Bài 3: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Nội dung

- Các khái niệm cơ bản
- Lớp và đối tượng trong Java
- Đặc điểm 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):**

```
class <ClassName>
{
    <danh sách thuộc tính>
    <các khởi tạo>
    <danh sách các phươngthức>
}
```

Lớp và đối tượng trong Java (tt)

- **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:** cho phép bản thân lớp đó và những lớp dẫn xuất từ lớp đó truy cập đến.
  - **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 (packed)

Lớp và đối tượng trong Java (tt)

- **Phương thức:** chức năng xử ly, 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 (tt)

- **Tiền tố:**
  - **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 nạp 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 tượngkhác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình multithreads.

Lớp và đối tượng trong Java (tt)

- <*kiểu trả về*>: có thể là kiểu **void**, **kiểu cơ sở** hay **một lớp**.
- <*Tên phương thức*>: đặt theo qui ước giống tên biến.
- <*các đối số*>: có thể rỗng

Lớp và đối tượng trong Java (tt)

➢ ***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 (tt)


```
...
    // 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 (tt)

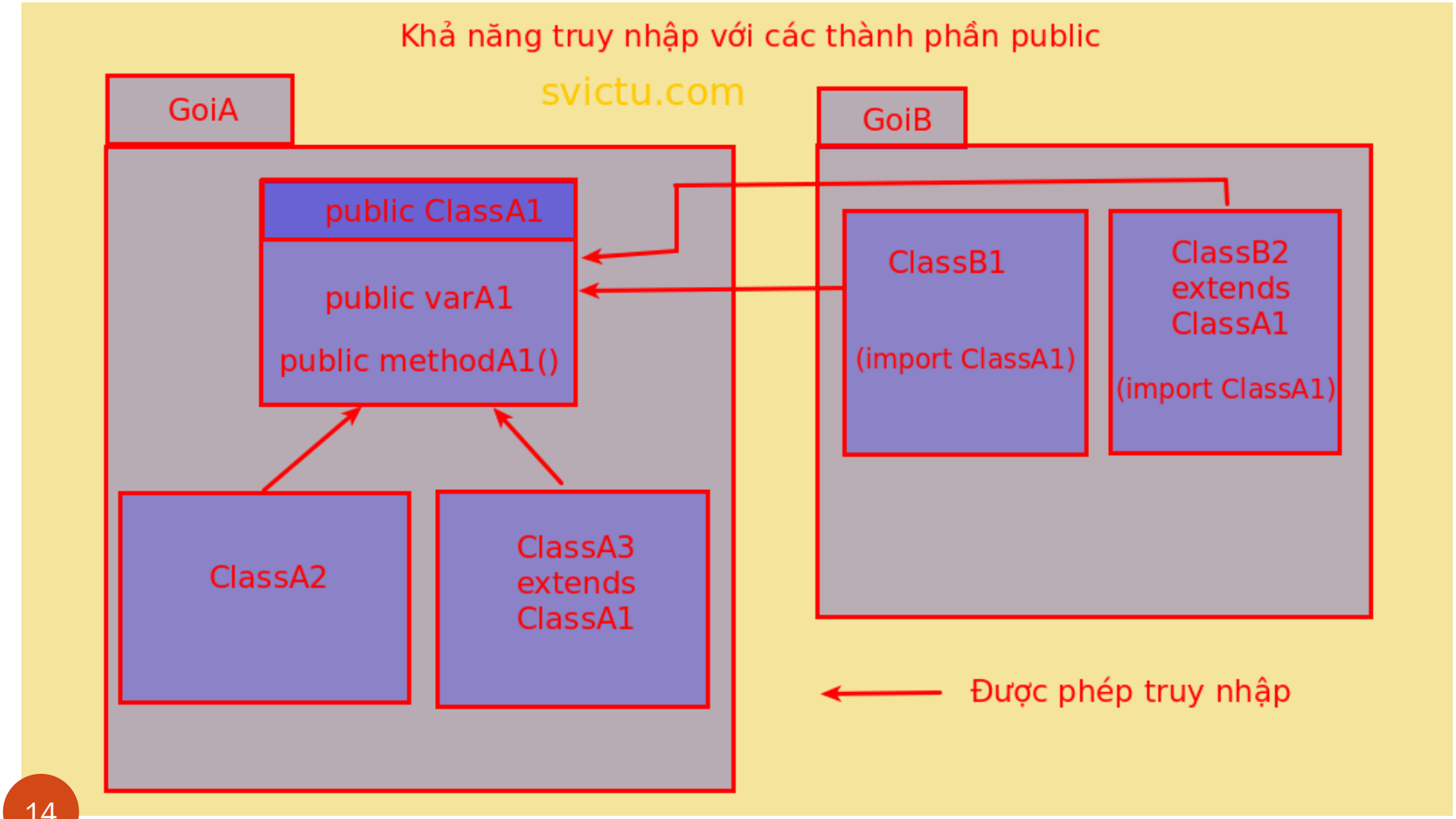
➢ ***Ví dụ 2:***

```
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";
…
```

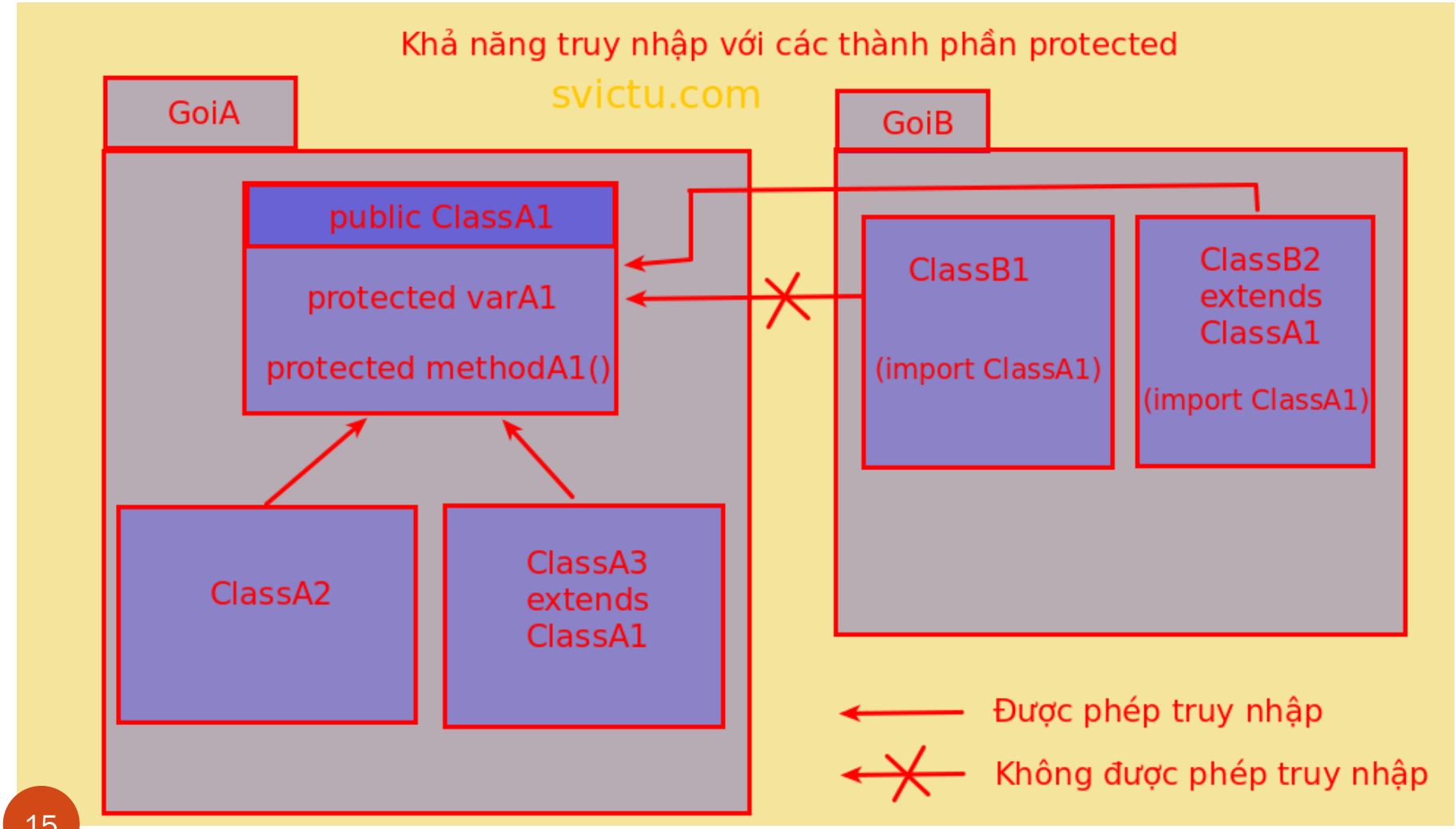
Các mức truy cập trong Java

- Để bảo vệ dữ liệu tránh bị truy nhập tự do từ bên ngoài, Java sử dụng các từ khoá quy định phạm vi truy nhập các thuộc tính và phương thức của lớp:
  - **public:** Thành phần công khai, truy nhập tự do từ bên ngoài.
  - **protected**: Thành phần được bảo vệ, được hạn chế truy nhập.
  - **default** (không viết gì): Truy nhập trong nội bộ gói.
  - **private**: Truy nhập trong nội bộ lớp.

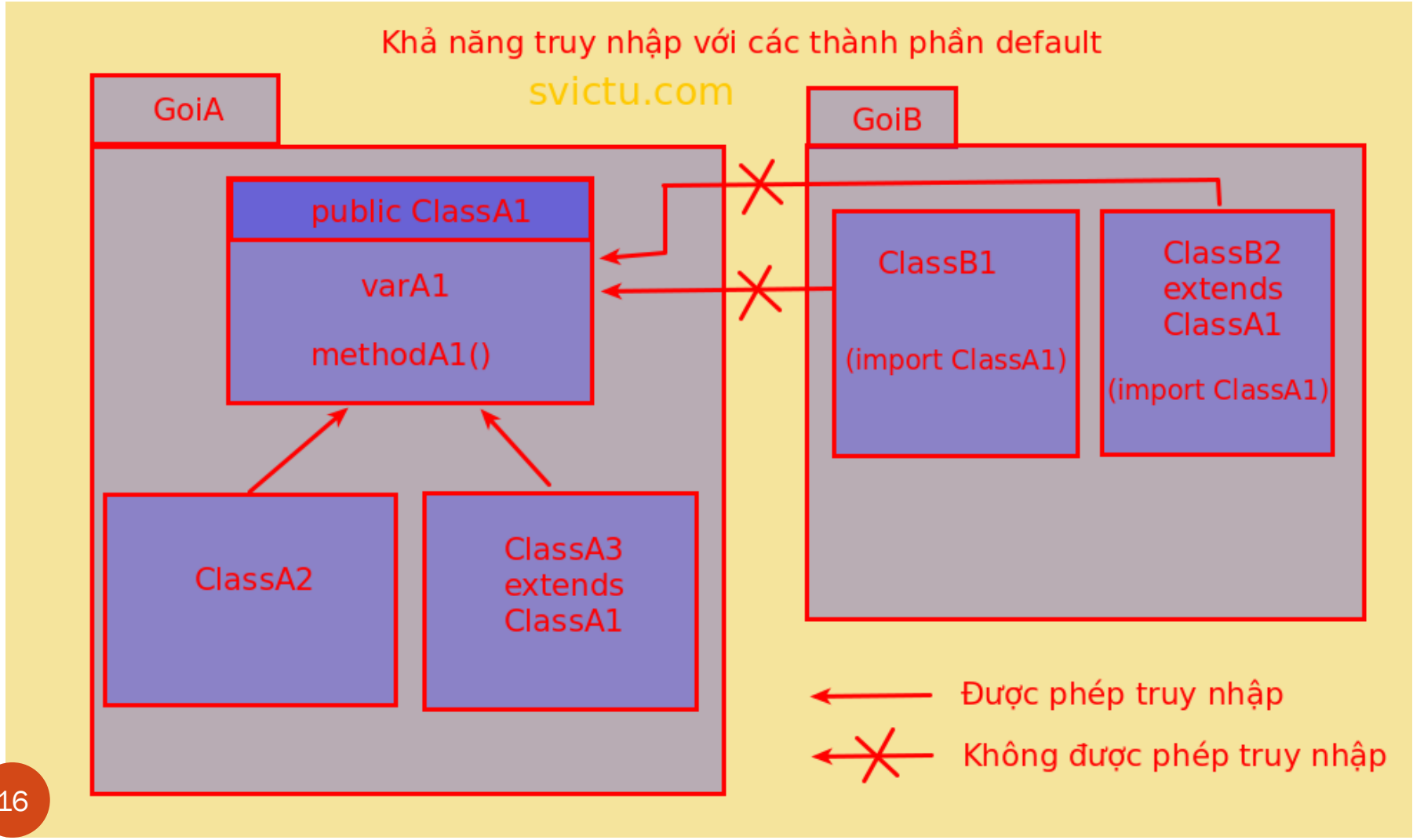
Các mức truy cập trong Java - public



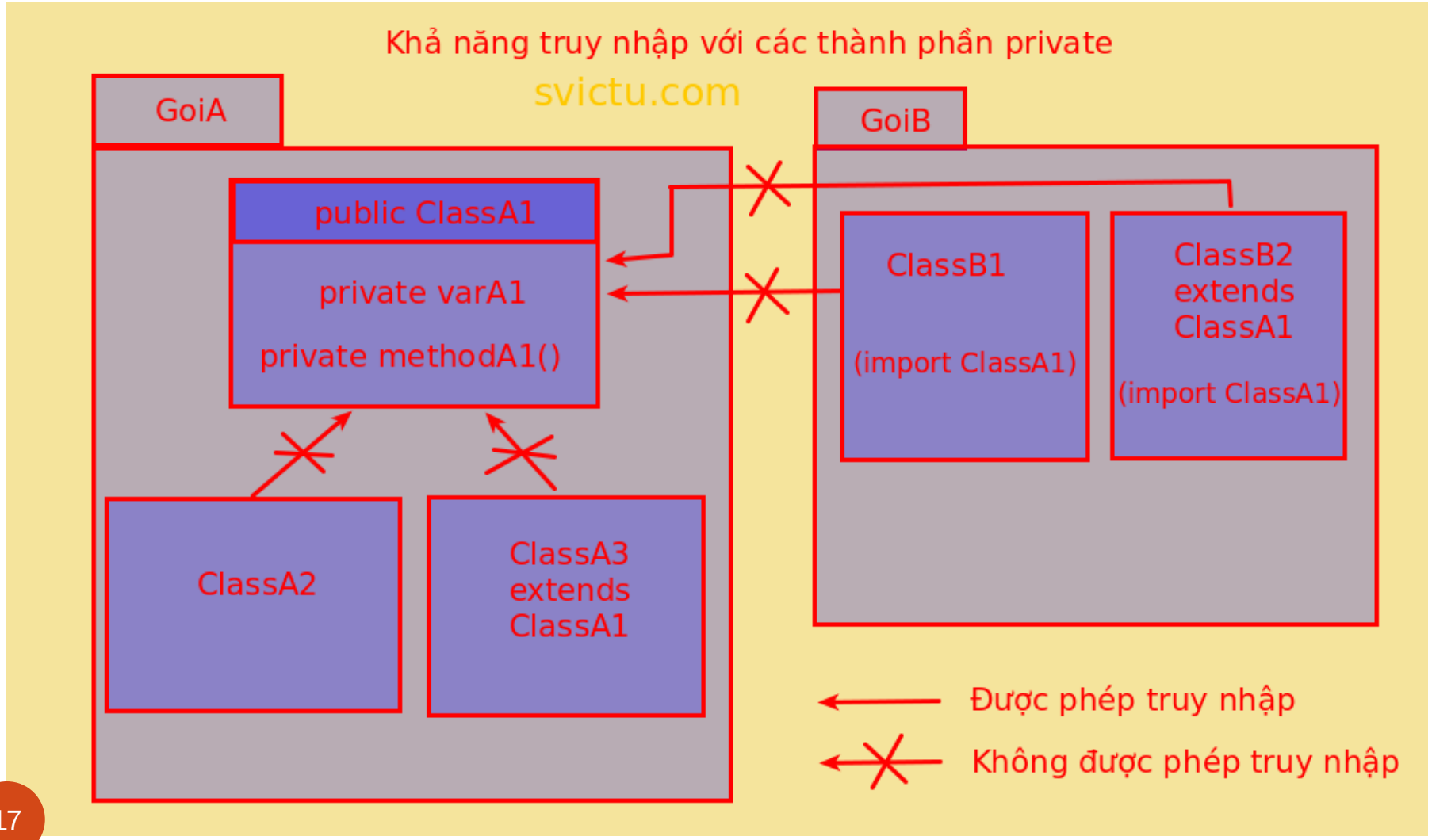
Các mức truy cập trong Java - protect



Các mức truy cập trong Java - default



Các mức truy cập trong Java - private



Các mức truy cập trong Java - tổng hợp

Tóm tắt nội dung: Giới thiệu các access modifier trong Java

Nội dung:

Access modifier là phạm vi truy cập của một class, một thuộc tính, một phương thức. Và có 4 loại:

- Public
- Protected
- Default (pravite-package)
- Private

Đặc tính truy xuất của 4 loại modifier đc thể hiện như sau:

	Private	Default	Protected	Public
Cùng class	Yes	Yes	Yes	Yes
Cùng package, khác class	No	Yes	Yes	Yes
Class con trong cùng package với class cha	No	Yes	Yes	Yes
Khác package, khác class	No	No	No	Yes
Class con khác package với class cha	No	No	Yes	Yes

Lưu ý: Đối với class thì chỉ có 2 Access modifier đó là puplic và default.

Lớp và đối tượng trong java (tt)

- **Hàm khởi tạo (hàm dựng hoặc 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.

Khai báo Constructor

- ***Ví dụ 1***

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

Khai báo Constructor (tt)

Ví dụ 2:

```
class Sinhvien{
    …
    // không có constructor mặc định
    // constructor có đối số
    Sinhvien(<các đối số>) {…}
}
…
Sinhvien sv = new Sinhvien();
// lỗi biên dịch
```

```
class Sinhvien{
    …
    // khai báo constructor mặc định
    Sinhvien(){}
   // constructor có đối số
      Sinhvien(<các đối số>) {…}
}
…
Sinhvien sv = new Sinhvien();
//Hoặc
Sinhvien sv = new Sinhvien (<các đối số>) ;
```

Ví dụ phương thức khởi tạo (Constructor)

```
package constructor;
class SinhVien {
private String Ten;
    public void In()
    {
       System.out.println("Ten:"+Ten);
    }
}
```

```
package constructor;
public class Constructor {
public static void main(String[] args) {
        SinhVien s= new SinhVien();
        s.In();
    }
}
```

Ten:null

Ví dụ phương thức khởi tạo (Constructor) (tt)

```
package constructor;
class SinhVien {
    private String Ten;
    public SinhVien()
    {
        Ten="Nguyen Van Tung";
    }
    public void In()
    {
        System.out.println("Ten:"+Ten);
    }
}
```

```
package constructor;
public class Constructor {
public static void main(String[] args) {
        SinhVien s= new SinhVien();
        s.In();
    }
}
```

Ten:Nguyen Van Tung

Ví dụ phương thức khởi tạo (Constructor) (tt)

```
package constructor;
class SinhVien {
    private String MSSV;
    private String Ten;
public SinhVien(){
        Ten="Nguyen Van Tung";
}
 public SinhVien(String str){
        Ten=str;
}
public void In(){
 System.out.println("Ten:"+Ten);
}
}
```

```
package constructor;
public class Constructor {
public static void main(String[] args) {
        SinhVien s= new SinhVien("Anh Thu");
        s.In();
    }
}
```

Ten: Anh Thu

Lớp và đối tượng trong java (tt)

- **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 (tt)

- **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 {
        private String      maSv, tenSv, dcLienlac;
        int   tuoi;
        …
        // Mutator Methods
        public void setmaSV(String maSv){
              this.maSv = maSv;
        }
        // Accessor Methods
        public String getmaSV(String (){
              return maSv;
        }
}
```

Đặc điểm hướng đối tượng trong java

- Tính đóng gói (Encapsulation)
- Tính kế thừa (Inheritance)
- Tính đa hình (Polymorphism)
- Lớp trừu tượng (Abstract)

Tính đóng gói (Encapsulation)

- **Đóng gói:**
  - Nhóm những gì có liên quan với nhau vào thành một và có thể sử dụng một cái tên để gọi.
  - Dùng để che dấu một phần hoặc tất cả thông tin, chi tiết cài đặt bên trong với bên ngoài.

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)

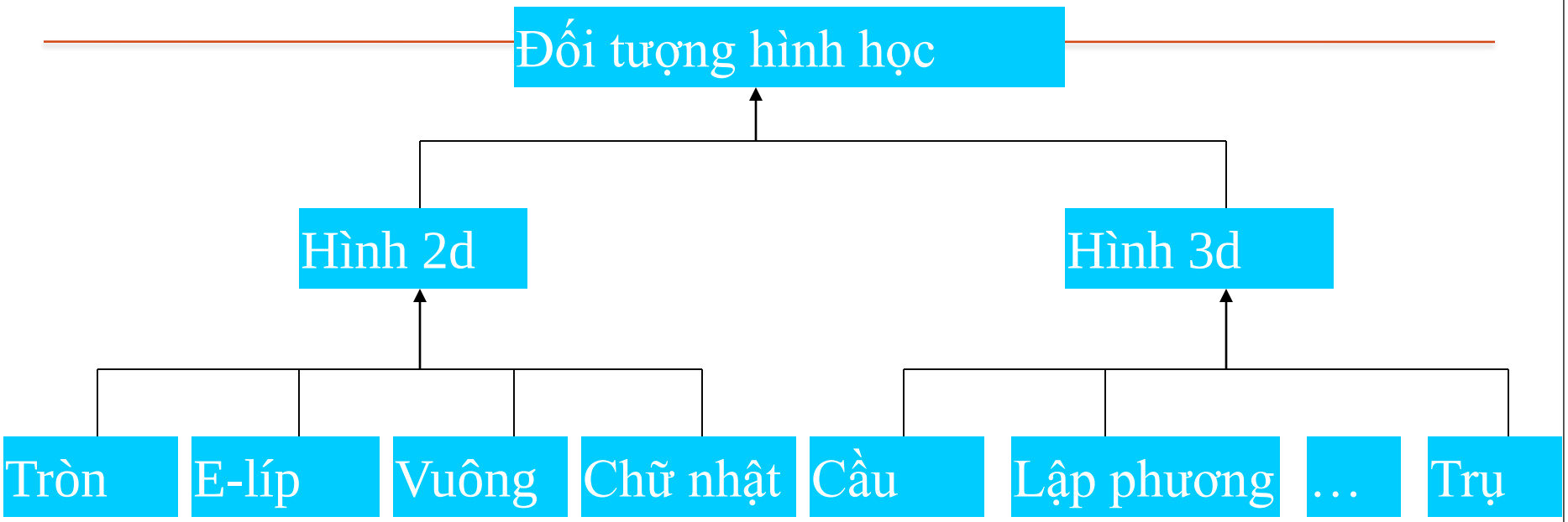
```
package  <tên gói>; // khai báo trước khi khai báo lớp
class <tên lớp> {
 …
}
```

Tính đóng gói (Encapsulation) (tt)

```
package VD1;
public class SinhVien {
	public void In(){
		System.out.println("Ten:");
	}
}
```

```
package VD2;
import VD1.SinhVien;
class VD{
public static void main(String[] args) {
		SinhVien sv= new SinhVien();
		sv.In();
	}
}
```

Tính kế thừa (Inheritance)



- 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

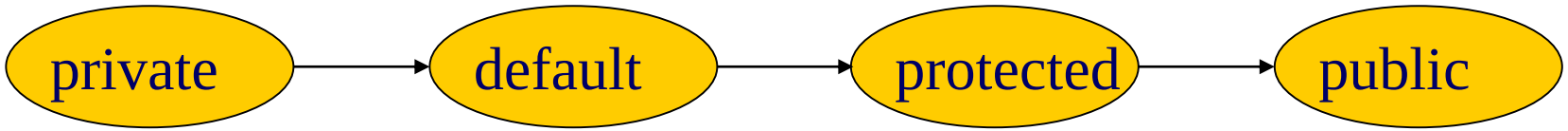
Tính kế thừa (Inheritance) (TT)

- **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 { …
}
```

Tính kế thừa (Inheritance) (TT)

- **Overriding Method (Nạp chồng phương thức)**
  - 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 không "nhỏ hơn" phương thức trong lớp cha



Tính kế thừa (Inheritance) (TT)

•***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 (Inheritance) (TT)

```
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

Từ khóa super

- Gọi **constructor** của lớp cha
- Nếu gọi tường minh thì phải là câu lệnh đầu tiên

```
class Nguoi {
    public Nguoi(String ten, int tuoi){
        …
    }
}
class SinhVien extends nguoi {
    public void show(){
        System.out.println(“....”);
        super(‘A’,20);
    }
}
```

Lỗi do super phải là câu lệnh đầu tiên

Sự thừa kế trong hàm khởi tạo
Constructor Inheritance

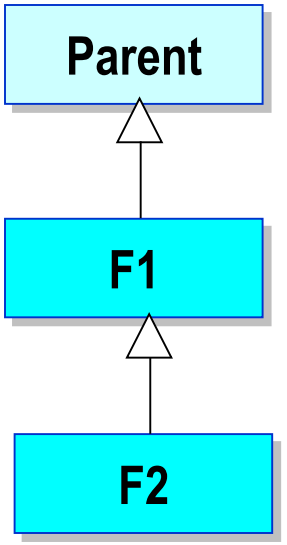
- Khai báo về thừa kế trong hàm khởi tạo
- Chuỗi các hàm khởi tạo (Constructor Chaining)
- Các nguyên tắc của hàm khởi tạo (Rules)
- Triệu hồi tường minh hàm khởi tạo của lớp cha

Sự thừa kế trong hàm khởi tạo

- Trong Java, hàm khởi tạo **không thể thừa kế** từ lớp cha như các loại phương thức khác.
- Khi tạo một thể hiện của lớp dẫn xuất, trước hết phải gọi đến hàm khởi tạo của lớp cha, tiếp đó mới là hàm khởi tạo của lớp con.
- Có thể triệu hồi hàm khởi tạo của lớp cha bằng cách sử dụng từ khóa super trong phần khai báo hàm khởi tạo của lớp con.

Chuỗi hàm khởi tạo - Constructor Chaining

```
class Parent{
	public Parent(){
		System.out.println("This is constructor of Parent class");
	}
}
class F1 extends Parent{
	public F1() {
		System.out.println("This is constructor of F1 class");
	}
}
class F2 extends F1{
	public F2(){
		System.out.println("This is constructor of F2 class");
	}
}
```



Chuỗi hàm khởi tạo (tt)

```
public static void main(String[] args) {
      F2 f1=new F2();
   }
```

```
This is constructor of Parent class
This is constructor of F1 class
This is constructor of F2 class
```

1. Object
2. Parent() call **super()**
3. F1() call **super ()**
4. F2() call **super()**
5. Main() call new F2()

- Khi tạo một thể hiện của lớp dẫn xuất, trước hết phải gọi đến hàm khởi tạo của lớp cha, tiếp đó là hàm khởi tạo của lớp con.

Các nguyên tắc của hàm khởi tạo

- Hàm khởi tạo mặc nhiên (default constructor) sẽ tự động sinh ra bởi trình biên dịch nếu lớp không khai báo hàm khởi tạo.
- Hàm khởi tạo mặc nhiên luôn luôn không có tham số (no-arg)
- Nếu trong lớp có định nghĩa hàm khởi tạo, hàm khởi tạo mặc nhiên sẽ không còn được sử dụng.
- Nếu không có lời gọi tường minh đến hàm khởi tạo của lớp cha tại lớp con, trình biên dịch sẽ tự động chèn lời gọi tới hàm khởi tạo mặc nhiên (implicity) hoặc hàm khởi tạo không tham số (explicity) của lớp cha trước khi thực thi đoạn code khác trong hàm khởi tạo lớp con.

Có 1 vấn đề?

```
public class Parent
{
    private int a;

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

Lỗi: Không tìm hàm khởi tạo của lớp Parent không có tham số

Sửa như thế nào?

```
public class Parent
{
    private int a;

    public Parent()
    {
        a = 0;
        System.out.println("Invoke parent default constructor");
    }

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

Sửa: Thêm hàm khởi tạo lớp Parent

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

Triệu hồi tường minh hàm khởi tạo lớp cha


```
public class Parent
{
    private int a;

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}

public class F1 extends Parent
{
    public F1(int value)
    {
        super (value);
        System.out.println("Invoke F1 default constructor");
    }
}
```

Không gọi tường minh hàm khởi tạo lớp cha ở lớp con


```
public class Parent
{
    private int a;

    public Parent()
    {
        a = 0;
        System.out.println("Invoke parent default constructor");
    }

    public Parent(int value)
    {
        a = value;
        System.out.println("Invoke parent parameter constructor");
    }
}
```

```
public class F1 extends Parent
{
    public F1()
    {
        System.out.println("Invoke F1 default constructor");
    }
}
```

```
public static void main(String[] args) {
    F1 f1=new F1();
}
```

Kết quả:
Invoke parent default constructor
Invoke F1default constructor

Ví dụ thừa kế (tt)

```
class Person {
    private String CMND;
    private String Name;
    private int age;
    public Person(String cm, String na, int a){
            CMND=cm;
            Name=na;
            age=a;
      }
    public void Print(){
        System.out.println("Chung minh"+"\t"+"Tên"+"\t"+"Tuoi");
        System.out.print(CMND+"\t"+Name+"\t"+age);
    }
```

Ví dụ thừa kế (tt)

```
class Employee extends Person {
    private double salary;
    public Employee(String cm, String na, int a, double sa){
        super(cm,na,a);
        salary=sa;
    }
    public void Print(){
        super.Print();
        System.out.print("Luong thang:"+salary);
    }
}
```

Ví dụ thừa kế (tt)

```
class Maneger extends Employee {
    private double allowance;
    public Maneger(String cm,String na,int a,double sa,double allow){
        super(cm,na,a,sa);
        allowance=allow;
    }
    public void Print(){
        super.Print();
        System.out.print("Phu cap:"+allowance);
    }
}
```

Ví dụ thừa kế (tt)

```
class Maneger extends Employee {
    private double allowance;
    public Maneger(String cm, String na, int a, double sa, double allow){
        super(cm,na,a,sa);
        allowance=allow;
    }
    public void Print(){
        super.Print();
        System.out.print("Phu cap:"+allowance);
    }
}
```

Ví dụ thừa kế (tt)

```
public class Nhanvien {
    public static void main(String[] args) {
        Person  p=new Person("1234", "NGuyen Huu Đat",23);
        Employee e=new Employee("2345","Tran Ngoc Tuan", 24,10000000);
        Maneger mng= new Maneger("3456", "Lê Văn Toàn",25,10000000,2000000);
        System.out.println("Thong tin nguoi:");
        p.Print();
        System.out.println();
        System.out.println("Thong nhan vien:");
        e.Print();
        System.out.println();
        System.out.print("Thong tin quan ly");
        mng.Print();
    }
}
```

Kết quả

Thong tin nguoi:

Chung minh Tên Tuoi

1234 NGuyen Huu Đat 23

Thong nhan vien:

Chung minh Tên Tuoi

2345 Tran Ngoc Tuan 24Luong thang:1.0E7

Thong tin quan lyChung minh Tên Tuoi

3456 Lê Văn Toàn 25Luong thang:1.0E7Phu cap:2000000.0

Biến, phương thức và lớp Final

- Biến Final - Final Variables
- Phương thức Final - Final Methods
- Lớp Final - Final Classes

Biến final

- Từ khóa "final" được sử dụng với biến để chỉ rằng giá trị của biến là hằng số.
- Hằng số là giá trị được gán cho biến vào thời điểm khai báo và sẽ không thay đổi về sau.

```
public final int MAX_COLS =100;
```

Phương thức hằng (Final)

- Được sử dụng để ngăn chặn việc ghi đè (override) hoặc che lấp (hidden) trong các lớp Java.
- Phương thức được khai báo là private hoặc là một thành phần của lớp final thì được xem là phương thức hằng.
- Phương thức hằng không thể khai báo là trừu tượng (abstract).

```
public final void find()
{
      //.....
}
```

Lớp hằng - Final Classes

- Là lớp không có lớp con (lớp vô sinh)
- Hay: là lớp không có kế thừa
- Được sử dụng để hạn chế việc thừa kế và ngăn chặn việc sửa đổi một lớp.

```
public final class Student {
    // ...
}
```

Tính đa hình

- **Đa hình**: Cùng một phương thức có thể có những cách thi hành khác nhau tại những thời điểm khác nhau. Trong Java tự động thể hiện tính đa hình
- **Abstract**: Lớp trừu tượng, hàm trừu tượng
- **Interface**: được cài đặt bởi các lớp con để triển khai các phương thức mà lớp muốn có.

Tính đa hình – Ví dụ

```
package tronvuong;
class Hinh {
    public void Ve(){
        System.out.println("Ve hinh");
    }
}
```

```
package tronvuong;
class HinhTron extends Hinh {
    public void Ve(){
        System.out.println("Ve tron");
    }
}
```

```
package tronvuong;
class HinhVuong extends Hinh{
    public void Ve(){
        System.out.println("Ve vuong");
    }
}
```

Tính đa hình – Ví dụ (tt)

```
package tronvuong;
public class TronVuong {
    public static void main(String[] args) {
        Hinh h=new Hinh();
        h.Ve();
        Hinh h1 = new HinhVuong();
        h1.Ve();
        Hinh h2 = new HinhTron();
        h2.Ve(); }
}
```

Kết quả xuất ra màn hình:

- Ve hinh
- Ve vuong
- Ve tron

Lớp trừu tượng (Abstract)

- Là lớp đặc biệt, các phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết.
- Dùng để định nghĩa các phương thức và thuộc tính chung cho các lớp con của nó.
- Dùng từ khóa **abstract** để khai báo một lớp trừu tượng
- Lớp **trừu tượng** không thể tạo ra đối tượng nhưng có thể khai báo biến thuộc lớp trừu tượng để tham chiếu đến các đối tượng thuộc lớp con của nó.
- Có thể khai báo 0,1 hoặc nhiều phương thức trừu tượng bên trong lớp.
- Một lớp trừu tượng thì bên trong nó có những method đặt là abstract (không có thân hàm), có method không đặt là abstract (có thân hàm).
- Các lớp con của lớp trừu tượng phải cài đặt các phương thức trừu tượng của lớp trừu tượng, nếu không nó trở thành trừu tượng

Khai báo lớp trừu tượng

- **Khai báo:**

  [public] abstract class *Tênlớp*{

  <các thuộc tính>

  <các phương thức trừu tượng>

  <các phương thức không trừu tượng>

  }

- **Tính chất:** mặc định là public, bắt buộc phải có từ khoá abstract để xác định đây là một lớp trừu tượng.
- **Lưu ý:** Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng

Khai báo phương thức lớp trừu tượng

- **Khai báo:**

[public] abstract <kiểu dữ liệu trả về> <tên phương thức>([<các tham số>]);

- Không khai báo tường minh mặc định là public.
- Tính chất của phương thức trừu tượng không được là **private** hay **static**
- Phương thức trừu tượng chỉ được khai báo dưới dạng khuôn mẫu nên không có phần dấu móc "{}" mà kết thúc bằng dấu chấm phẩy ";".

Ví dụ lớp trừu tượng

```
package tronvalapphuong;
 astract class Hinh
 {
        static final double PI=3.1415;
        public abstract double DienTich();
        public abstract double TheTich();
  }
```

Ví dụ lớp trừu tượng (tt)

```
package tronvalapphuong;
class HinhTron extends Hinh {
    private double R;
    public HinhTron(double r) {
        R=r;
    }
    @Override
    public double DienTich() {return PI*R*R; }
   @Override
    public double TheTich() {
        return 0;  }
}
```

Ví dụ lớp trừu tượng (tt)

```
package tronvalapphuong;
class HinhLapPhuong extends Hinh {
    private double a;private double b;private double c;
    public HinhLapPhuong(double aa, double bb, double cc) {
        a=aa;b=bb;c=cc;
    }
    @Override
    public double DienTich() {return(2*(a*b+b*c+a*c));   }
    @Override
    public double TheTich() {
        return a*b*c;
    }
    }
```

Ví dụ lớp trừu tượng (tt)

```
package tronvalapphuong;
public class TronVaLapPhuong {
    public static void main(String[] args) {
        Hinh  hr= new HinhTron(5.5);
        System.out.println("Hinh tron");
        System.out.println("Dien Tich: "+hr.DienTich());
        System.out.println("The Tich: "+hr.TheTich());
        Hinh hlp=new HinhLapPhuong(2,3,4);
        System.out.println("Hinh lap phuong: ");
        System.out.println("Dien Tich: "+hlp.DienTich());
        System.out.println("The Tich: "+hlp.TheTich());
    }
}
```

Kết quả

Hinh tron

Dien Tich: 95.03037499999999

The Tich: 0.0

Hinh lap phuong:

Dien Tich: 52.0

The Tich: 24.0

Giao tiếp (giao diện – Interfaces)

- **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.
  - 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 (final), static và các phương thức là trừu tượng (mặc dù không có từ khóa abstract).
  - Một lớp implements nhiều interface

Khai báo Giao tiếp

- **Khai báo:**

  [public] interface <tên giao tiếp> [extends <danh sách giao tiếp>]

- **Tính chất:**
  - Luôn luôn là public, không khai báo tường minh thì mặc định là public.
  - *Danh sách các giao tiếp:* danh sách các giao tiếp cha đã được định nghĩa để kế thừa, các giao tiếp cha được phân cách nhau bởi dấu phẩy.
  - *Lưu ý:* Một giao tiếp chỉ có thể **kế thừa** từ các **giao tiếp khác** mà **không thể** được **kế thừa từ các lớp sẵn có**.

Khai báo phương thức của Giao tiếp

- **Khai báo:**

  [public] <kiểu giá trị trả về> <tên phương thức> ([<các tham số>]) [throws <danh sách ngoại lệ>];

- **Tính chất:**
  - Thuộc tính hay phương thức luôn luôn là public, không khai báo tường minh thì mặc định là public.
  - Phương thức được khai báo dưới dạng mẫu, không có cài đặt chi tiết, không có phần dấu móc "{}" mà kết thúc bằng dấu chấm phẩy ";". Phần cài đặt chi tiết được thực hiện trong lớp sử dụng giao tiếp.
  - *Thuộc tính* luôn có tính chất là hằng (final), tĩnh (static) và public. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính.

```
public static final int AGE = 22; hoặc
int AGE = 22;
```

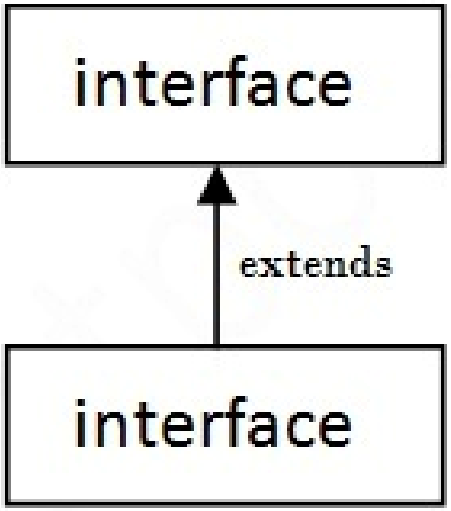
Sử dụng Giao tiếp

- **Khai báo:**

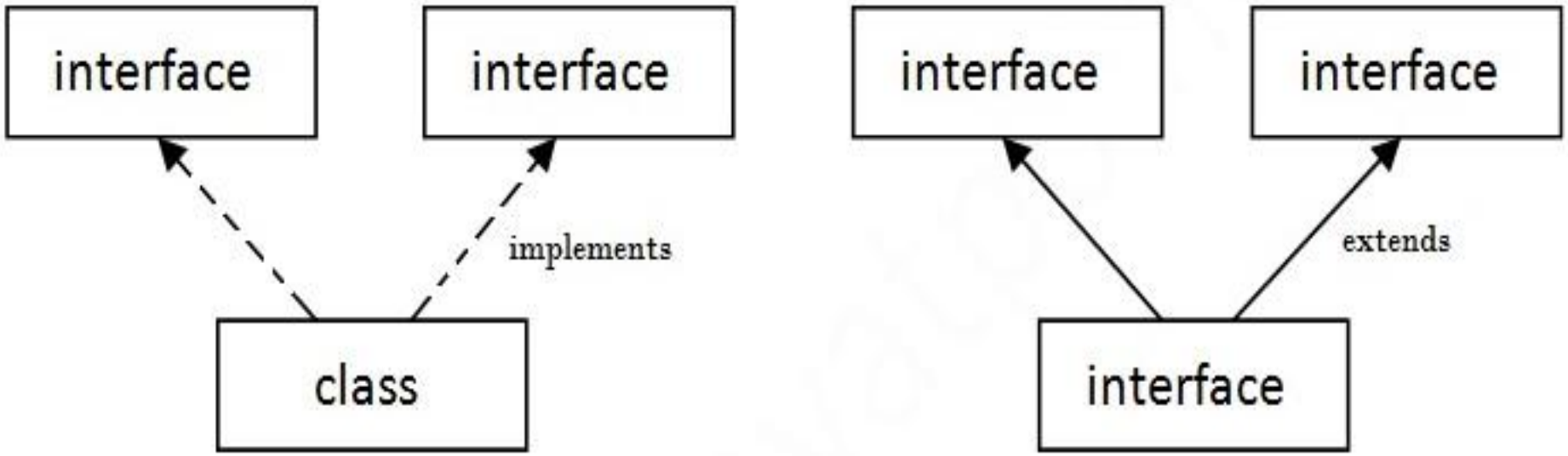
  <tính chất> class <tên lớp> implements <các giao tiếp>{…}

  - **Tính chất** và **tên lớp** được sử dụng như trong khai báo lớp thông thường.
  - **Các giao tiếp:** một lớp có thể cài đặt nhiều giao tiếp. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các giao tiếp mà nó sử dụng.
  - *Lưu ý:* Một phương thức được khai báo trong giao tiếp phải được cài đặt cụ thể trong lớp có cài đặt giao tiếp nhưng **không được** phép **khai báo chồng**.

Quan hệ giữa các lớp interface



Đa thừa kế trong Java



Ví dụ - interface

```
// Định nghĩa một interface "Hinh" trong tập tin
package vidu1;
public interface Hinh {
    final double PI=3.1415;
    public double DienTich();
    public double ChuVi();
    public String LayTenHinh();
    public void Nhap();
}
```

Ví dụ- interface

Định nghĩa lớp "HinhTron" implement từ lớp "Hinh"

```
package vidu1;
import java.util.Scanner;
class HinhTron implements Hinh{
    private double R;
    @Override
    public double DienTich() {
        return PI*R*R;
       }
    @Override
    public double ChuVi() {
        return 2*PI*R;
    }
```

```
@Override
    public String LayTenHinh() {
        return ("Hình tròn");
    }
    @Override
    public void Nhap(){
        System.out.print("Nhap R=");
        Scanner scan = new
Scanner(System.in);
        R=scan.nextDouble();
    }
}
```

Ví dụ- interface

Định nghĩa lớp "HinhVuong" implement từ lớp "Hinh"

```
package vidu1;
import java.util.Scanner;
class HinhVuong implements Hinh{
    private double canh;
    @Override
    public double DienTich() {
        return canh*canh;
    }
    @Override
    public double ChuVi() {
        return canh*4;
    }
    @Override
    public String LayTenHinh() {
        return ("Hình vuông");
    }
    @Override
    public void Nhap()
    {
        System.out.print("Nhap canh=");
        Scanner scan = new
Scanner(System.in);
        canh=scan.nextDouble();
    }
}
```

Ví dụ- interface

Sử dụng các lớp

```
package vidu1;
public class Vidu1 {
  public static void main(String[] args) {
        Hinh h=new HinhTron();
        h.Nhap();
        System.out.println("Dien tich hình tron:"+h.DienTich());
        System.out.println("Chu vi hình tron:"+h.ChuVi());
        h= new HinhVuong();
        h.Nhap();
        System.out.println("Dien tich hình vuong:"+h.DienTich());
        System.out.println("Chu vi hình :"+h.ChuVi());
    }
}
```

Kết quả

Nhap R=2

Dien tich hình tron:12.566

Chu vi hình tron:12.566

Nhap canh=4

Dien tich hình vuong:16.0

Chu vi hình :16.0

Abtract và interface

ABSTRACT	INTERFACE
Abstract Class là "khuôn mẫu" cho Class	Interface chứa các "khuôn mẫu" cho Method
Lớp trừu tượng có thể có phương thức static, phương thức main và constructor	Interface không thể có phương thức static, main hoặc constructor.
Lớp con phải định nghĩa lại (Override) các phương thức trừu tượng (Có khai báo với từ khóa Abstract)	Các lớp con của Interface sẽ phải định nghĩa lại toàn bộ các phương thức có trong Interface, (Mặc định các phương thức trong Interface đều là phương thức trừu tượng - Abstract method)
Một lớp con của lớp trừu tượng chỉ được thừa kế 1 lớp cha là lớp trừu tượng (Dùng từ khóa extends để kế thừa).	Một lớp con có thể thừa kế nhiều Interface cùng một lúc (Dùng từ khóa implement) (đa kế thừa)
Abstract class có thể kế thừa abstract class khác và implement nhiều interface	Interface chỉ có thể kế thừa nhiều interface.

Abtract và interface

ABSTRACT	INTERFACE
Phương thức và thuộc tính của Abstract Class có thể sử dụng những access modifier: public, protected, default, private. Và có thể có final, non-final, static, non-static	Phương thức và thuộc tính của Interface luôn mặc định access modifier là public. Với phương thức thì luôn là abstract và thuộc tính luôn là static final.

Giao tiếp - Interface (tt)

- Default methods
- Static methods

Default methods

- **Khai báo:**

default <kiểu giá trị trả về> <tên phương thức> ([<các tham số>]) {.....}

- **Tính chất:**
  - Phương thức *default* giúp mở rộng interface mà không phải lo ngại phá vỡ các class được implements từ nó
  - Phương thức default giúp tháo gỡ các class cơ sở (base class), có thể tạo phương thức default và trong class được implement có thể chọn phương thức để *override*
  - Phương thức default cũng có thể được gọi là phương thức Defender (Defender Methods) hay là phương thức Virtual mở rộng (Virtual extension methods)

Ví dụ Default methods


```
public interface Interface1 {
    void method1(String str);
    default void log(String str){
        System.out.println("I1 logging::"+str);
        print(str);
    }
}
```

Static methods

- **Khai báo:**

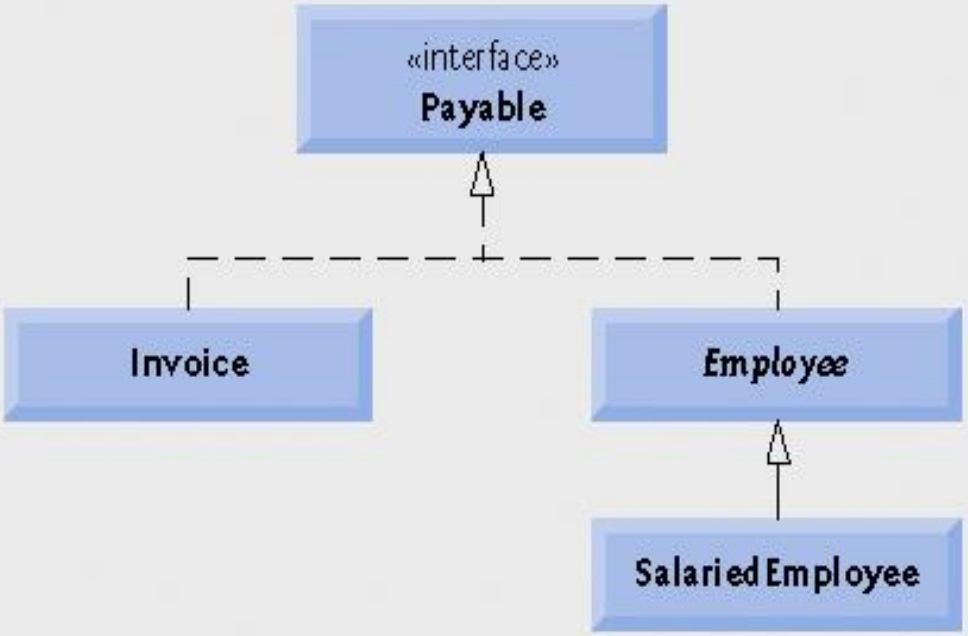
static <kiểu giá trị trả về> <tên phương thức> ([<các tham số>]) {.....}

- **Tính chất:**
  - Cũng giống phương thức *default* ngoại trừ việc nó **không thể được override** chúng trong class được implements.
  - Phương thức static chỉ hiển thị trong phương thức của interface, **nếu xóa** phương thức static trong **class được implements**, chúng ta sẽ không thể sử dụng nó cho đối tượng (object) của class được implements
  - Phương thức static rất hữu ích trong việc cung cấp các phương thức tiện ích, ví dụ như là kiểm tra null, sắp xếp tập hợp, v.v…
  - Phương thức static giúp chúng ta bảo mật, không cho phép class implements từ nó có thể override

Ví dụ Static methods


```
static boolean isNull(String str) {
        System.out.println("Interface Null Check");
        return str == null ? true : "".equals(str);
 }
```

Ví dụ



Ví dụ

```
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`

```
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`


```
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
```

```
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`


```
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
```

```
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`)

```
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

```
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

```
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
```

Quan hệ giữa Class và Interface

	Class	Interface
Class	extends	implements
Interface		extends

Q & A