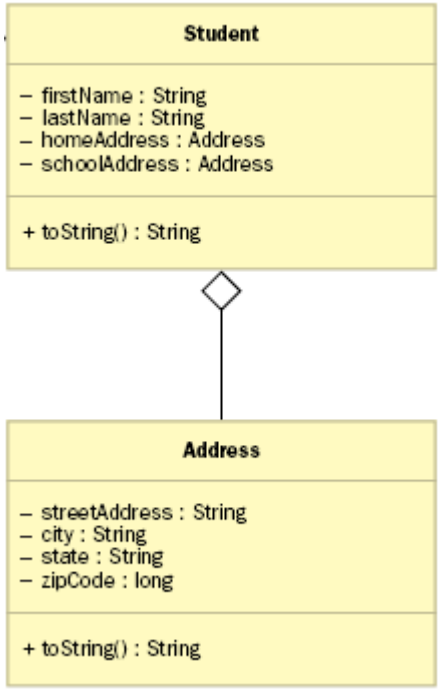
KẾ THỪA

- **UML**
  - Biểu diễn mối quan hệ kế thừa (is-a):



KẾ THỪA

- **UML**
  - Biểu diễn mối quan hệ bao gồm:
  - Mối quan hệ này được sử dụng khi có nhiều đối tượng (ví dụ Student, Employee, Teacher,…) đều có mỗi quan hệ "Has a" với lớp Address



KẾ THỪA

- **Overriding: gán chồng**
  - Lớp con có thể gán chồng một phương thức của lớp cha bằng cách định nghĩa lại phương thức đó.

```
public class BankAccount {
  private double myBalance;  // ....

  public String toString() {
    return getID() + " $" + getBalance();
} }

public class FeeAccount extends BankAccount {
  private static final double FEE = 2.00;

  public String toString() {     // overriding
    return getID() + " $" + getBalance()
      + " (Fee: $" + FEE + ")";
} }
```

```
BankAccount b = new BankAccount("Ed", 9.0);
FeeAccount f = new FeeAccount("Jen", 9.0);
System.out.println(b.toString());
System.out.println(f.toString());
```

Output:

```
Ed $9.0
Jen $9.0 (Fee: $2.0)
```

KẾ THỪA

- **Modifier**
  - Public: Tất cả các lớp có thể truy cập
  - Private: Các phương thức trong phạm vi lớp và các đối tượng của lớp đó có thể truy cập.
  - Protected: các phương thức trong phạm vi lớp cha và lớp con có thể truy cập.

KẾ THỪA

- **Modifier**

```
public class Point2D {
  private int x, y;
  public Point2D(int x, int y) {
    this.x = x;   this.y = y;
  }
}

public class Point3D extends Point2D {
  private int z;
  public Point3D(int x, int y, int z) {
    this.x = x;  this.y = y;  // Báo lỗi!
    this.z = z;
  }
}
```

KẾ THỪA

- Lớp Giao diện (interface)
  - *Lớp giao diện* là một tập các phương thức public được đưa ra để người dùng có thể tương tác với đối tượng.
  - Lớp giao diện bao gồm các hằng số và các *phương thức trừu tượng*.
  - Phương thức trừu tượng là phương thức chỉ có khai báo mà không có triển khai.

```
interface MyInterface {
        public void method1();
        public void method2();
}
```

KẾ THỪA

- Lớp Giao diện (interface)
  - *Lớp triển khai*

```
class Demo implements MyInterface {
public void method1() {
        System.out.println("implementation of method1");
        }
public void method2() {
        System.out.println("implementation of method2");
        }
public static void main(String arg[]) {
        MyInterface obj = new Demo(); obj.method1();
        }
}
```

KẾ THỪA

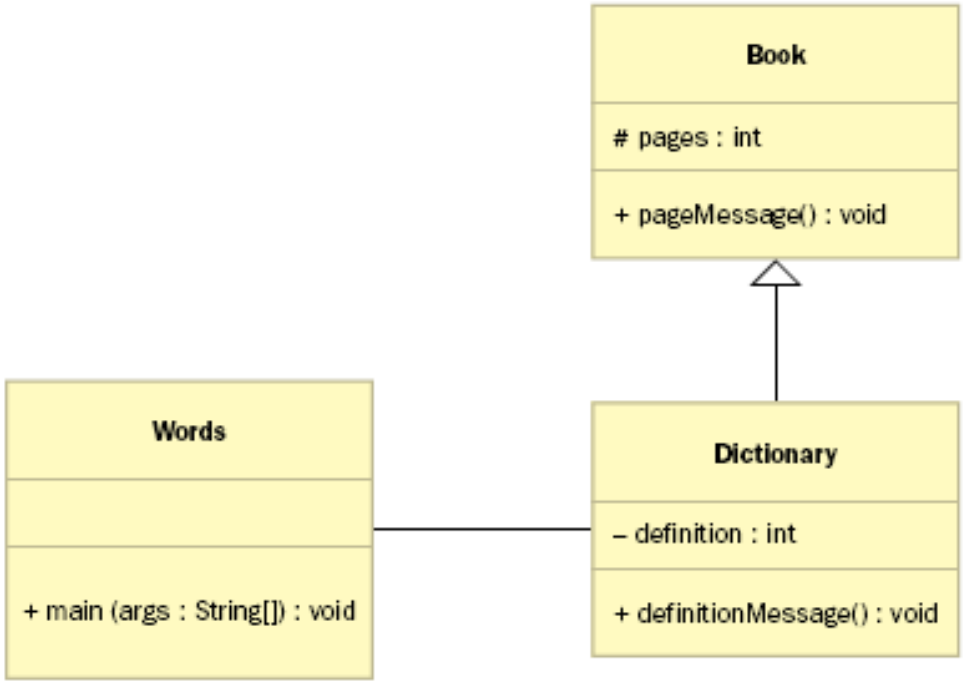
- Lớp Giao diện (interface)
  - *Tại sao phải cần lớp giao diện:* Lớp giao diện được dùng khi thiết kế một lớp có các phương thức chung. Các lớp triển khai lớp giao diện này bắt buộc phải tuân thủ các phương thức chung đó.
  - Ví dụ: Lớp giao diện *Độngvật* có phương thức *kêu()* có nghĩa là tất cả các lớp động vật đều phải *kêu* nhưng có thể *kêu* với các cách *implement* khác nhau. Lớp người là nói, lớp chó là sủa, lớp hổ là gầm,…

KẾ THỪA

- Lớp Giao diện (interface)
  - Lớp giao diện chỉ cho một lớp giao diện khác kế thừa.
  - Lớp giao diện chỉ chứa các phương thức trừu tượng mà không có các phương thức cụ thể.
  - Các phương thức trừu tượng trong lớp giao diện chỉ có thể là public.
  - Lớp giao diện chỉ chứa các biến loại public static final.

KẾ THỪA

- Bài tập: Viết chương trình theo mô tả dưới đây
  - PageMessage: in ra số trang cả Book
  - definitionMessage: in ra số định nghĩa/1 trang



MULTIMEDIA

KẾ THỪA

- Lớp trừu tượng
  - Lớp trừu tượng được sử dụng để mô tả về một khái niệm chung trong một cây gia phả.
  - Lớp trừu tượng không được hiện thực hóa.
  - Lớp trừu tượng thông thường bao gồm các phương thức trừu tượng.
  - Các lớp con của lớp trừu tượng có nhiệm vụ gán chồng tất cả các phương thức trừu tượng của lớp trừu tượng.

KẾ THỪA

- Lớp trừu tượng
  - Ví dụ:

```
//abstract parent class
stract class Animal{
        //abstract method
        public abstract void sound();
}
//Dog class extends Animal class
public class Dog extends Animal{
        public void sound(){
                System.out.println("Woof");
        }
        public static void main(String args[]){
                Animal obj = new Dog();
                obj.sound();
        }
}
```

KẾ THỪA

- Lớp trừu tượng
  - Tại sao cần sử dụng lớp trừu tượng?
    - Khi các lớp con đều có cùng phương thức nào đó giống như lớp cha, nhưng các phương thức này khác nhau đối với mỗi lớp con.
    - Ngoài ra, tất cả các lớp con bắt buộc phải có phương thức này.
    - Ví dụ:
      - Nếu là lớp con của lớp Animal thì đều có phương thức là "kêu". Tuy nhiên, phương thức "kêu" của các lớp Dog, Cat, Lion,… đều khác nhau.
      - Sử dụng lớp trừu tượng Animal là để yêu cầu tất cả các động vật là lớp con của Animal đều phải có phương thức "kêu".

KẾ THỪA

- Lớp trừu tượng
  - Tại sao cần sử dụng lớp trừu tượng?
    - Khi các lớp con đều có cùng phương thức nào đó giống như lớp cha, nhưng các phương thức này khác nhau đối với mỗi lớp con.
    - Ngoài ra, tất cả các lớp con bắt buộc phải có phương thức này.
    - Ví dụ:
      - Nếu là lớp con của lớp Animal thì đều có phương thức là "kêu". Tuy nhiên, phương thức "kêu" của các lớp Dog, Cat, Lion,… đều khác nhau.
      - Sử dụng lớp trừu tượng Animal là để yêu cầu tất cả các động vật là lớp con của Animal đều phải có phương thức "kêu".

MULTIMEDIA

KẾ THỪA

- Lớp trừu tượng
  - Một số nguyên tắc sử dụng lớp trừu tượng
    - Lớp trừu tượng không nhất thiết phải implement toàn bộ phương thức của nó.
    - Lớp con kế thừa lớp trừu tượng nhất thiết phải implement phương thức abstract của lớp trừu tượng.

```
/Declaration using abstract keyword
abstract class A{ /
        /This is abstract method
        abstract void myMethod();
        //This is concrete method with body
        void anotherMethod(){
        //Does something
        }
}
```

KẾ THỪA

- Lớp trừu tượng
  - Một số nguyên tắc sử dụng lớp trừu tượng
    - Không thể tạo ra đối tượng trực tiếp của lớp trừu tượng. Nếu muốn tạo đối tượng thì phải tạo lớp con của lớp của trừu tượng.
    - Nếu lớp con của lớp trừu tượng không muốn implement các phương thức của lớp cha thì lớp con đó cũng phải khai báo là lớp trừu tượng.

An abstract class is like a template

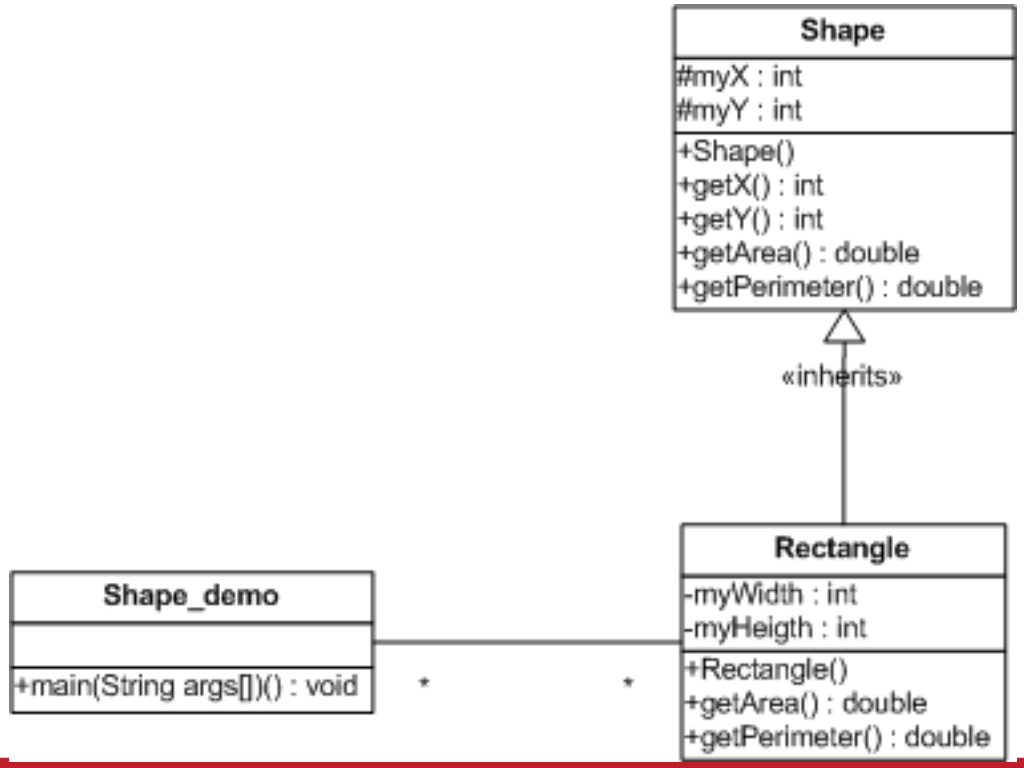
MULTIMEDIA

KẾ THỪA

- Lớp trừu tượng vs. Lớp giao diện
  - Có thể thừa kế từ nhiều interface nhưng chỉ có thể thừa kế từ 1 abstract class.
  - Có thể viết sẵn các thực thi trong Abstract class, nhưng interface thì không.
  - Interface dùng để gom các hành động cần được hiện thực, các khả năng của một đối tượng (ví dụ con người và xe cộ cùng có khả năng đi lại, nhưng con người không cùng tính chất với xe cộ), còn abstract class cho các lớp thừa kế cùng 1 loại, tính chất hay trạng thái (ví dụ đàn ông, phụ nữ, trẻ con, người già đều là con người).
  - Abstract class có tốc độ thực thi nhanh hơn interface.
  - Thêm 1 tính năng mới vào interface sẽ phá vỡ toàn bộ các lớp hiện thực, còn abstract thì không.

KẾ THỪA

- Lớp trừu tượng
  - Bài tập: Viết chương trình tính diện tích và chu vi hình chữ nhật. Lowps Shape là lớp trừu tượng.



KẾ THỪA

- Lớp trừu tượng

```
public abstract class Shape{
   private int myX, myY;

   public Shape(int x, int y) {
    myX = x;  myY = y;
   }
   public int getX() { return myX; }
   public int getY() { return myY; }

   public abstract double getArea();
   public abstract double getPerimeter();
}
```

KẾ THỪA

- Lớp trừu tượng

```
public class Rectangle extends Shape {
  private int myWidth, myHeight;

  public Rectangle(int x, int y, int w, int h)
  {
    super(x, y);
    myWidth = w;   myHeight = h;
  }

  public double getArea() {
    return myWidth * myHeight;
  }
  public double getPerimeter() {
    return 2*myWidth + 2*myHeight;
  }
}
```

KẾ THỪA

- **Super**
  - Từ khóa super được sử dụng để truy cập đến các **biến**, constructor, phương thức của lớp cha.

```
//Parent class or Superclass or base class
class Superclass {
        int num = 100;
}
//Child class or subclass or derived class
class Subclass extends Superclass {
/* The same variable num is declared in the Subclass
* which is already present in the Superclass */
        int num = 110;
        void printNumber(){
        System.out.println(super.num);
}
        public static void main(String args[]){
                Subclass obj= new Subclass();
                obj.printNumber();
        }
}
```

KẾ THỪA

- **Super**
  - Từ khóa super mặc định được sử dụng để gọi phương thức khởi tạo của lớp cha khi gọi phương thức khởi tạo của lớp con.

```
public class parent {
        parent(){
                System.out.println("Constructor of parent class");
        }
}

public class subClass extends parent {
        subClass(){
                System.out.println("Constructor of child class");
        }
}

public class abstract_demo {
        public static void main(String[] args) {
                subClass obj = new subClass();
        }
}
```

KẾ THỪA

- **Super**
  - Từ khóa super được sử dụng để truy cập đến các biến, constructor, **phương thức** của lớp cha.

```
public class parent {
	void display(){
		System.out.println("Hello parentClass!");
	}
}
```

```
public class subClass extends parent {
	void display(){
		System.out.println("Hello subClass!");
	}
	void displayMsg(){
		display();
		super.display();
	}
```

```
public class abstract_demo {
	public static void main(String[] args) {
		subClass obj1= new subClass(10,"Hello");
		obj1.displayMsg();
	}
}
```

ĐA HÌNH (POLYMORPHISM)

- **Khái niệm đa hình**
  - Đa hình là một đặc điểm của OPP cho phép đối tượng thực hiện một hành động theo nhiều cách.
  - Có 2 loại đa hình:
    - Compile time polymorphism (overloading)
    - Run-time polymorphism (overriding)

ĐA HÌNH (POLYMORPHISM)

- **Khái niệm đa hình**
  - Overriding polymorphism

```
public class Animal{
          public void sound(){
          System.out.println("Animal is making a sound");
          }
}
```

```
public class Cat extends Animal{
          public void sound(){
          System.out.println("Meow"); } }
```

```
class Horse extends Animal{
          public void sound(){
                    System.out.println("Neigh");
          }
          public static void main(String args[]){
                    Animal obj = new Horse();
                    obj.sound();
                    Animal obj = new Cat();
                    obj.sound();
          }
}
```

ĐA HÌNH (POLYMORPHISM)

- **Khái niệm đa hình**
  - Lưu ý: Khi khai báo một đối tượng kiểu của lớp cha, đối tượng đó không thể gọi các phương thức của lớp con.

```
Object var2 = new Person();
var2.getName(); // Báo lỗi do Object không có phương thức getName()
```

  - Muốn gọi các phương thức của lớp con, biến đó phải được ép kiểu

```
((Person)var2).getName();
```

ĐA HÌNH (POLYMORPHISM)

- **Bài tập**
  - Viết chương trình quản lý lương của nhân viên trong công ty.