Chương 2

ĐẶC ĐIỂM CƠ BẢN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Các khái niệm cơ bản

❖ **Đối tượng (object)**: trong thế giới thực khái niệm đối tượng có thể xem như một thực thể: người, vật, bảng dữ liệu,…

- ✓ Đối tượng giúp hiểu rõ thế giới thực
- ✓ Cơ sở cho việc cài đặt trên máy tính
- ✓ Mỗi đối tượng có định danh, thuộc tính, hành vi

Ví dụ: đối tượng sinh viên

MSSV: "TH0701001"; Tên sinh viên: "Nguyễn Văn A"

❖ **Hệ thống các đối tượng**: là 1 tập hợp các đối tượng

- ✓ Mỗi đối tượng đảm trách 1 công việc
- ✓ Các đối tượng có thể quan hệ với nhau
- ✓ Các đối tượng có thể trao đổi thông tin với nhau
- ✓ Các đối tượng có thể xử lý song song, hay phân tán

Các khái niệm cơ bản

❖ **Lớp (class)**: là khuôn mẫu (template) để sinh ra đối tượng. Lớp là sự trừu tượng hóa của tập các đối tượng có các thuộc tính, hành vi tương tự nhau, và được gom chung lại thành 1 lớp.

Ví dụ: lớp các đối tượng ***Sinhviên***

- ✓Sinh viên "Nguyễn Văn A", mã số TH0701001 → 1 đối tượng thuộc lớp ***Sinhviên***
- ✓Sinh viên "Nguyễn Văn B", mã số TH0701002 → là 1 đối tượng thuộc lớp ***Sinhviên***

❖ **Đối tượng (object) của lớp**: một đối tượng cụ thể thuộc 1 lớp là 1 thể hiện cụ thể của 1 lớp đó.

Lớp và đối tượng trong java

- **Khai báo lớp**

class *<ClassName>*

{

<danh sách thuộc tính>

<các khởi tạo>

<danh sách các phươngthức>

}

Lớp và đối tượng trong java

❖ **Thuộc tính:** các đặc điểm mang giá trị của đối tượng, là vùng dữ liệu được khai báo bên trong lớp

```
class <ClassName>     {
        <Tiền tố> <kiểu dữ liệu> <tên thuộc tính>;
}
```

Kiểm soát truy cập đối với thuộc tính

* **public**: có thể truy xuất từ bất kỳ 1 lớp khác.
* **protected**: có thể truy xuất được từ những lớp con.
* **private**: không thể truy xuất từ 1 lớp khác.
* **static:** dùng chung cho mọi thể hiện của lớp.
* **final:** hằng
* **default:** (không phải từ khóa) có thể truy cập từ các class trong cùng gói

Lớp và đối tượng trong java

❖ **Phương thức:** chức năng xử lý, hành vi của các đối tượng.

```
class <ClassName>        {
        …
        <Tiền tố> <kiểu trả về> <tên phương thức>(<các đối số>){
                …
        }
}
```

Lớp và đối tượng trong java

* **public**: có thể truy cập được từ bên ngoài lớp khai báo.

* **protected**: có thể truy cập được từ lớp khai báo và các lớp dẫn xuất (lớp con).

* **private**: chỉ được truy cập bên trong lớp khai báo.

* **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có thể được thực hiện kể cả khi không có đối tượng của lớp

* **final**: không được khai báo chồng ở các lớp dẫn xuất.

* **abstract**: không có phần source code, sẽ được cài đặt trong các lớp dẫn xuất.

* **synchoronized**: dùng để ngăn những tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình miltithreads.

Lớp và đối tượng trong java

Ví dụ 1: class *Sinhvien {*

```
// Danh sách thuộc tính
String    maSv, tenSv, dcLienlac;
int       tuoi;
…
// Danh sách các khởi tạo
Sinhvien(){}
Sinhvien (…)      { …}
…
// Danh sách các phương thức
public void capnhatSV (…) {…}
public void xemThongTinSV() {…}
…
}
```

Lớp và đối tượng trong java

```
…
// Tạo đối tượng mới thuộc lớp Sinhvien
Sinhvien sv = new Sinhvien();
…
// Gán giá trị cho thuộc tính của đối tượng
sv.maSv = “TH0601001” ;
sv.tenSv = “Nguyen Van A”;
sv.tuoi = “20”;
sv.dcLienlac = “KP6, Linh Trung, Thu Duc”;
…
// Gọi thực hiện phương thức
sv.xemThongTinSV();
```

Lớp và đối tượng trong java

Ví dụ 2:

```
class Sinhvien {
    // Danh sách thuộc tính
    private String      maSv;
    String    tenSv, dcLienlac;
    int         tuoi;
    …
}
…
Sinhvien sv = new Sinhvien();
sv.maSv = "TH0601001"; /* Lỗi truy cập thuộc tính private từ
                          bên ngoài lớp khai báo */
Sv.tenSv = "Nguyen Van A";
…
```

Lớp và đối tượng trong java

❖ **Khởi tạo (constructor):** là một loại phương thức đặc biệt của lớp, dùng để khởi tạo một đối tượng.

- ✓ Dùng để khởi tạo giá trị cho các thuộc tính của đối tượng.
- ✓ Cùng tên với lớp.
- ✓ Không có giá trị trả về.
- ✓ Tự động thi hành khi tạo ra đối tượng (new)
- ✓Có thể có tham số hoặc không.

❖ **Lưu ý**: Mỗi lớp sẽ có 1 constructor mặc định (nếu ta không khai báo constructor nào). Ngược lại nếu ta có khai báo 1 constructor khác thì constructor mặc định chỉ dùng được khi khai báo tường minh.

Lớp và đối tượng trong java

- ***Ví dụ 1***

```
class Sinhvien
{
        ...
        // Không có định nghĩa constructor nào
}
...
// Dùng constructor mặc định
Sinhvien sv = new Sinhvien();
```

Lớp và đối tượng trong java

Ví dụ 2:

class Sinhvien

{

…

// không có constructor mặc định

Sinhvien(<các đối số>) {…}

}

…

Sinhvien sv = new Sinhvien();

// lỗi biên dịch

class Sinhvien

{

…

// khai báo constructor mặc định

Sinhvien(){}

Sinhvien(<các đối số>) {…}

}

…

Sinhvien sv = new Sinhvien();

Lớp và đối tượng trong java

❖ **Overloading method:** Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức.

Ví dụ:

```
class Sinhvien {
        ...
        public void xemThongTinSV() {
                ...
        }
        public void xemThongTinSV(String psMaSv) {
                ...
        }
}
```

Lớp và đối tượng trong java

❖ **Tham chiếu *this:*** là một biến ẩn tồn tại trong tất cả các lớp, ***this*** được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

Ví dụ:

```
class Sinhvien    {
        String    maSv, tenSv, dcLienlac;
        int       tuoi;
        …
        public void xemThongTinSV() {
                System.out.println(this.maSv);
                System.out.println(this.tenSv);
                …
        }
}
```

Tính đóng gói

❖ **Đóng gói:** nhóm những gì có liên quan với nhau vào thành một và có thể sử dụng một cái tên để gọi.

Ví dụ:

✓ Các phương thức đóng gói các câu lệnh.

✓ Đối tượng đóng gói dữ liệu và các hành vi/phương thức liên quan.

(Đối tượng = Dữ liệu + Hành vi/Phương thức)

Tính đóng gói

❖ **Đóng gói:** dùng để che dấu một phần hoặc tất cả thông tin, chi tiết cài đặt bên trong với bên ngoài.

- Ví dụ: khai báo các lớp thuộc cùng gói trong java

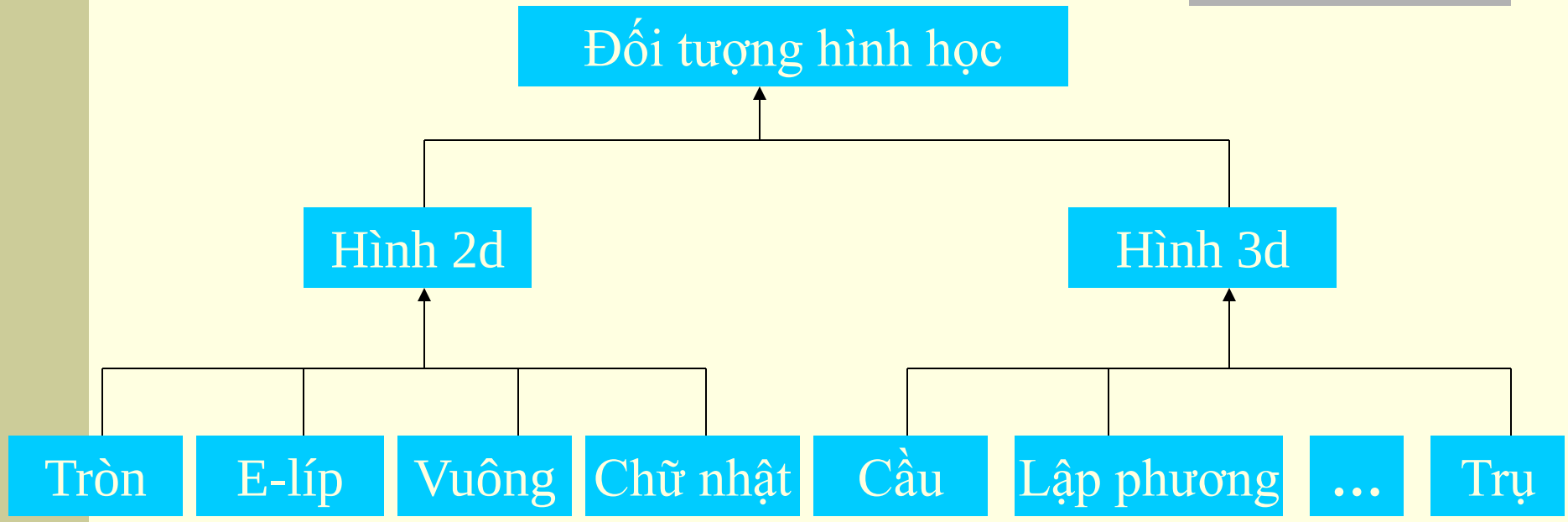
package *<tên gói>; // khai báo trước khi khai báo lớp*

class *<tên lớp> {*

...

}

Tính kế thừa



- Thừa hưởng các thuộc tính và phương thức đã có
- Bổ sung, chi tiết hóa cho phù hợp với mục đích sử dụng mới
  - ✓ **Thuộc tính**: thêm mới
  - ✓ **Phương thức**: thêm mới hay hiệu chỉnh

Tính kế thừa

- ✓ **Lớp dẫn xuất hay lớp con (SubClass)**
- ✓ **Lớp cơ sở hay lớp cha (SuperClass)**
- ✓ 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***.

Ví dụ: *class nguoi { …*

}

class sinhvien ***extends*** *nguoi { …*

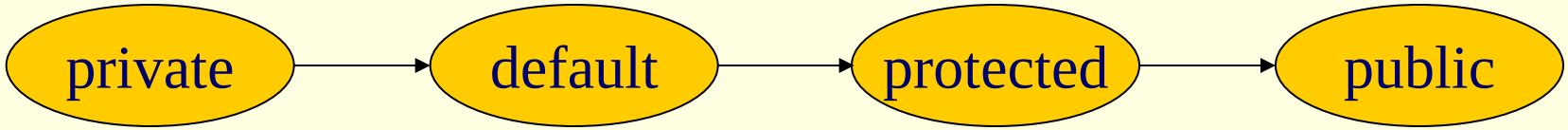
}

Lưu ý: default không phải là 1 từ khóa

Tính kế thừa

❖ **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 k0 "nhỏ hơn" phương thức trong lớp cha



Tính kế thừa

- ***Ví dụ:*** *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***

Tính kế thừa

```
class HinhChuNhat extends HinhVuong {
        private int cd;
        private int cr;
        public float tinhdientich() {
                return cd*cr;
        }
        …
}
```

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*

```
clone
equals
finalize
getClass
hashCode
notify, notifyAll, wait
toString
```

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.*

```
1  // Fig. 9.15: CommissionEmployee4.java
2  // CommissionEmployee4 class represents a commission employee.
3
4  public class CommissionEmployee
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private double grossSales; // gross weekly sales
10    private double commissionRate; // commission percentage
11
12    // five-argument constructor
13    public CommissionEmployee( String first, String last, String ssn,
14       double sales, double rate )
15    {
16       // implicit call to Object constructor occurs here
17       firstName = first;
18       lastName = last;
19       socialSecurityNumber = ssn;
20       setGrossSales( sales ); // validate ar
21       setCommissionRate( rate ); // validate
22
23       System.out.printf(
24          "\nCommissionEmployee4 constructor:\n%s\n", this );
25    } // end five-argument CommissionEmployee4 constructor
26
```

Constructor outputs message to demonstrate method call order.

CommissionEmployee4.java

- (1 of 4)
- Lines 23-24

```
27    // set first name
28    public void setFirstName( String first )
29    {
30       firstName = first;
31    } // end method setFirstName
32
33    // return first name
34    public String getFirstName()
35    {
36       return firstName;
37    } // end method getFirstName
38
39    // set last name
40    public void setLastName( String last )
41    {
42       lastName = last;
43    } // end method setLastName
44
45    // return last name
46    public String getLastName()
47    {
48       return lastName;
49    } // end method getLastName
50
51    // set social security number
52    public void setSocialSecurityNumber( String ssn )
53    {
54       socialSecurityNumber = ssn; // should validate
55    } // end method setSocialSecurityNumber
56
```

CommissionEmployee4.java

- (2 of 4)

```
57    // return social security number
58    public String getSocialSecurityNumber()
59    {
60       return socialSecurityNumber;
61    } // end method getSocialSecurityNumber
62
63    // set gross sales amount
64    public void setGrossSales( double sales )
65    {
66       grossSales = ( sales < 0.0 ) ? 0.0 : sales;
67    } // end method setGrossSales
68
69    // return gross sales amount
70    public double getGrossSales()
71    {
72       return grossSales;
73    } // end method getGrossSales
74
75    // set commission rate
76    public void setCommissionRate( double rate )
77    {
78       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
79    } // end method setCommissionRate
80
```

CommissionEmployee4.java

- (3 of 4)

```
81    // return commission rate
82    public double getCommissionRate()
83    {
84       return commissionRate;
85    } // end method getCommissionRate
86
87    // calculate earnings
88    public double earnings()
89    {
90       return getCommissionRate() * getGrossSales();
91    } // end method earnings
92
93    // return String representation of CommissionEmployee4 object
94    public String toString()
95    {
96       return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
97          "commission employee", getFirstName(), getLastName(),
98          "social security number", getSocialSecurityNumber(),
99          "gross sales", getGrossSales(),
100         "commission rate", getCommissionRate() );
101    } // end method toString
102 } // end class CommissionEmployee4
```

CommissionEmployee4.java

- (4 of 4)

```
1  // Fig. 9.16: BasePlusCommissionEmployee5.java
2  // BasePlusCommissionEmployee5 class declaration.
3
4  public class BasePlusCommissionEmployee5 extends CommissionEmployee4
5  {
6     private double baseSalary; // base salary per week
7
8     // six-argument constructor
9     public BasePlusCommissionEmployee5( String first, String last,
10       String ssn, double sales, double rate, double salary )
11    {
12       super( first, last, ssn, sales, rate );
13       setBaseSalary( salary ); // validate a
14
15       System.out.printf(
16          "\nBasePlusCommissionEmployee5 constructor:\n%s\n", this );
17    } // end six-argument BasePlusCommissionEmployee5 constructor
18
19    // set base salary
20    public void setBaseSalary( double salary )
21    {
22       baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23    } // end method setBaseSalary
24
```

Constructor outputs message to demonstrate method call order.

BasePlusCommissionEmployee5.java

- (1 of 2)
- Lines 15-16

```
25    // return base salary
26    public double getBaseSalary()
27    {
28       return baseSalary;
29    } // end method getBaseSalary
30
31    // calculate earnings
32    public double earnings()
33    {
34       return getBaseSalary() + super.earnings();
35    } // end method earnings
36
37    // return String representation of BasePlusCommissionEmployee5
38    public String toString()
39    {
40       return String.format( "%s %s\n%s: %.2f", "base-salaried",
41          super.toString(), "base salary", getBaseSalary() );
42    } // end method toString
43 } // end class BasePlusCommissionEmployee5
```

BasePlusCommissionEmployee5.java

- (2 of 2)

```
1  // Fig. 9.17: ConstructorTest.java
2  // Display order in which superclass and subclass constructors are called.
3
4  public class ConstructorTest
5  {
6     public static void main( String args[] )
7     {
8        CommissionEmployee4 employee1 = new CommissionEmployee4(
9           "Bob", "Lewis", "333-33-3333", 5000, .04 );
10
11       System.out.println();
12       BasePlusCommissionEmployee5 employee2 =
13          new BasePlusCommissionEmployee5(
14          "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
15
16       System.out.println();
17       BasePlusCommissionEmployee5 employee3 =
18          new BasePlusCommissionEmployee5(
19          "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
20    } // end main
21 } // end class ConstructorTest
```

Instantiate `CommissionEmployee4` object

Instantiate two `BasePlusCommissionEmployee5` objects to demonstrate order of subclass and superclass constructor method calls.

Construc torTest java

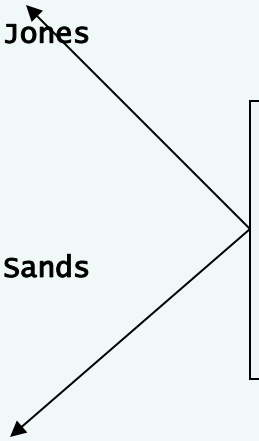
```
CommissionEmployee4 constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee4 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00


CommissionEmployee4 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00
```



Subclass `BasePlusCommissionEmployee5` constructor body executes after superclass `CommissionEmployee4`'s constructor finishes execution.

- ConstructorTest.java

Tính kế thừa

- **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 {
    ...
}
```

Tính đa hình

❖ **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.

Tính đa hình

❖ **Lớp trừu tượng:** là lớp dùng để thể hiện sự trừu tượng hóa ở mức cao.

Ví dụ: lớp "Đối tượng hình học", "Hình 2D", "Hình 3D"

(Ví dụ định nghĩa lớp các đối tượng hình học cơ bản)

❖ **Từ khóa abstract: để khai báo một lớp abstract**.

❖Lớp abstract không thể tạo ra đối tượng.

Giao tiếp - interface

❖ **Interface:** giao tiếp của một lớp**, là phần đặc tả** (không có phần cài đặt cụ thể) của lớp, nó chứa các khai báo phương thức và thuộc tính để bên ngoài có thể truy xuất được. (java, C#, …)

- ✓ Lớp sẽ cài đặt các phương thức trong interface.
- ✓ Trong lập trình hiện đại các đối tượng không đưa ra cách truy cập cho một lớp, thay vào đó cung cấp các interface. Người lập trình dựa vào interface để gọi các dịch vụ mà lớp cung cấp.
- ✓ Thuộc tính của interface là các hằng và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

Giao tiếp - interface

Ví dụ:

```
// Định nghĩa một interface Shape trong tập tin shape.java
public interface Shape {
        // Tính diện tích
        public abstract double area();
        // Tính thể tích
        public abstract double volume();
        // trả về tên của shape
        public abstract String getName();
}
```

Giao tiếp - interface

// Lớp *Point* cài đặt/hiện thực *interface* tên *shape*.

```
// Định nghĩa lớp Point trong tập tin Point.java
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 );
    }
```

Giao tiếp - 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;
}
```

Giao tiếp - interface

```
// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
public String toString() {
        return "[" + x + ", " + y + "]";
}
// Tính diện tích
public double area() {
        return 0.0;
}
// Tính thể tích
public double volume() {
        return 0.0;
}
```

Giao tiếp - interface

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

Giao tiếp - interface

❖ **Kế thừa interface**

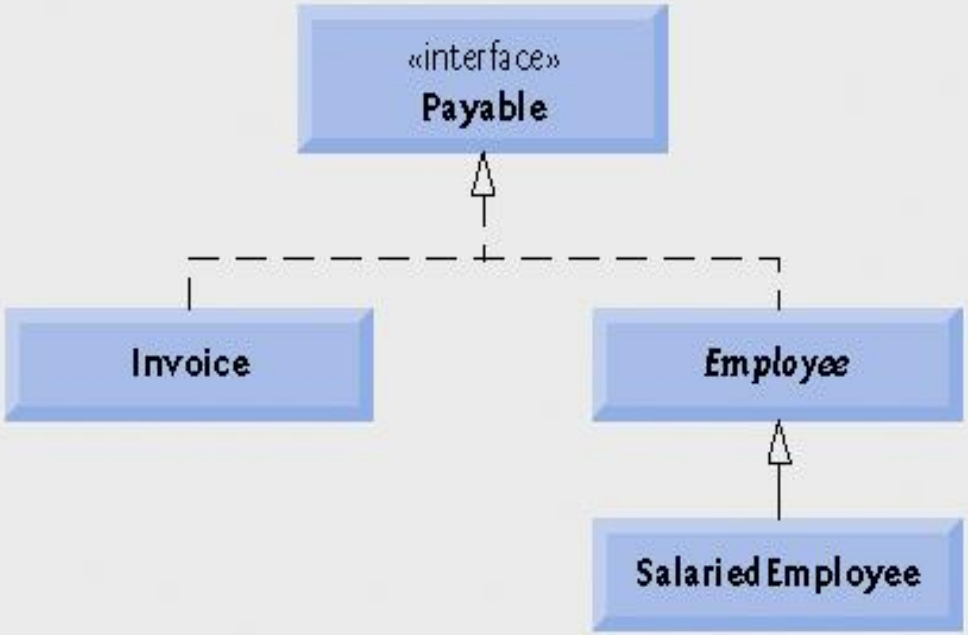
public interface *InterfaceName* ***extends*** *interface1, interface2, interface3*

{

// ...

}

Ví dụ



Outline

- Payable.java

```
1  // Fig. 10.11: Payable.java
2  // Payable interface declaration.
3
4  public interface Payable
5  {
6     double getPaymentAmount(); // calculate payment; no implementation
7  } // end interface Payable
```

Declare interface `Payable`

Declare `getPaymentAmount` method which is implicitly `public` and `abstract`

Outline

- Invoice.java

```
1  // Fig. 10.12: Invoice.java
2  // Invoice class implements Payable.
3
4  public class Invoice implements Payable
5  {
6     private String partNumber;
7     private String partDescription;
8     private int quantity;
9     private double pricePerItem;
10
11    // four-argument constructor
12    public Invoice( String part, String description, int count,
13       double price )
14    {
15       partNumber = part;
16       partDescription = description;
17       setQuantity( count ); // validate and store quantity
18       setPricePerItem( price ); // validate and store price per item
19    } // end four-argument Invoice constructor
20
21    // set part number
22    public void setPartNumber( String part )
23    {
24       partNumber = part;
25    } // end method setPartNumber
26
```

Class `Invoice` implements interface `Payable`

- (1 of 3)

Outline

- Invoice.java
- (2 of 3)

```
27    // get part number
28    public String getPartNumber()
29    {
30       return partNumber;
31    } // end method getPartNumber
32
33    // set description
34    public void setPartDescription( String description )
35    {
36       partDescription = description;
37    } // end method setPartDescription
38
39    // get description
40    public String getPartDescription()
41    {
42       return partDescription;
43    } // end method getPartDescription
44
45    // set quantity
46    public void setQuantity( int count )
47    {
48       quantity = ( count < 0 ) ? 0 : count; // quantity cannot be negative
49    } // end method setQuantity
50
51    // get quantity
52    public int getQuantity()
53    {
54       return quantity;
55    } // end method getQuantity
56
```

Outline

- Invoice.java
- (3 of 3)

```
57    // set price per item
58    public void setPricePerItem( double price )
59    {
60       pricePerItem = ( price < 0.0 ) ? 0.0 : price; // validate price
61    } // end method setPricePerItem
62
63    // get price per item
64    public double getPricePerItem()
65    {
66       return pricePerItem;
67    } // end method getPricePerItem
68
69    // return String representation of Invoice object
70    public String toString()
71    {
72       return String.format( "%s: \n%s: %s (%s) \n%s: %d \n%s: $%,.2f",
73          "invoice", "part number", getPartNumber(), getPartDescription(),
74          "quantity", getQuantity(), "price per item", getPricePerItem() );
75    } // end method toString
76
77    // method required to carry out contract with interface Payable
78    public double getPaymentAmount()
79    {
80       return getQuantity() * getPricePerItem(); // calculate total cost
81    } // end method getPaymentAmount
82 } // end class Invoice
```

Declare `getPaymentAmount` to fulfill contract with interface `Payable`

Outline

- Employee.java
- (1 of 3)

```
1  // Fig. 10.13: Employee.java
2  // Employee abstract superclass implements Payable.
3
4  public abstract class Employee implements Payable
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9
10    // three-argument constructor
11    public Employee( String first, String last, String ssn )
12    {
13       firstName = first;
14       lastName = last;
15       socialSecurityNumber = ssn;
16    } // end three-argument Employee constructor
17
```

Class `Employee` implements interface `Payable`

```
18    // set first name
19    public void setFirstName( String first )
20    {
21       firstName = first;
22    } // end method setFirstName
23
24    // return first name
25    public String getFirstName()
26    {
27       return firstName;
28    } // end method getFirstName
29
30    // set last name
31    public void setLastName( String last )
32    {
33       lastName = last;
34    } // end method setLastName
35
36    // return last name
37    public String getLastName()
38    {
39       return lastName;
40    } // end method getLastName
41
```

Outline

Employee.java

- (2 of 3)

Outline

Employee.java

- (3 of 3)

```
42    // set social security number
43    public void setSocialSecurityNumber( String ssn )
44    {
45       socialSecurityNumber = ssn; // should validate
46    } // end method setSocialSecurityNumber
47
48    // return social security number
49    public String getSocialSecurityNumber()
50    {
51       return socialSecurityNumber;
52    } // end method getSocialSecurityNumber
53
54    // return String representation of Employee object
55    public String toString()
56    {
57       return String.format( "%s %s\nsocial security number: %s",
58          getFirstName(), getLastName(), getSocialSecurityNumber() );
59    } // end method toString
60
61    // Note: We do not implement Payable method getPaymentAmount here so
62    // this class must be declared abstract to avoid a compilation error.
63 } // end abstract class Employee
```

`getPaymentAmount` method is not implemented here

```
1  // Fig. 10.14: SalariedEmployee.java
2  // SalariedEmployee class extends Employee, which implements Payable.
3
4  public class SalariedEmployee extends Employee
5  {
6     private double weeklySalary;
7
8     // four-argument constructor
9     public SalariedEmployee( String first, String last, String ssn,
10       double salary )
11    {
12       super( first, last, ssn ); // pass to Employee constructor
13       setWeeklySalary( salary ); // validate and store salary
14    } // end four-argument SalariedEmployee constructor
15
16    // set salary
17    public void setWeeklySalary( double salary )
18    {
19       weeklySalary = salary < 0.0 ? 0.0 : salary;
20    } // end method setWeeklySalary
21
```

Class `SalariedEmployee` extends class `Employee` (which implements interface `Payable`)

Outline

- SalariedEmployee.java
- (1 of 2)

Outline

- Salaried Employe
- .java
- (2 of 2)

```
22    // return salary
23    public double getWeeklySalary()
24    {
25       return weeklySalary;
26    } // end method getWeeklySalary
27
28    // calculate earnings; implement interface Payable method that was
29    // abstract in superclass Employee
30    public double getPaymentAmount()
31    {
32       return getWeeklySalary();
33    } // end method getPaymentAmount
34
35    // return String representation of SalariedEmployee object
36    public String toString()
37    {
38       return String.format( "salaried employee: %s\n%s: $%,.2f",
39          super.toString(), "weekly salary", getWeeklySalary() );
40    } // end method toString
41 } // end class SalariedEmployee
```

Declare `getPaymentAmount` method instead of `earnings` method

Outline

PayableInterface a

```
1  // Fig. 10.15: PayableInterfaceTest.java
2  // Tests interface Payable.
3
4  public class PayableInterfaceTest
5  {
6     public static void main( String args[] )
7     {
8        // create four-element Payable array
9        Payable payableObjects[] = new Payable[ 4 ];
10
11       // populate array with objects that implement Payable
12       payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
13       payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
14       payableObjects[ 2 ] =
15          new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
16       payableObjects[ 3 ] =
17          new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );
18
19       System.out.println(
20          "Invoices and Employees processed polymorphically:\n" );
21
```

Declare array of `Payable` variables

Assigning references to `Invoice` objects to `Payable` variables

Assigning references to `SalariedEmployee` objects to `Payable` variables

(1 of 2)

```
22       // generically process each element in array payableObjects
23       for ( Payable currentPayable : payableObjects )
24       {
25          // output currentPayable and its appropriate payment amount
26          System.out.printf( "%s \n%s: $%,.2f\n\n",
27             currentPayable.toString(),
28             "payment due", currentPayable.getPaymentAmount() );
29       } // end for
30    } // end main
31 } // end class PayableInterfaceTest
```

Call `toString` and `getPaymentAmount` methods polymorphically

```
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
```

Outline

- PayableI
  nterface
  Test.java
- (2 of 2)

Quan hệ giữa Class và Interface

	Class	Interface
Class	extends	implements
Interface		extends