

**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

# **NGÔN NGỮ LẬP TRÌNH JAVA**

## **Chương 2** **HƯỚNG ĐỐI TƯỢNG TRONG JAVA (P2)**

GVGD: ThS. Lê Thanh Trọng

# NỘI DUNG

- 1. Tính kế thừa**
- 2. Tính đa hình**
- 3. Interface**
- 4. Chương trình minh họa**

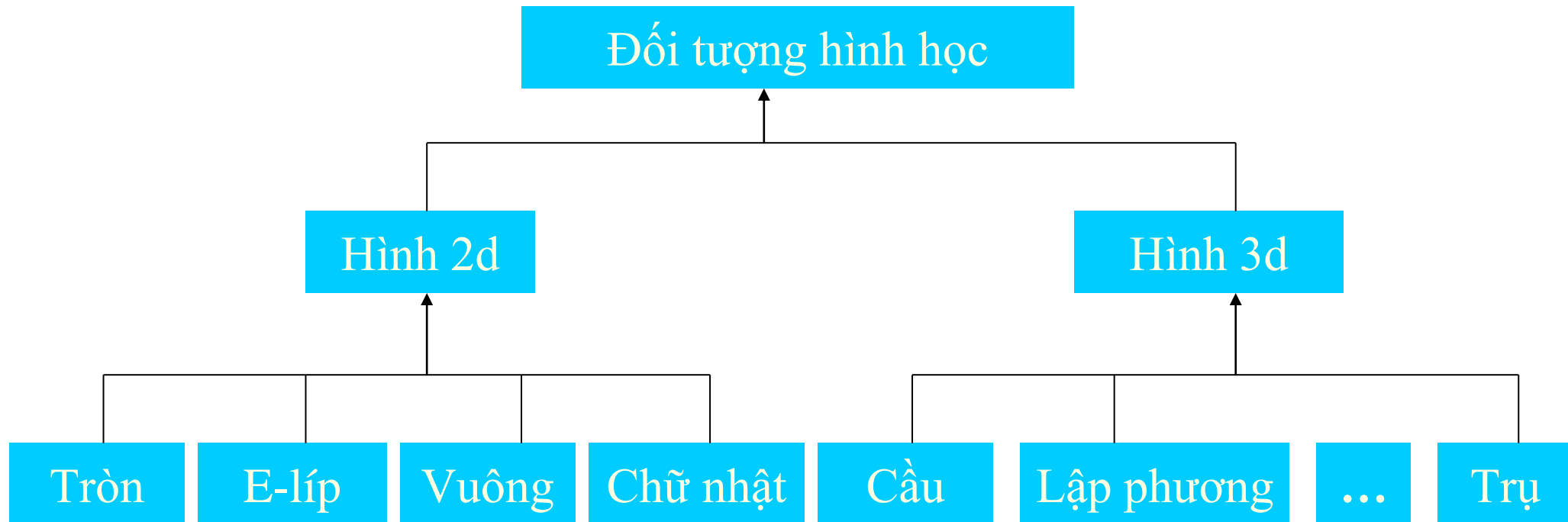
# NỘI DUNG

- 1. Tính kế thừa**
2. Tính đa hình
3. Interface
4. Chương trình minh họa

# Tính kế thừa

- ❖ Xây dựng lớp (lớp con/subclass) mới dựa trên lớp đã có (lớp cha/superclass)
- ❖ Lớp con thừa hưởng các thuộc tính và phương thức của lớp cha
- ❖ Lớp con bổ sung, chi tiết hóa cho phù hợp hơn
- ❖ Thuộc tính: thêm mới
- ❖ Phương thức: thêm mới hay hiệu chỉnh

# Tính kế thừa



# Tính kế thừa

- ❖ Lớp con có thể kế thừa tất cả hay một phần các thành phần dữ liệu (thuộc tính), phương thức của lớp cha (public, protected, default)
- ❖ Dùng từ khóa **extends**

```
class nguoi { ...
```

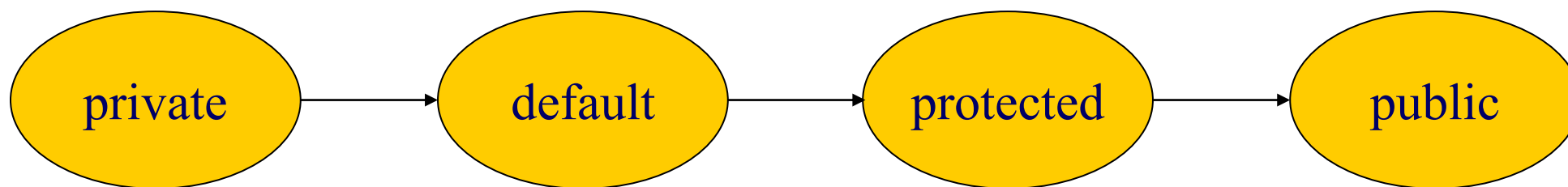
```
}
```

```
class sinhVien extends nguoi { ...
```

```
}
```

# Overriding Method

- ❖ Được định nghĩa trong lớp con
- ❖ Có tên, kiểu trả về & các đối số giống với phương thức của lớp cha
- ❖ Có kiểu, phạm vi truy cập ko “nhỏ hơn” phương thức trong lớp cha



# Overriding Method

```
class HinhHoc { ...  
    public float tinhDienTich() {  
        return 0;  
    }  
    ...  
}
```

```
class HinhVuong extends HinhHoc {  
    private int Canh;  
    public float TinhDienTich() {  
        return Canh*Canh;  
    }  
    ...  
}
```

Chỉ có thể **public** do phương thức `tinhDientich()` của lớp cha là **public**



# Overriding Method

```
class HìnhChuNhat extends HìnhVuong {  
    private int Dai;  
    private int Rong;  
    public float tinhDienTich() {  
        return Dai*Rong;  
    }  
    ...  
}
```

Chỉ có thể **public** do phương thức *tinhdientich()* của lớp cha là **public**

# Lớp Object

- ❖ Cây kế thừa trong Java chỉ có 1 gốc.
- ❖ Mọi lớp đều kế thừa trực tiếp hoặc gián tiếp từ lớp Object.
- ❖ Phương thức của lớp Object

Phương thức	Mô tả
<i>clone</i>	Tạo đối tượng là một bản sao
<i>equals</i>	So sánh bằng (giá trị) giữa 2 đối tượng
<i>finalize</i>	Gọi với GC (Garbage Collection) khi không còn tham chiếu đến đối tượng
<i>getClass</i>	Trả về lớp đối tượng thuộc về
<i>hashCode</i>	Trả về giá trị hashcode của đối tượng
<i>notify/notifyAll</i>	Kích hoạt luồng/tất cả các luồng đang đợi đến lượt truy cập đối tượng
<i>wait</i>	Thread đang truy cập đối tượng đợi đến khi có lệnh notify/notifyAll
<i>toString</i>	Trả về chuỗi đại diện cho đối tượng

# Từ khóa super

- ❖ Gọi constructor của lớp cha
- ❖ Nếu gọi tường minh thì phải là câu lệnh đầu tiên
- ❖ Constructor cuối cùng được gọi là của lớp Object
- ❖ Truy cập đến thuộc tính bị che ở lớp cha

```
class SuperClass
{
    SuperClass()
    {
        System.out.println("SuperClass
constructor is called");
    }
}
```

```
class SubClass extends SuperClass
{
    SubClass()
    {
        System.out.println("SubClass
constructor is called");
        super();
    }
}
```

# Lớp Object

- ❖ Cây kế thừa trong Java chỉ có 1 gốc
- ❖ Mọi lớp đều kế thừa trực tiếp hoặc gián tiếp từ lớp Object
- ❖ Phương thức của lớp Object

Phương thức	Mô tả
<i>clone</i>	Tạo đối tượng là một bản sao
<i>equals</i>	So sánh bằng (giá trị) giữa 2 đối tượng
<i>finalize</i>	Gọi với GC (Garbage Collection) khi không còn tham chiếu đến đối tượng
<i>getClass</i>	Trả về lớp đối tượng thuộc về
<i>hashCode</i>	Trả về giá trị hashcode của đối tượng
<i>notify/notifyAll</i>	Kích hoạt luồng/tất cả các luồng đang đợi đến lượt truy cập đối tượng
<i>wait</i>	Thread đang truy cập đối tượng đợi đến khi có lệnh notify/notifyAll
<i>toString</i>	Trả về chuỗi đại diện cho đối tượng

# Lớp final

- ❖ Là lớp không cho phép các lớp khác dẫn xuất từ nó hay lớp final không thể có lớp con.
- ❖ Định nghĩa dùng từ khóa final

```
public final class A {
```

```
    ...
```

```
}
```

# NỘI DUNG

1. Tính kế thừa
- 2. Tính đa hình**
3. Interface
4. Chương trình minh họa

# Tính đa hình

- ❖ Cùng một phương thức có thể có những cách thi hành khác nhau
- ❖ Interface: được cài đặt bởi các lớp con để triển khai một phương thức mà lớp muốn có
- ❖ Lớp trừu tượng: lớp dùng để thể hiện sự trừu tượng hóa ở mức cao, ví dụ: lớp "Đối tượng hình học", "Hình 2D", "Hình 3D"
- ❖ Từ khóa **abstract**: để khai báo một lớp trừu tượng (abstract), lớp trừu tượng không thể tạo ra đối tượng

# NỘI DUNG

1. Tính kế thừa
2. Tính đa hình
- 3. Interface**
4. Chương trình minh họa



# Interface

- ❖ Interface: giao tiếp của một lớp, là phần đặc tả (không có phần cài đặt cụ thể) của lớp
- ❖ Chứa các khai báo phương thức và thuộc tính để bên ngoài có thể truy xuất được
- ❖ Lớp hiện thực (**implements**) interface: nếu lớp không cài đặt hết các phương thức của giao diện thì phải được khai báo là abstract
- ❖ Người lập trình dựa vào interface để gọi các dịch vụ mà lớp cung cấp
- ❖ Thuộc tính là các **hằng, static, public**
- ❖ Các phương thức phải là **public, abstract**

# Interface

*// Định nghĩa một interface Shape trong tập tin shape.java*

*public **interface** Shape {*

*// Tính diện tích*

*public abstract double computeArea();*

*// Tính thể tích*

*public abstract double computeVolume();*

*// trả về tên của shape*

*public abstract String getName();*

*}*

# Interface

*// Lớp Point cài đặt/hiện thực interface tên Shape*

*public class Point **extends** Object **implements** Shape {*

*protected int x, y; // Tọa độ x, y của 1 điểm*

*// constructor không tham số.*

*public Point() {*

*setPoint( 0, 0 );*

*}*

*// constructor có tham số.*

*public Point(int xCoordinate, int yCoordinate) {*

*setPoint( xCoordinate, yCoordinate );*

*}*

# Interface

*// gán tọa độ x, y cho 1 điểm*

```
public void setPoint( int xCoordinate, int yCoordinate ) {
```

```
    x = xCoordinate;
```

```
    y = yCoordinate;
```

```
}
```

*// lấy tọa độ x của 1 điểm*

```
public int getX() {
```

```
    return x;
```

```
}
```

*// lấy tọa độ y của 1 điểm*

```
public int getY() {
```

```
    return y;
```

```
}
```

# Interface

*// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi*

```
public String toString() {  
    return "[" + x + ", " + y + "];  
}
```

*// Tính diện tích*

```
public double ComputeArea() {  
    return 0.0;  
}
```

*// Tính thể tích*

```
public double ComputeVolume() {  
    return 0.0;  
}
```

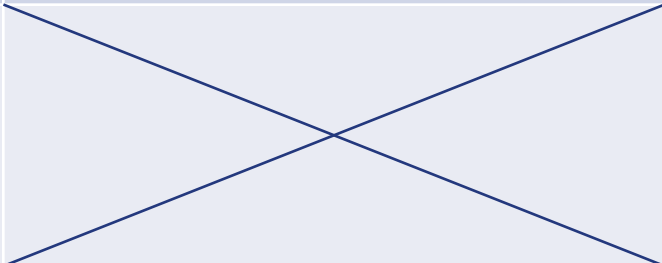
# Interface

```
// trả về tên của đối tượng shape  
public String getName() {  
    return "Point";  
}  
} // end class Point
```

# Kế thừa interface

```
public interface InterfaceName extends interface1, interface2, interface3  
  
{  
  
    // ...  
  
}
```

# Quan hệ giữa class và interface

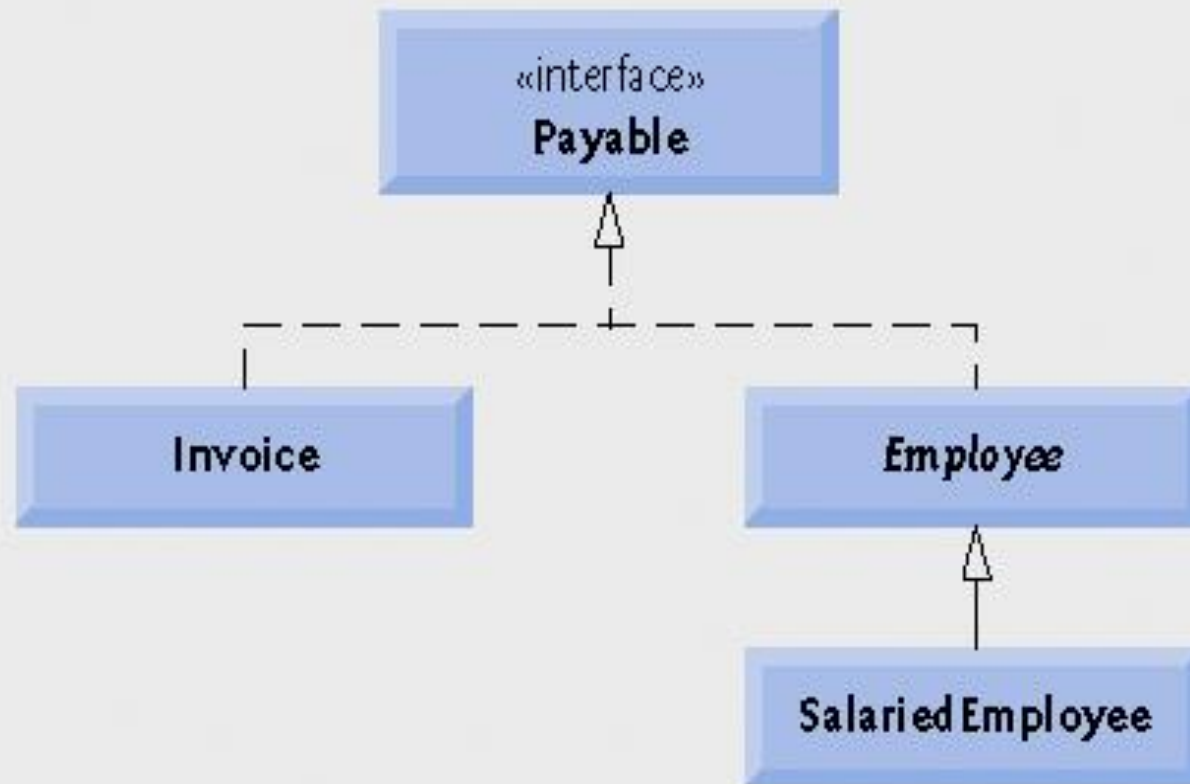
	Class	Interface
Class	extends	implements
Interface		extends



# NỘI DUNG

1. Tính kế thừa
2. Tính đa hình
3. Interface
- 4. Chương trình minh họa**

# Ví dụ



# Payable.java



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

```
public interface Payable  
{  
    double getPaymentAmount();  
} // end interface Payable
```

# Invoice.java

*public class Invoice implements Payable*

```
{  
    private String partNumber;  
    private String partDescription;  
    private int quantity;  
    private double pricePerItem;  
  
    // four-argument constructor  
    public Invoice( String part, String description, int count,  
        double price )  
    {  
        partNumber = part;  
        partDescription = description;  
        setQuantity( count ); // validate and store quantity  
        setPricePerItem( price ); // validate and store price per  
item  
    } // end four-argument Invoice constructor  
  
    // set part number  
    public void setPartNumber( String part )  
    {  
        partNumber = part;  
    } // end method setPartNumber  
  
    // get part number  
    public String getPartNumber()  
    {  
        return partNumber;  
    } // end method getPartNumber
```

# Invoice.java



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

*// set description*

```
public void setPartDescription( String description )  
{  
    partDescription = description;  
} // end method setPartDescription
```

*// get description*

```
public String getPartDescription()  
{  
    return partDescription;  
} // end method getPartDescription
```

*// set quantity*

```
public void setQuantity( int count )  
{  
    quantity = ( count < 0 ) ? 0 : count; // quantity cannot  
be negative  
} // end method setQuantity
```

*// get quantity*

```
public int getQuantity()  
{  
    return quantity;  
} // end method getQuantity
```

*// set price per item*

```
public void setPricePerItem( double price )  
{  
    pricePerItem = ( price < 0.0 ) ? 0.0 : price; // validate  
price  
} // end method setPricePerItem
```

# Invoice.java

```
// get price per item
public double getPricePerItem()
{
    return pricePerItem;
} // end method getPricePerItem

// return String representation of Invoice object
public String toString()
{
    return String.format( "%s: \n%s: %s (%s) \n%s: %d
\n%s: $%,.2f",
        "invoice", "part number", getPartNumber(),
        getPartDescription(),
        "quantity", getQuantity(), "price per item",
        getPricePerItem() );
} // end method toString

// method required to carry out contract with interface
```

```
Payable
public double getPaymentAmount()
{
    return getQuantity() * getPricePerItem(); // calculate
total cost
} // end method getPaymentAmount
} // end class Invoice
```

# Employee.java

```
public abstract class Employee implements Payable
{
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;

    // three-argument constructor
    public Employee( String first, String last, String ssn
)
    {
        firstName = first;
        lastName = last;
        socialSecurityNumber = ssn;
    } // end three-argument Employee constructor

    // set first name

    public void setFirstName( String first )
    {
        firstName = first;
    } // end method setFirstName

    // return first name
    public String getFirstName()
    {
        return firstName;
    } // end method getFirstName

    // set last name
    public void setLastName( String last )
    {
        lastName = last;
    } // end method setLastName
```

# Employee.java

```
// return last name
public String getLastName()
{
    return lastName;
} // end method getLastName

// set social security number
public void setSocialSecurityNumber( String ssn )
{
    socialSecurityNumber = ssn; // should validate
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
    return socialSecurityNumber;
} // end method getSocialSecurityNumber

// return String representation of Employee object
public String toString()
{
    return String.format( "%s %s\nsocial security
number: %s",
        getFirstName(), getLastName(),
        getSocialSecurityNumber() );
} // end method toString

// Note: We do not implement Payable method
getPaymentAmount here so
// this class must be declared abstract to avoid a
compilation error.
} // end abstract class Employee
```



# SalariedEmployee.java

```
public class SalariedEmployee extends  
Employee  
{  
    private double weeklySalary;  
    // four-argument constructor  
    public SalariedEmployee( String first,  
String last, String ssn,  
        double salary )  
    {  
        super( first, last, ssn ); // pass to  
Employee constructor  
        setWeeklySalary( salary ); // validate and  
store salary  
    } // end four-argument SalariedEmployee  
  
    constructor  
    // set salary  
    public void setWeeklySalary( double  
salary )  
    {  
        weeklySalary = salary < 0.0 ? 0.0 :  
salary;  
    } // end method setWeeklySalary  
  
    // return salary  
    public double getWeeklySalary()  
    {  
        return weeklySalary;  
    } // end method getWeeklySalary
```

# SalariedEmployee.java

```
// calculate earnings; implement interface
Payable method that was
// abstract in superclass Employee
public double getPaymentAmount()
{
    return getWeeklySalary();
} // end method getPaymentAmount

// return String representation of
SalariedEmployee object
public String toString()
{
    return String.format( "salaried
employee: %s\n%s: $%,.2f",
super.toString(), "weekly salary",
getWeeklySalary() );
} // end method toString
} // end class SalariedEmployee
```

# PayableInterfaceTest.java

```
public class PayableInterfaceTest
{
    public static void main( String args[] )
    {
        // create four-element Payable array
        Payable payableObjects[] = new Payable[ 4 ];

        // populate array with objects that implement Payable
        payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
        payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
        payableObjects[ 2 ] =
            new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
        payableObjects[ 3 ] =
            new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );
    }
}
```

# PayableInterfaceTest.java

```
System.out.println(  
    "Invoices and Employees processed polymorphically:\n" );  
  
// generically process each element in array payableObjects  
for ( Payable currentPayable : payableObjects )  
{  
    // output currentPayable and its appropriate payment amount  
    System.out.printf( "%s \n%s: $%,.2f\n\n",  
        currentPayable.toString(),  
        "payment due", currentPayable.getPaymentAmount() );  
} // end for  
} // end main  
} // end class PayableInterfaceTest
```

# Kết quả

Invoices and Employees processed polymorphically:

invoice:

part number: 01234 (seat)

quantity: 2

price per item: \$375.00

payment due: \$750.00

invoice:

part number: 56789 (tire)

quantity: 4

price per item: \$79.95

payment due: \$319.80

salaried employee: John Smith

social security number: 111-11-1111

weekly salary: \$800.00

payment due: \$800.00

salaried employee: Lisa Barnes

social security number: 888-88-8888

weekly salary: \$1,200.00

payment due: \$1,200.00

## Tóm tắt bài học

- ❖ **this** đại diện cho chính đối tượng đang xây dựng, **super** đại diện cho lớp đối tượng cha (trong quan hệ kế thừa)
- ❖ Kế thừa giúp xây dựng lớp mới một cách tiện lợi, sử dụng từ khóa **extends** phạm vi kế thừa luôn là public
- ❖ Abstract class là lớp trừu tượng, không thể phát sinh đối tượng, có vai trò là lớp “chỉ đường” cho việc xây dựng các lớp con
- ❖ Java chỉ có đơn kế thừa lớp, muốn sử dụng đa kế thừa thì sử dụng **interface**
- ❖ Interface chứa các thuộc tính public, static và final, phương thức thì public, abstract