



LẬP TRÌNH VIÊN CÔNG NGHỆ JAVA

Module 2

☞ Click vào phụ lục để chuyển tới bài cần đọc

Phụ lục

Bài 1 Lập trình hướng đối tượng nâng cao.....	2
Bài 2 Collections và Generic	20
Bài 3 Lớp lồng cấp – Inner Class	35
Bài 4: XML DOM	48
Bài 5 làm việc với CSDL – p1.....	65
Bài 6 làm việc với CSDL – p2.....	78
Bài 7 Bảo mật ứng dụng	92
Bài 8 Design Pattern – p1	102
Bài 9 Design Pattern – p2	117



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 1: *LTHDT Nâng cao*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



1. Tính kế thừa và đa hình
2. Abstract (Trùu tượng)
3. Interface (Giao diện)



Tính kế thừa và đa hình



❑ Khái niệm

- Kế thừa (Inheritance): Quá trình mà các tính chất và hành vi được truyền từ thực thể cha đến thực thể con
- Kế thừa giúp tránh phải tạo các tính chất và hành vi đã có sẵn và sử dụng lại cái đã có sẵn để tạo ra thực thể mới



Tính kế thừa và đa hình



❑ Tính năng và thuật ngữ

- Kế thừa cho phép định nghĩa một **lớp tổng quát** trước, và định nghĩa các **lớp chuyên biệt** sau bằng cách cài đặt thêm các chi tiết vào định nghĩa lớp
- Lớp mà kế thừa các phương thức và các thuộc tính gọi là **lớp con** (lớp dẫn xuất)
- Lớp mà cho phép lớp con kế thừa các phương thức và các thuộc tính gọi là **lớp cha** (lớp cơ sở)



Tính kế thừa và đa hình

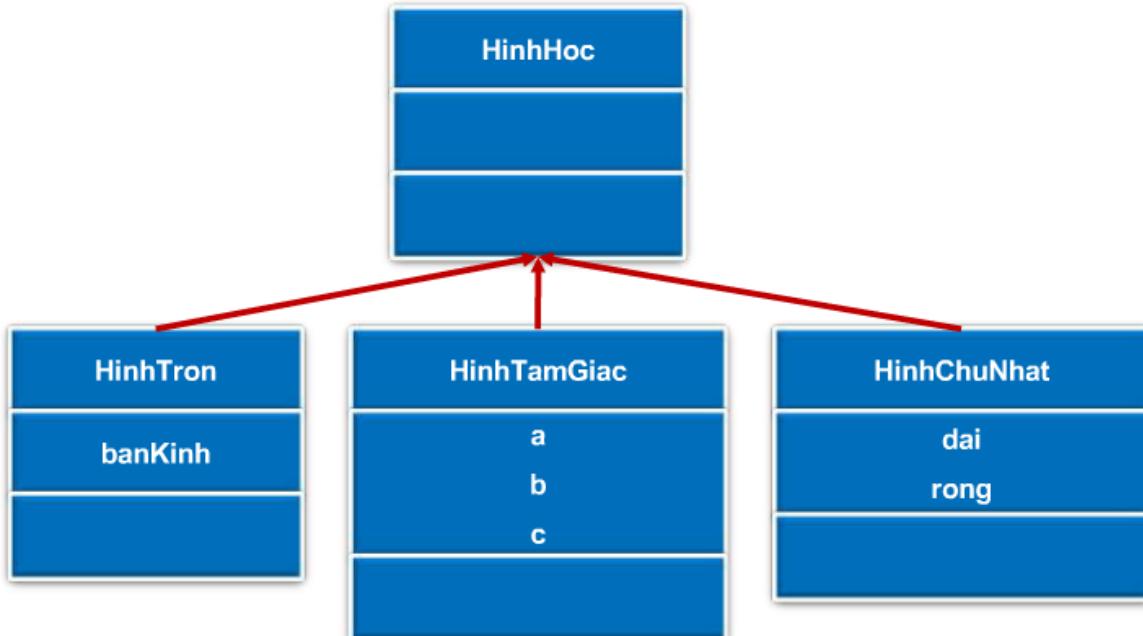


❑ Lớp con

- Sử dụng từ khóa **extends** để tạo ra lớp con
- Một lớp con chỉ có thể kế thừa trực tiếp từ một lớp cha
- Nếu lớp con không kế thừa từ lớp cha nào hết, mặc định xem như nó kế thừa từ lớp cha tên là **Object**
- Hàm dựng không được kế thừa.



Tính kế thừa và đa hình



Tính kế thừa và đa hình

Khái niệm kế thừa đa hình

```
class HinhHoc{
    public HinhHoc(){
        System.out.println("Hinh hoc khong tham so");
    }
    public HinhHoc(int giaTri){
        System.out.println("Hinh hoc co tham so");
    }
}
```



Tính kế thừa và đa hình



Khái niệm kế thừa đa hình

```
class HinhHoc{
    public HinhHoc(){
        System.out.println("Hinh hoc khong tham so");
    }
    public HinhHoc(int giaTri){
        System.out.println("Hinh hoc co tham so");
    }
}
```



Tính kế thừa và đa hình



Khái niệm kế thừa đa hình

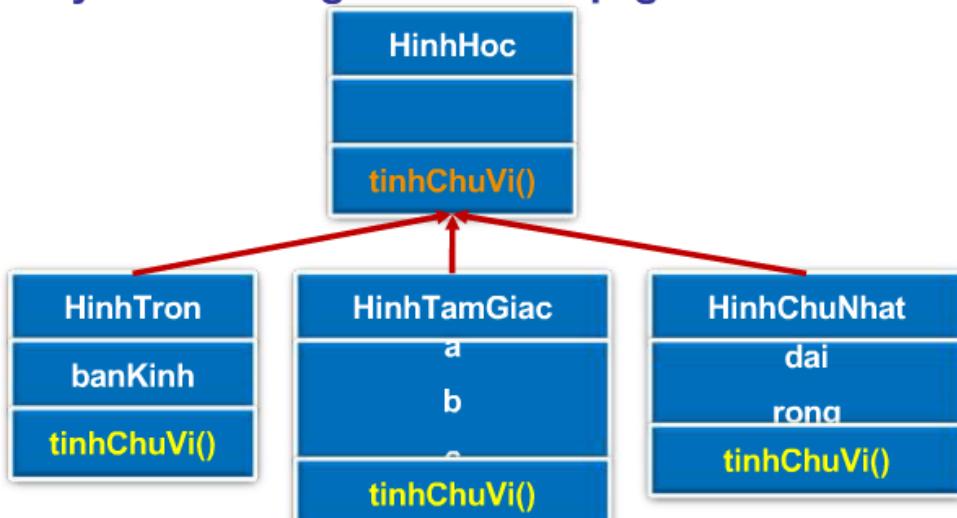
```
class HinhTron extends HinhHoc{
    double bankinh;
    public HinhTron(){
        System.out.println("Hinh tron");
    }
}
class HinhTamGiac extends HinhHoc{
    double a,b,c;
    public HinhTamGiac(int a){
        super(a); System.out.println("Hinh tam giac");
    }
}
```



Tính kế thừa và đa hình



- **Tính đa hình (polymorphism): một thực thể có nhiều hình thức, trạng thái khác nhau tùy thuộc vào yêu cầu và ngữ cảnh sử dụng**



Tính kế thừa và đa hình



- **Ghi đè hàm (overriding)**

- Lớp con định nghĩa một phương thức mới trùng tên và trùng tham số với một phương thức của lớp cha
- Phương thức ghi đè ở lớp con phải:
 - Cùng tên và tham số
 - Không được sử dụng chỉ định từ truy xuất yếu hơn
- Từ khóa **super**
 - Truy xuất đến phương thức cùng tên của lớp cha
 - Truy xuất đến hàm dựng của lớp cha



Tính kế thừa và đa hình



❑ Nạp chồng hàm

- Là khả năng một lớp có nhiều phương thức có cùng tên
 - Khác nhau về số tham số và có cùng kiểu dữ liệu của tham số
 - Có cùng số tham số và khác nhau kiểu dữ liệu của tham số
 - Hàm dựng cũng có khả năng nạp chồng với quy tắc như trên



Tính kế thừa và đa hình



❑ Nạp chồng hàm

- Ví dụ nạp chồng hàm

```
class HinhChuNhat extends HinhHoc{
    double dai, rong;
    public HinhChuNhat() {}
    public HinhChuNhat(double d,double r) {
        dai=d; rong=r;
    }
    public HinhChuNhat(double a) {
        dai=rong=a; }
}
```



Tính kế thừa và đa hình



❑super

- Được dùng trong **hàm dựng** hoặc trong **hàm thực thể** để chỉ đến đối tượng cha của đối tượng hiện hành
- Được dùng như một **thể hiện của lớp cha** trong khai báo của lớp con



Tính kế thừa và đa hình



❑super

- Ví dụ:

```
class HinhChuNhat extends HinhHoc{
    double dai,rong;
    public HinhChuNhat(){}
    public HinhChuNhat(double dai,double rong){
        super();
        this.dai = dai;this.rong = rong;
    }
    public HinhChuNhat(double a) { dai = rong = a; }
}
```



Tính kế thừa và đa hình



this

- Được dùng trong **hàm dựng** hoặc trong **hàm thực thể** để **chỉ đến đối tượng hiện hành**
- Được dùng để truy xuất đến các biến thành viên mà bị các biến cục bộ che khuất vì khai báo trùng tên



Tính kế thừa và đa hình



this

- Ví dụ

```
class HinhChuNhat extends HinhHoc{
    double dai,rong;
    public HinhChuNhat(){}
    public HinhChuNhat(double dai,double rong){
        this.dai=dai;this.rong=rong;
    }
    public HinhChuNhat(double a) { dai=rong=a; }
}
```



Tính kế thừa và đa hình



❑ static

- Biến thuộc lớp

- Được khai báo sử dụng từ khóa **static**
- Khi giá trị của biến thay đổi, tất cả các đối tượng của lớp sẽ được cập nhật để nhìn thấy giá trị mới của biến
- *Có thể được quản lý mà không cần phải tạo ra thực thể của lớp*
- *Được truy xuất tới thông qua tên lớp*



Tính kế thừa và đa hình



❑ static

- Ví dụ

```

class MyClass{
    public static int
    giatri=10;
    public void Tang(int
    x){
        giatri+=x;
    }
    public void Xuat(){
        System.out.println(gia
        tri);
    }
}

public static void
main(String args[]){
    MyClass a=new
    MyClass();
    MyClass b=new Class();
    System.out.println(
    MyClass.giatri);
    a.Tang(10); b.Xuat();
}

```



Tính kế thừa và đa hình



❑Hàm static

- Cũng được xem là phương thức của lớp, nhưng không tham chiếu tới bất kỳ thực thể nào của lớp, thường dùng để truy xuất tới các biến và hàm thành viên
- Từ khóa **this** không được dùng bên trong hàm static
- Có thể được truy xuất bằng cách sử dụng tên lớp và tên thực thể



Tính kế thừa và đa hình



❑Hàm static

```

class MyClass{
    public static int giatri
    = 10;
    public void Tang(int x){
        giatri += x;
    }
    public void Xuat(){
        System.out.println(giatri
    }
}

public static void
main(String args[]){
    MyClass a = new
    MyClass();
    MyClass b = new
    Class();
    System.out.println(
    MyClass.giatri);
    a.Tang(10);
    b.Xuat();
}

```



Tính kế thừa và đa hình



❑ Khối lệnh static

- Là khối lệnh nằm trong cặp dấu ngoặc nhọn
- Bắt đầu bằng từ khóa static
- *Dùng để khởi tạo các biến static của lớp*
- Khối lệnh khởi tạo static chỉ dùng để tham chiếu đến các biến trong lớp mà đã được khai báo trước đó



Tính kế thừa và đa hình



❑ static

```
class MyClass{
    static int counter = 0;
    static {
        counter++;
        System.out.println("Khoi
lenh static: counter=" +
counter);
    }
    MyClass(){System.out.println("Ham
khai tao: counter=" +
counter);}
}

public static void
main(String args[]){
    MyClass m = new
MyClass();
    System.out.println("Phuong
thuc main: counter =
"+MyClass.counter);
    //System.out.println
("Phuong thuc main: counter
= "+m.counter);
}
```



Nội dung



1. Tính kế thừa và đa hình
2. Abstract (Trừu tượng)
3. Interface (Giao diện)



Abstract (Trùu tượng)



- ❑ **Tính trừu tượng được sử dụng trong các tình huống đối tượng có tồn tại nhưng thực sự nó thuộc một đối tượng con cụ thể nào đó**
- ❑ **Đặc tả abstract áp dụng cho:**
 - lớp gọi là lớp trừu tượng (abstract class)
 - hàm gọi là hàm trừu tượng (abstract method)

abstract class <Tên Lớp> {}

abstract <Kiểu dữ liệu> <Tên hàm>(<Đối số>);

Chỉ là khai báo, không hiện thực



Abstract (Trừu tượng)



- ❑ Lớp trừu tượng không khởi tạo thực thể vì chưa hiện thực đầy đủ
- ❑ Lớp trừu tượng chứa một hoặc nhiều hàm trừu tượng. Tuy nhiên, **không** có hàm trừu tượng vẫn khai báo lớp trừu tượng
- ❑ Một lớp chứa hàm trừu tượng thì lớp đó phải là lớp trừu tượng
- ❑ Một lớp thừa kế lớp trừu tượng phải hiện thực tất cả các hàm trừu tượng hoặc lớp đó cũng là lớp trừu tượng



Abstract (Trùu tượng)



- ❑ Lớp trừu tượng hiện thực (**implements**) một giao diện (**interface**) và có thể không hiện thực một hàm

```
abstract class Shape {
    abstract void draw();
    void message() {
        System.out.println("I cannot live without being a
parent.");
    }
}
class Circle extends Shape {
    void draw() {
        System.out.println("Circle drawn.");
    }
}
```



Abstract (Trùu tượng)

```
class Cone extends Shape {  
    void draw() {  
        System.out.println("Cone drawn.");  
    }  
}  
  
public class RunShape {  
    public static void main(String[] args) {  
        Circle circ = new Circle();  
        Cone cone = new Cone();  
        circ.draw();  
        cone.draw();  
        cone.message();  
    }  
}
```



Nội dung

1. Tính kế thừa và đa hình
2. Abstract (Trùu tượng)
3. Interface (Giao diện)



Interface (Giao diện)



- ❑ **Thể hiện tính bao đóng (encapsulation) trong lập trình hướng đối tượng**
- ❑ **Hỗ trợ đa thừa kế: một lớp có thể thừa kế một lớp và hiện thực nhiều giao diện**
`<subclass> extends <superclass>
implements <interface1>, <interface2>{}`
- ❑ **Lớp hiện thực giao diện phải hiện thực tất cả các hàm được đặc tả trong giao diện đó**



Interface (Giao diện)



- ❑ **Cấu trúc định nghĩa:**

```
interface <InterfaceName> {
    <dataType1> <var1>;
    <dataType2> <var2>;
    <ReturnType1> <methodName1> ( );
    <ReturnType2> <methodName2>(<parameters>);
}
```
- ❑ **Hàm khai báo có đặc tả俨 public và abstract**
- ❑ **Biến khai báo có đặc tả俨 là public, static và final**



Interface (Giao diện)

```
interface ParentOne {
    int pOne = 1;
    void printParentOne();
}

interface ParentTwo {
    int pTwo = 2;
    void printParentTwo();
}

interface Child extends ParentOne,
ParentTwo{
    int child = 3;
    void printChild();
}
```



Interface (Giao diện)

```
class InheritClass implements
Child {
    public void printParentOne(){
        System.out.println(pOne);
    }
    public void printParentTwo(){
        System.out.println(pTwo);
    }
    public void printChild(){
        System.out.println(child);
    }
}

class TestInterface {
    public static void
main(String[] args){
    InheritClass ic
= new InheritClass();

    ic.printParentOne();
    ic.printParentTwo();
    ic.printChild();
}
}
```







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 2: **Collections và Generic**

Ngành LT & CSDL

www.t3h.vn



2014

SOI4



Nội dung



1. Giới thiệu
2. Collections
3. Generic



Giới thiệu



- ❑ Một ứng dụng thực tế thường liên quan đến một nhóm những phần tử dữ liệu. Nhóm phần tử này trong java được gọi là **kiểu dữ liệu tập hợp (Collections)**
- ❑ **Kiểu dữ liệu tập hợp** trong java được hỗ trợ bởi hai interface **Collection** và **Map**
- ❑ Java cũng hỗ trợ những lớp hiện thực những interface trên gồm cấu trúc dữ liệu và những thao tác trên những cấu trúc dữ liệu đó



Giới thiệu



❑ Khi đưa một đối tượng ngoài vào bên trong một kiểu dữ liệu tập hợp chưa biết kiểu dữ liệu cụ thể trong giai đoạn compile.

- Đây là cách chuyển đổi kiểu không an toàn và có thể xảy ra lỗi trong giai đoạn runtime



Giới thiệu



❑ Lập trình Generic cung cấp cách giao tiếp lên kiểu dữ liệu các phần tử của tập hợp

- Chuyển đổi kiểu an toàn
- Compiler kiểm tra các thao tác dữ liệu



Nội dung



1. Giới thiệu
2. Collections
3. Generic



Collections (Kiểu tập hợp)



- ❑ Một tập hợp là một đối tượng dữ liệu gồm nhiều phần tử con có cùng kiểu dữ liệu hoặc khác kiểu dữ liệu.
- ❑ Java xây dựng một kiến trúc thống nhất gọi là khung tập hợp (Collections Framework) tạo ra một kiểu tập hợp như là một hiện thực của kiểu dữ liệu trừu tượng ADT (abstract data type)



Kiểu tập hợp



❑ Khung tập hợp trong Java được hiện thực gồm 3 thành phần:

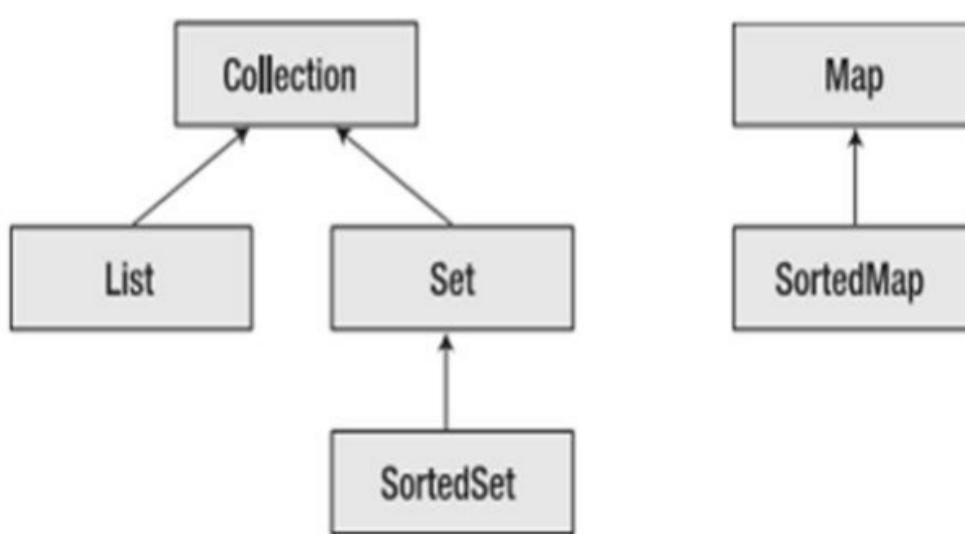
- Giao diện: định nghĩa ra một tập các thao tác
- Hiện thực: hiện thực những thao tác được định bến trong các interface. Một interface tương ứng có nhiều lớp implement khác nhau
- Giải thuật: đưa ra các giải thuật hiện thực tương ứng với từng cấu trúc tập hợp khác nhau như giải thuật tìm kiếm, giải thuật sắp xếp,...



Kiểu tập hợp



❑ Tập giao diện



Kiểu tập hợp



- ❑ Giao diện List và Set thừa kế từ giao diện Collection
- ❑ Không có lớp hiện thực nào trực tiếp của giao diện Collection
- ❑ List: là một tập hợp có thứ tự của các phần tử
 - Có thể chứa các phần tử trùng lặp (duplicate) về dữ liệu
 - Được hiện thực bởi 3 lớp ArrayList, LinkedList và Vector



Kiểu tập hợp



- ❑ Set: là tập hợp các phần tử duy nhất (unique)
 - Không có phần tử trùng lặp
 - Được hiện thực bởi lớp HashSet và LinkHashSet





Kiểu tập hợp

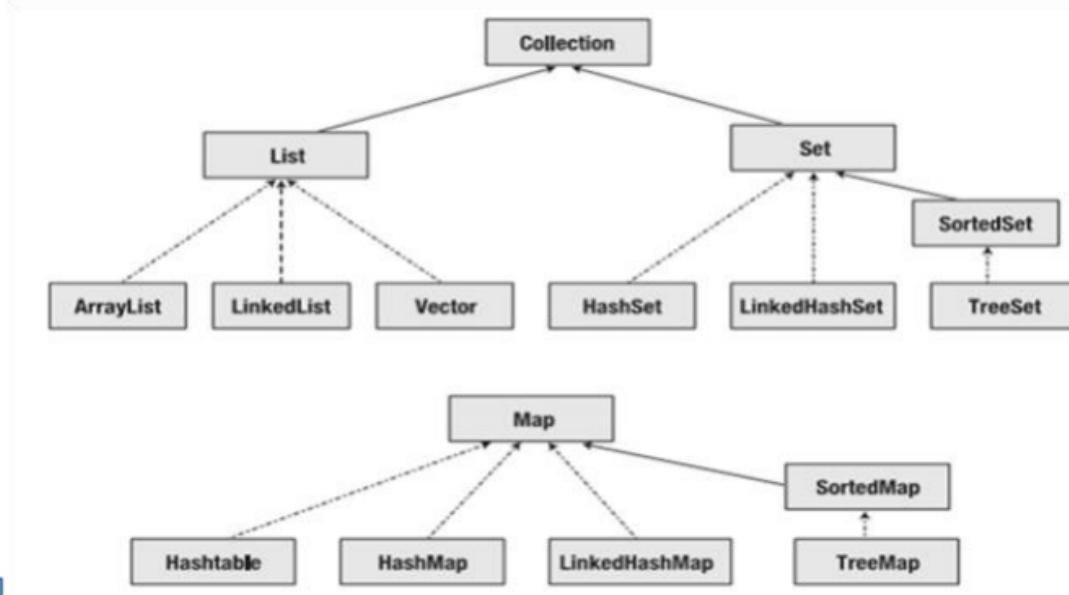
❑ **Map:** là một tập hợp gồm các phần tử ánh xạ khóa tới giá trị (key-value)

- Mỗi key được ánh xạ đến duy nhất một value
- Key là duy nhất
- Hiện thực bởi lớp HashMap và HashTable



Kiểu tập hợp

❑ **Sơ đồ gồm giao diện và hiện thực**



Kiểu tập hợp



❑Những điểm chính hiện thực

- Performance: Thời gian của những tác vụ trên dữ liệu như tìm kiếm (searching), sắp xếp (sorting)
- Sorted/Ordered: Các phần tử được sắp xếp theo thứ tự nào đó
- Tính duy nhất của các phần tử trong một tập hợp
- Synchronized (đồng bộ): hỗ trợ cơ chế lập trình đồng bộ dữ liệu



Kiểu tập hợp



❑Hiện thực giao diện List

- ArrayList
 - Số lượng phần tử động (có thể tăng hoặc giảm)
 - Thời gian truy cập các phần tử là hằng số (constant-time access)



Kiểu tập hợp



❑ Hiện thực giao diện List

- LinkedList

- Thời gian thêm mới và xóa một phần tử là hằng số
- Truy cập ngẫu nhiên (random access) là thời gian tuyến tính

- Vector: Giống tương tự ArrayList nhưng cung cấp cơ chế đồng bộ cho ứng dụng đa luồng



Kiểu tập hợp



❑ Hiện thực giao diện Set

- HashSet

- Cung cấp khả năng truy cập nhanh hơn TreeSet
- Không đảm bảo các phần tử sẽ được sắp xếp

- TreeSet

- Các phần tử được sắp xếp, nhưng thời gian truy cập không tốt bằng HashSet



Kiểu tập hợp



▫ Hiện thực giao diện Set

- HashSet

- Giống HashSet nhưng có danh sách liên kết kép

- Tất các lớp hiện thực đều không hỗ trợ đồng bộ



Kiểu tập hợp



▫ Hiện thực giao diện Map

- HashTable

- Dựa cơ bản theo cấu trúc dữ liệu hashtable

- Có thể dùng bất kì đối tượng khác Null làm key hoặc value

- Một đối tượng được dùng như key thì phải hiện thực phương thức hashCode() và equals() để lưu trữ và truy xuất đối tượng trong bộ nhớ



Kiểu tập hợp



▫ Hiện thực giao diện Map

- HashMap

- Dựa cơ bản theo cấu trúc dữ liệu hashtable
- Giống HashTable nhưng cho phép Null và không hỗ trợ đồng bộ



Kiểu tập hợp



▫ Hiện thực giao diện Map

- LinkedHashMap

- Định nghĩa thứ tự lặp dùng để sắp xếp key được thêm mới vào trong map

- TreeMap

- Hiện thực giao diện SortedMap
- Đảm bảo có thứ tự key (tăng)
- Không hỗ trợ cơ chế đồng bộ



Kiểu tập hợp



❑ Hiệu suất tìm kiếm trên kiểu tập hợp

- Có thể tìm kiếm một phần tử trong tập hợp hoặc mảng dùng phương thức `binarySearch()` trong lớp `Arrays`
 - Hàm nhận vào đối số là tập hợp được sắp xếp và trả về chỉ số của phần tử, ngược lại là -1



Nội dung



1. Giới thiệu
2. Collections
3. Generic



Generics



□ Giới thiệu

- Viết một ứng dụng in ra một dãy số nguyên, số thực, chuỗi,...
- Viết một ứng dụng tìm giá trị lớn nhất của 3 đối số có kiểu số nguyên, kiểu số thực, kiểu chuỗi,...



Generics



- Cho phép người dùng đặc tả kiểu dữ liệu của từng phần tử trong tập hợp
- Hỗ trợ kiểm tra kiểu dữ liệu của phần tử giai đoạn compile
- Tái sử dụng cho nhiều kiểu dữ liệu
- Cho phép trừu tượng hóa kiểu dữ liệu



Lập trình Generics



```
// generic method printArray
public static < E > void printArray( E[] inputArray )
{
    // display array elements
    for ( E element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray
```



Lập trình Generics



□Chú ý:

```
List<String> ls = new ArrayList<String>();
List<Object> lo = ls;
```







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 3: *Lớp lồng cấp – Inner Class*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



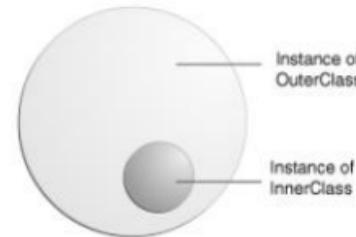
1. Giới thiệu
2. Các loại lớp lồng cấp – Inner class



Giới thiệu



- ❑ Lớp lồng cấp là lớp được định nghĩa bên trong một lớp khác.
- ❑ Thành phần bên trong của lớp gồm biến và phương thức. Tuy nhiên, với việc hỗ trợ lớp lồng cấp thì lớp cũng là thành phần bên trong một lớp



Giới thiệu



- Lớp lồng cấp có tầm vực cũng giống như biến hoặc phương thức bên trong một lớp**
- Ứng dụng thực tế:** Trong một số nghiệp vụ xử lý đặc biệt thì ứng dụng hiện thực lớp lồng cấp.
- Lớp này là lớp đặc biệt chỉ được sử dụng trong trường hợp riêng và gắn liền một lớp nào đó**
- Ví dụ những ứng dụng xử lý sự kiện như sự kiện về chuột, bàn phím (Lập trình Event-handling)**



Giới thiệu



- Một trong những lợi ích chính của lớp lồng cấp là mối quan hệ đặc biệt (special relationship) giữa thực thể của lớp lồng cấp chia sẻ với thực thể lớp ngoài (outer class)**
- Mối quan hệ đặc biệt cho phép lớp lồng cấp truy cập đến những thành phần bên trong (biến và phương thức) của lớp ngoài**
 - Một lớp lồng có thể truy cập tất cả các thành phần lớp ngoài có đặc tả private



Nội dung



1. Giới thiệu
2. Các loại lớp lồng cấp – Inner class



Các loại lớp lồng cấp – Inner class



❑ Lớp lồng cấp chính quy (Regular inner class)

- Thành phần bên trong lớp lồng cấp chính qui không cho phép khai báo static
- Để truy cập lớp lồng cấp ở bên ngoài thì phải thông qua thực thể của lớp ngoài
- Truy cập bên trong lớp ngoài thì thông qua đối tượng của lớp lồng cấp giống như lớp thông thường



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

- Khai báo

```
class MyOuter {
    class MyInner { }
}
```

- Khi biên dịch thì sẽ tạo ra hai file lưu trữ trong ổ cứng có tên là MyOuter.class và MyOuter\$MyInner.class



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

- Ví dụ:

```
class MyOuter {
    private int x = 7;
    // định nghĩa lớp lồng cấp
    class MyInner {
        public void seeOuter() {
            System.out.println("Outer x is " + x);
        }
    } // đóng định nghĩa lớp lồng cấp
} // đóng định nghĩa lớp ngoài
```



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

- Lưu ý: Thực thể của lớp lồng chỉ được tạo khi tồn tại thực thể của lớp ngoài
- Khởi tạo đối tượng lớp lồng cấp bên trong lớp ngoài



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

```
class MyOuter {  
    private int x = 7;  
    public void makeInner() {  
        MyInner in = new MyInner();  
        in.seeOuter();  
    }  
    class MyInner {  
        public void seeOuter() {  
            System.out.println("Outer x is " + x);  
        }  
    }  
}
```



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

- Tạo đối tượng lớp lồng cấp bên ngoài lớp ngoài

```
public static void main (String[] args) {
    MyOuter mo = new MyOuter();
    MyOuter.MyInner inner = mo.new MyInner();
    inner.seeOuter();
}
```



Các loại lớp lồng cấp – Inner class



```
class MyOuter {
    private int x = 7;
    public void makeInner() {
        MyInner in = new MyInner();
        in.seeOuter();
    }
    class MyInner {
        public void seeOuter() {
            System.out.println("Outer x is " + x);
            System.out.println("class ref is " + this);
            System.out.println("class ref is " + MyOuter.this);
        }
    }
    public static void main (String[] args) {
        MyOuter.MyInner inner = new MyOuter().new MyInner();
        inner.seeOuter();
    }
}
```



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp chính quy

- là một thành phần của lớp nên đóng vai trò như là biến thực thể và phương thức
- Vì thế, những đặc tả sau có thể ứng dụng:
 - Đặc tả truy cập: public, private, protected
 - final
 - abstract
 - static: trường hợp này lớp lồng cấp gọi lớp lồng cấp tĩnh



Các loại lớp lồng cấp – Inner class



□ Lớp lồng cấp cục bộ hàm (Method-Local inner class)

- Là lớp lồng cấp được định nghĩa bên trong phương thức
- Lớp lồng cấp cục bộ hàm chỉ có tầm vực sử dụng bên trong hàm chứa định nghĩa nó
- Lớp lồng cấp cục bộ hàm chia sẻ mối quan hệ đặc biệt với lớp ngoài và có thể truy cập các thành phần lớp ngoài với đặc tả private



Các loại lớp lồng cấp – Inner class



❑ Lớp lồng cấp cục bộ hàm

- Lưu ý: lớp lồng cấp cục bộ hàm không truy cập được biến cục bộ trong hàm
 - Trừ khi, biến cục bộ khai báo đặc tả final
- Chỉ có 2 đặc tả cho lớp lồng cấp cục bộ hàm là:
 - abstract
 - final



Các loại lớp lồng cấp – Inner class



❑ Lớp lồng cấp cục bộ hàm

```
class MyOuter2 {
    private String x = "Outer2";
    void doStuff() {
        String z = "local variable";
        class MyInner {
            public void seeOuter() {
                System.out.println("Outer x is " + x);
                System.out.println("Local variable z is " + z); ←
            } // close inner class method
        } // close inner class definition
    } // close outer class method doStuff()
} // close outer class
```



Các loại lớp lồng cấp – Inner class



❑ Lớp lồng cấp vô danh (Anonymous inner class)

- Là lớp lồng cấp được định nghĩa nhưng không có tên
- Có thể định nghĩa lớp lồng cấp vô danh bên trong đối số của một hàm
- Định nghĩa lớp lồng cấp vô danh thừa kế một lớp cha hoặc hiện thực giao diện nào đó



Các loại lớp lồng cấp – Inner class



❑ Lớp lồng cấp vô danh

```
class Popcorn {
    public void pop() {
        System.out.println("popcorn");
    }
}
class Food {
    Popcorn p = new Popcorn() {
        public void pop() {
            System.out.println("anonymous
popcorn");
        }
    };
}
```



Các loại lớp lồng cấp – Inner class

```

class Popcorn {
    public void pop() {
        System.out.println("popcorn");
    }
}
class Food {
    Popcorn p = new Popcorn() {
        public void sizzle() {
            System.out.println("anonymous sizzling popcorn");
        }
        public void pop() {
            System.out.println("anonymous popcorn");
        }
    };
    public void popIt() {
        p.pop();
        p.sizzle();
    }
}

```



Các loại lớp lồng cấp – Inner class

❑ Lớp lồng cấp vô danh

```

interface Cookable {
    public void cook();
}
class Food {
    Cookable c = new Cookable() {
        public void cook() {
            System.out.println("anonymous cookable
implementer");
        }
    };
}

```

Lớp lồng cấp vô danh hiện thực giao diện



Các loại lớp lồng cấp – Inner class



☐ Lớp lồng cấp vô danh

```
class MyWonderfulClass {
    void go() {
        Bar b = new Bar();
        b.doStuff(new Foo() {
            public void foof() {
                System.out.println("foofy");
            } // end foof method
        }); // end inner class def, arg, and end statement
    } // end go()
} // end class
interface Foo {
    void foof();
}
class Bar {
    void doStuff(Foo f) { }
}
```

Lớp lồng cấp vô danh được định nghĩa trong đối số hàm



Lớp lồng cấp tĩnh



☐ Lớp lồng cấp tĩnh (Static nested classes/ top-level nested classes)

- Lớp lồng cấp chính qui được đặc tả static

```
class BigOuter {
    static class Nested { }
}
class Broom {
    public static void main (String [] args) {
        BigOuter.Nested n = new BigOuter.Nested();
    }
}
```







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 4: XML DOM

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung

1. Giới thiệu XML
2. Mô hình DOM
3. Truy xuất tài liệu XML
4. Ghi tài liệu XML



Giới thiệu



Giới thiệu



❑ XML (eXtensible Markup Language) – Ngôn ngữ đánh dấu mở rộng

- Được thiết kế để chuyển và lưu trữ dữ liệu
- Do W3C đề nghị sử dụng
- Mục đích chính của XML là đơn giản hóa việc chia sẻ dữ liệu giữa các hệ thống khác nhau, đặc biệt là các hệ thống được kết nối với Internet



Giới thiệu



❑ Đặc điểm

- Dễ dàng viết được những chương trình xử lý tài liệu XML
- Dễ đọc và có tính hợp lý cao
- Giảm thiểu những thuộc tính tùy chọn
- Dễ dàng được sử dụng trên Internet
- Hỗ trợ nhiều ứng dụng



Giới thiệu



□ Cấu trúc

- Tài liệu XML có cấu trúc cây (tree) bắt đầu bằng một nút gốc (root) và các nhánh lá (leave)
- Tài liệu XML là tài liệu tự mô tả và có cấu trúc đơn giản



Giới thiệu



□ Cấu trúc

```

Khai báo phiên bản và loại mã hóa
<?xml version="1.0" encoding="UTF-8"?>
<note> _____ Nút gốc (root
    <to>Students</to> _____ element)
    <from>Teacher</from> _____ Nút con (child
        <heading>Reminder</heading> _____ element)
        <body>Please come to class this Monday evening
            on time</body>
    </note> _____ Kết thúc nút
            _____ gốc

```



Giới thiệu



❑ Cú pháp

- Có một root Element duy nhất
- Mỗi tag được mở ra cần phải đóng lại
- Tag có phân biệt chữ hoa và chữ thường
- Nút con (Child Element) phải nằm trong nút cha (Parent Element)



Giới thiệu



❑ Cú pháp

- Ghi chú trong XML: <!-- Nội dung ghi chú-->
- Lưu trữ dấu xuống dòng: LF
- Chú ý: Trong file .xml cần phải có dòng khai báo XML ở đầu file



Giới thiệu



❑ Ví dụ

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Don_hang>
    <Ngay_dat_hang>2010-01-27</Ngay_dat_hang>
    <Ten_khach_hang>Khuất Thuỳ Phương</Ten_khach_hang>
    <Noi_dung>
        <Ma_san_pham>1</Ma_san_pham>
        <So_luong>2</So_luong>
    </Noi_dung>
    <Noi_dung>
        <Ma_san_pham>4</Ma_san_pham>
        <So_luong>1</So_luong>
    </Noi_dung>
</Don_hang>
```



Nội dung



1. Giới thiệu XML
2. Mô hình DOM
3. Truy xuất tài liệu XML
4. Ghi tài liệu XML

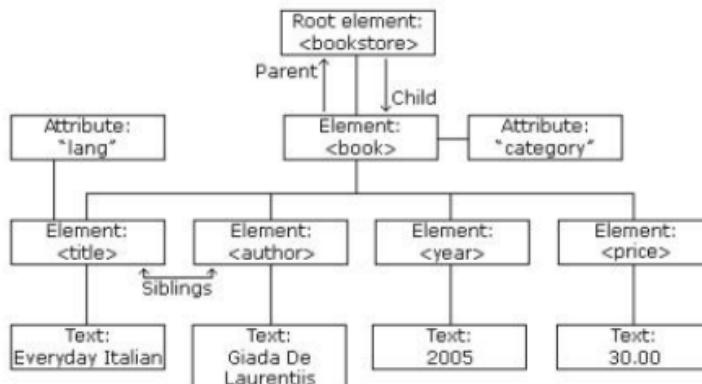


Mô hình DOM



□ Khái niệm

- Mô hình đối tượng cho phép xử lý trên tài liệu XML từ các ngôn ngữ lập trình
- Cấu trúc dữ liệu động biểu diễn thông tin của văn bản có cấu trúc nói chung và tài liệu XML nói riêng



Mô hình DOM



□ Đặc điểm

- XML DOM xác định các đối tượng và thuộc tính của tất cả các phần tử XML cũng như các phương thức để truy cập chúng (cách lấy, thay đổi, thêm hoặc xóa các phần tử XML...)
- Đọc ghi dễ dàng
- Khả năng truy vấn cao với ngôn ngữ Xpath
- Theo DOM, mọi thứ trong tài liệu XML đều là node
 - Toàn bộ tài liệu là một node tài liệu
 - Mỗi phần tử XML là một node phần tử
 - Văn bản trong phần tử XML là node văn bản
 - Mỗi thuộc tính là một node thuộc tính
 - Ghi chú là node ghi chú



Nội dung

1. Giới thiệu XML
2. Mô hình DOM
3. Truy xuất tài liệu XML
4. Ghi tài liệu XML



Truy xuất tài liệu XML

❑ Các bước thực hiện

- Bước 1: Import các gói thư viện XML có liên quan trước khi sử dụng

```
import org.w3c.dom.*;  
import javax.xml.parsers.*;  
import java.io.*;
```



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 2: Tạo đối tượng Document Builder

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
  
DocumentBuilder builder =  
    factory.newDocumentBuilder();
```



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 3: Tạo đối tượng Document từ một file hoặc stream

```
Document document =  
    builder.parse(new File(file));
```



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 4: Lấy phần tử gốc (root element)

```
Element root =  
document.getDocumentElement();
```



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 5: Kiểm tra các thuộc tính (attributes)

- `getAttribute("attributeName")`: trả về một thuộc tính cụ thể
- `getAttributes()`: trả về một Map (table) các tên/giá trị (names/values)



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 6: Kiểm tra các phần tử con (sub-element)

- `getElementsByTagName ("subelementName")`
trả về danh sách các sub-elements
- `getChildNodes ()`: trả về danh sách các node con



Truy xuất tài liệu XML



❑ Các bước thực hiện

- Bước 6:

- Cả hai phương thức đều trả về cấu trúc dữ liệu chứa các Node entry, chứ không phải là Element.
 - Node là parent interface của Element
 - Kết quả của phương thức `getElementsByTagName` có thể là định kiểu cho Element
 - Kết quả của `getChildNodes` là các Node entry của nhiều kiểu khác nhau



Truy xuất tài liệu XML



❑ Kiểm tra phần tử con với phương thức getElementsByTagName

- Cung cấp tag name như đối số
 - Tag name có phân biệt chữ hoa, chữ thường


```
someElement.getElementsByTagName("blah") khác
          someElement.getElementsByTagName("Blah")
```
 - "*" được cho phép như là đối số của tất cả các tag con
 - Phương thức này trả về NodeList



Truy xuất tài liệu XML



❑ Kiểm tra phần tử con với phương thức getElementsByTagName

- NodeList có hai phương thức
 - getLength: số lượng các phần tử con
 - item(index): phần tử có index được truyền vào
 - Item trả về Node (parent interface của Element), có thể định kiểu cho giá trị trả về



Truy xuất tài liệu XML



❑ Lấy Body Content

- Đơn giản khi lấy giá trị của thuộc tính

```
<foo someAttribute="some data"/>
```

- Khó hơn khi lấy giá trị của tag body

```
<foo>some data</foo>
```

- Lấy tag body

- Đơn giản hóa cây để kết hợp các text node

```
rootElement.normalize()
```

- Lấy text từ node

```
String body =
```

```
someElement.getFirstChild().getNodeValue();
```



Nội dung



1. Giới thiệu XML
2. Mô hình DOM
3. Truy xuất tài liệu XML
4. Ghi tài liệu XML



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 1: Import các gói thư viện XML có liên quan trước khi sử dụng

```
import org.w3c.dom.*;  
  
import javax.xml.parsers.*;  
  
import java.io.*;  
  
import javax.xml.transform.*
```



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 2: Tạo đối tượng Document Builder

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
  
DocumentBuilder builder =  
    factory.newDocumentBuilder();
```



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 3: Tạo đối tượng Document từ một file hoặc stream

```
Document doc;
File file = new File(fileName);
if (file.isFile()) {
    doc = dBuilder.parse(new
        FileInputStream(file));
} else {
    doc = dBuilder.newDocument();
}
```



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 4: Lấy phần tử gốc, nếu không có phần tử gốc thì tạo ra phần tử gốc

```
Element root =
document.getDocumentElement();
if (root == null) {
    root = doc.createElement("tenGoc");
    doc.appendChild(root);
}
```



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 5: Tạo phần tử con

```
Element childElement =
doc.createElement("tenCon");
```

- Bước 6: Thiết lập các thuộc tính cho phần tử con

```
childElement.setAttribute("tenThuocTinh",
giaTri);
```

- Bước 6: Thêm một phần tử con vào

```
root.appendChild(childElement);
```



Ghi tài liệu XML



❑ Các bước thực hiện

- Bước 7: Thiết lập thuộc tính tập tin XML

```
Transformer tr =
TransformerFactory.newInstance().newTransformer();
tr.setOutputProperty(OutputKeys.INDENT, "yes");
tr.setOutputProperty(OutputKeys.METHOD, "xml");
tr.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
tr.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");
```







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 5: *Làm việc với CSDL – p1*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



1. Giới thiệu hệ quản trị cơ sở dữ liệu
2. CSDL MySQL
3. CSDL Oracle



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Cơ sở dữ liệu - CSDL (Database - DB)

- Là một tập hợp dữ liệu được lưu trữ một cách có tổ chức nhằm giúp việc xem, tìm kiếm và lấy thông tin được nhanh chóng và chính xác, giúp giảm công sức và thời gian quản lý thông tin cần thiết.



Giới thiệu hệ quản trị cơ sở dữ liệu



□ Cơ sở dữ liệu

- Chức năng

- Lưu trữ
- Truy cập
- Tổ chức
- Xử lý



Giới thiệu hệ quản trị cơ sở dữ liệu



□ Hệ quản trị CSDL (Database

Management System - DBMS)

- Là phần mềm hay hệ thống được thiết kế để quản trị CSDL.
- Hệ QT CSDL hỗ trợ khả năng lưu trữ, sửa chữa, xóa và tìm kiếm thông tin trong CSDL.
- Có rất nhiều loại hệ quản trị CSDL khác nhau: MySQL, Oracle, SQL Server...



Giới thiệu hệ quản trị cơ sở dữ liệu



□ Hệ QT CSDL

- Đa số hệ QT CSDL đều có một đặc điểm chung là sử dụng ngôn ngữ truy vấn có cấu trúc (*Structured Query Language - SQL*)
- Ưu điểm của HQTCSDL:
 - Quản lý dữ liệu
 - Đảm bảo tính nhất quán cho dữ liệu
 - Tạo khả năng chia sẻ dữ liệu
 - Cải tiến tính toàn vẹn cho dữ liệu



Giới thiệu hệ quản trị cơ sở dữ liệu



□ Các từ khóa SQL

- SELECT: được sử dụng để *lấy dữ liệu* từ một hoặc nhiều bảng trong cơ sở dữ liệu
 - SELECT là lệnh thường dùng nhất
 - Trong việc tạo ra câu truy vấn SELECT, người dùng phải đưa ra mô tả cho những dữ liệu mình muốn lấy.



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- Những từ khóa liên quan tới SELECT:

- FROM: chỉ định dữ liệu sẽ được lấy ra từ những bảng nào, và các bảng đó quan hệ với nhau như thế nào.
- WHERE: để xác định những bản ghi nào sẽ được lấy ra theo điều kiện nào đó.



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- Những từ khóa liên quan tới SELECT:

- GROUP BY: nhóm dữ liệu
- HAVING: giới hạn những bản ghi sau khi nhóm dữ liệu với GROUP BY.
- ORDER BY: xác định dữ liệu lấy ra sẽ được sắp xếp theo những cột nào.



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- Ví dụ: Liệt kê danh sách các nhân viên có mức lương tối thiểu lớn hơn 2.600.000 VNĐ sắp theo thứ tự tăng dần của các giá trị trong cột trợ cấp

```
SELECT *
FROM NHAN_VIEN
WHERE LUONG_TOI_THIEU > 2600000
ORDER BY TRO_CAP
```



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- Ví dụ: Liệt kê danh sách các khoa và số sinh viên trong từng khoa

```
SELECT K.TEN_KHOA, COUNT(S.MA_SV) AS
SO_SV
FROM KHOA K INNER JOIN SINH_VIEN S ON
K.MA_KHOA = S.MA_KHOA
GROUP BY K.TEN_KHOA
```



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- **INSERT:** được sử dụng để **thêm dữ liệu**

▪ Ví dụ: Hãy thêm một khách hàng mới với các thông tin: Nguyễn Văn Nam, 357 Lê Hồng Phong p.2 q.10 tp.HCM, 0989 456 123, nvnam@gmail.com

```
INSERT INTO KHACH_HANG(TEN_KH, DIA_CHI,
DIEN_THOAI, EMAIL)
```

```
VALUES ('Nguyễn Văn Nam', '357 Lê Hồng Phong
p.2 q.10 tp.HCM', '0989 456 123',
'nvnam@gmail.com')
```



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- **UPDATE:** được sử dụng để **cập nhật dữ liệu**

▪ Ví dụ: Hãy cập nhật địa chỉ mới cho khách hàng Nguyễn Văn Nam là: 227 Nguyễn Văn Cừ q.5 tp.HCM

```
UPDATE KHACH_HANG
SET DIA_CHI = '227 Nguyễn Văn Cừ q.5 tp.HCM'
WHERE TEN_KH = 'Nguyễn Văn Nam'
```



Giới thiệu hệ quản trị cơ sở dữ liệu



❑ Các từ khóa SQL

- **DELETE:** được sử dụng để **xóa dữ liệu**

- Ví dụ: Hãy xóa khách hàng có tên là Trần Văn Hải

```
DELETE
```

```
FROM KHACH_HANG
```

```
WHERE TEN_KH = 'Trần Văn Hải'
```



Nội dung



1. Giới thiệu hệ quản trị cơ sở dữ liệu
2. CSDL MySQL
3. CSDL Oracle



CSDL MySQL



□ Giới thiệu MySQL

- Là hệ QT CSDL nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng.
- Là CSDL tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh.



CSDL MySQL



□ Giới thiệu MySQL

- Thích hợp cho các ứng dụng có truy cập CSDL trên internet vì có tốc độ và tính bảo mật cao
- Hoàn toàn miễn phí
- Có nhiều phiên bản cho các hệ điều hành khác nhau
- Sử dụng Ngôn ngữ truy vấn có cấu trúc (SQL).



CSDL MySQL



□ Download và cài đặt

- Có thể download cả gói Wamp Server(trong đó MySQL) tại địa chỉ:

<http://www.wampserver.com/en/>

- Hoặc có thể download riêng MySQL tại địa chỉ: <http://www.mysql.com/downloads/>

(*Lưu ý: Giáo viên hướng dẫn học viên cách cài đặt và sử dụng CSDL MySQL*)



Nội dung



1. Giới thiệu hệ quản trị cơ sở dữ liệu
2. CSDL MySQL
3. CSDL Oracle



CSDL Oracle



□ Giới thiệu Oracle

- Là hệ QT CSDL mạnh mẽ nhất thế giới.
- Được thiết kế để triển khai cho mọi môi trường doanh nghiệp.
- Việc cài đặt, quản lý rất dễ dàng
- Phù hợp cho mọi loại dữ liệu, các ứng dụng và các môi trường khác nhau bao gồm cả windows và linux với chi phí tối thiểu.



CSDL Oracle



□ Giới thiệu Oracle

- > 2/3 trong số 500 tập đoàn công ty lớn nhất thế giới (fortune 500) sử dụng oracle.
- Ở Việt Nam hầu hết các đơn vị lớn thuộc các ngành ngân hàng, kho bạc, thuế, bảo hiểm, bưu điện, hàng không, dầu khí,... đều sử dụng hệ QT CSDL Oracle.



CSDL Oracle



□ Giới thiệu Oracle

- Độ ổn định và tin cậy cao
- Khả năng xử lý dữ liệu rất lớn, có thể lên đến hàng trăm terabyte (1 terabyte ~ 1,000 gigabyte ~ 1,000,000,000 kilobyte) mà vẫn đảm bảo tốc độ xử lý dữ liệu rất cao.
- Khả năng bảo mật cao
- Độc lập với hệ điều hành



CSDL Oracle



□ Download và cài đặt

- Có thể download tại địa chỉ:
<http://www.oracle.com/technetwork/databases/enterprise-edition/downloads/index.html>
hoặc liên hệ trực tiếp với Giáo viên để nhận gói cài đặt.

(*Lưu ý: Giáo viên hướng dẫn học viên cách cài đặt và sử dụng CSDL Oracle*)







Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 6: *Làm việc với CSDL – p2*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



1. Khái niệm JDBC
2. Phân loại cầu nối JDBC
3. Qui trình kết nối cơ sở dữ liệu



Khái niệm JDBC



❑ JDBC – Java Database Connectivity

- JDBC cung cấp một thư viện chuẩn API (Application programming interface) truy xuất CSDL
- JDBC API là một chuẩn kết nối CSDL độc lập với CSDL
- Hỗ trợ hầu hết các hệ QT CSDL (RDBMS - Relational Database Management Systems)

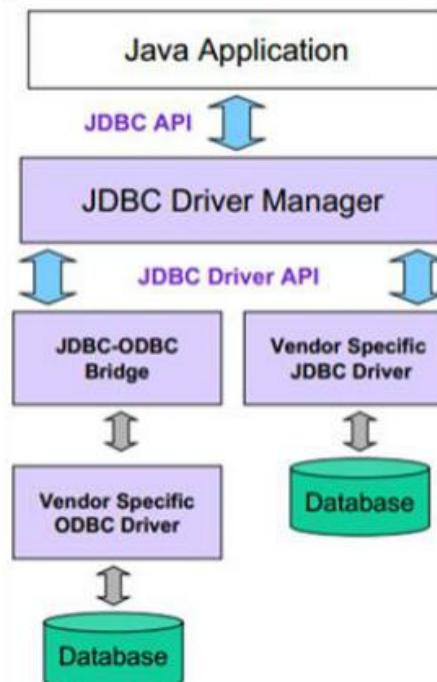


Khái niệm JDBC

❑ Kiến trúc JDBC gồm

2 lớp:

- JDBC API
- Cầu nối JDBC (JDBC driver manager): chịu trách nhiệm giao tiếp với cầu nối của từng loại hệ QT CSDL



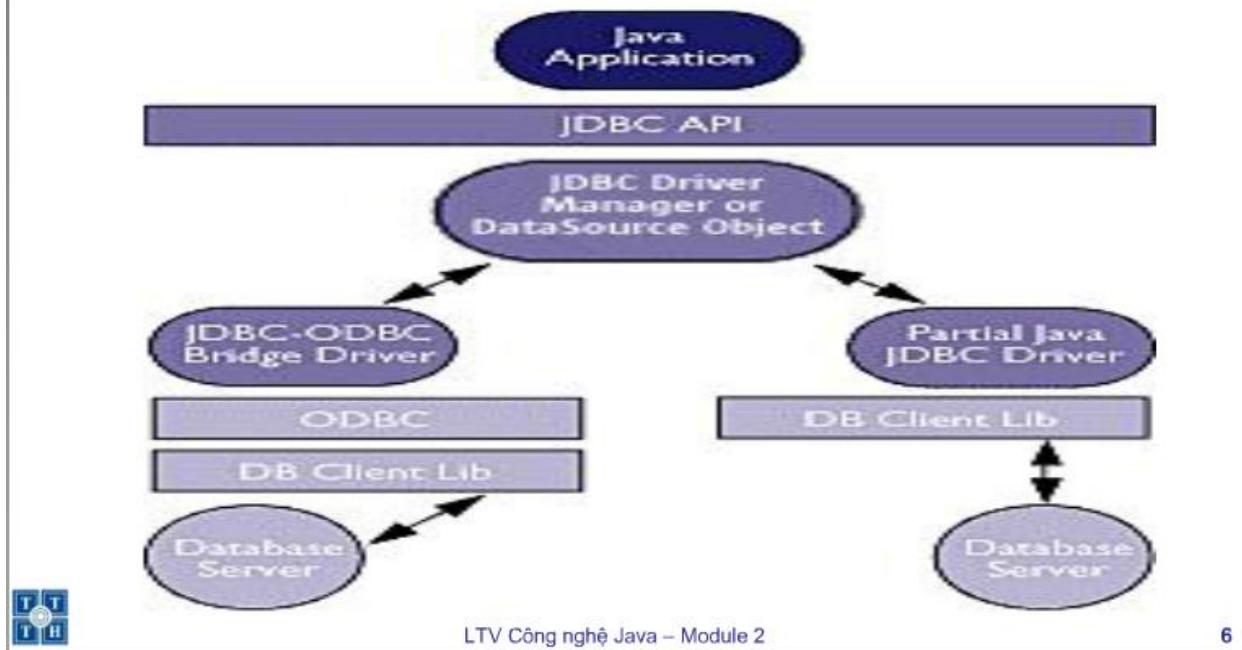
Nội dung

1. Khái niệm JDBC
2. Phân loại cầu nối JDBC
3. Qui trình kết nối cơ sở dữ liệu

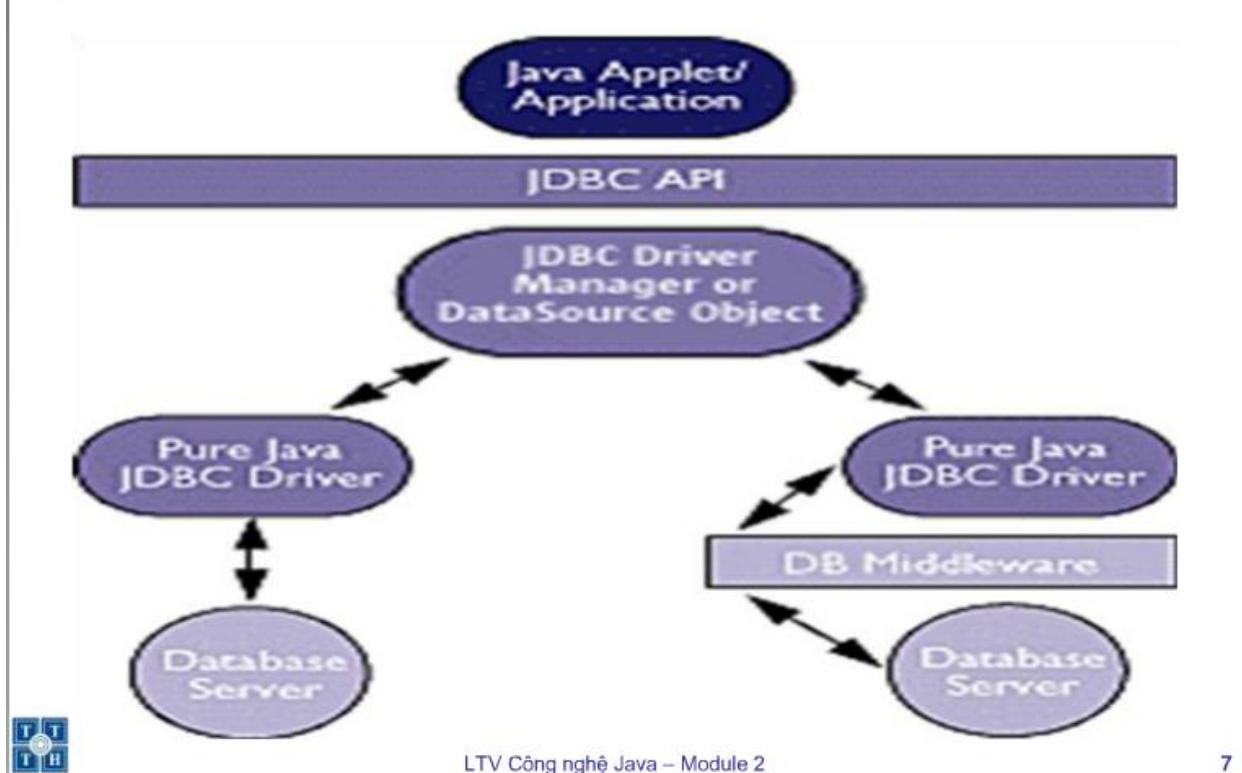


Phân loại cầu nối JDBC

□ Có 4 loại cầu nối



Phân loại cầu nối JDBC



Phân loại câu nối JDBC



❑Loại 1:

- Trình điều khiển câu nối JDBC-ODBC
- Cầu nối hiện thực ánh xạ JDBC API tới ODBC
- Phụ thuộc vào thư viện tự nhiên, không linh hoạt
- Trình điều khiển chậm nhất so với các trình điều khiển câu nối khác



Phân loại câu nối JDBC



❑Loại 2:

- Câu nối được viết bằng một phần ngôn ngữ lập trình Java và mã nền của dữ liệu nguồn
- Những câu nối dùng thư viện riêng của cơ sở dữ liệu mà nó kết nối

❑Loại 3:

- Câu nối dùng Java thuần và dùng giao thức độc lập để kết nối đến hệ QT CSDL thông qua một Middleware Server



Phân loại cầu nối JDBC



□Loại 4:

- Được viết thuần bằng Java và kết nối trực tiếp xuống CSDL
- Cho kết quả nhanh và tối ưu
- Được sử dụng rộng rãi



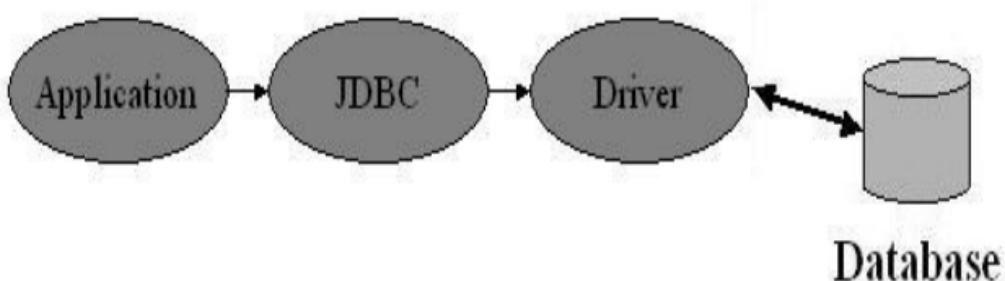
Nội dung



1. Khái niệm JDBC
2. Phân loại cầu nối JDBC
3. Qui trình kết nối cơ sở dữ liệu



Qui trình kết nối CSDL



Qui trình kết nối CSDL



❑ Gồm 7 bước:

- Bước 1: Nạp trình điều khiển cầu nối tương ứng cho loại quản trị CSDL kết nối
- Bước 2: Tạo chuỗi URL
- Bước 3: Thiết lập kết nối
- Bước 4: Tạo ra đối tượng lớp Statement
- Bước 5: Thực thi câu lệnh truy vấn của đối tượng Statement
- Bước 6: Xử lý kết quả
- Bước 7: Đóng kết nối



Qui trình kết nối CSDL



□ Bước 1: Nạp trình điều khiển cầu nối

```

try {
    Class.forName("connect.microsoft.MicrosoftDriver");
    Class.forName("oracle.jdbc.driver.OracleDriver");
} catch { ClassNotFoundException cnfe) {
    System.out.println("Error loading driver: " cnfe);
}

String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
    ":" + port + ":" + dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host +
    ":" + port + ":" +
    "?SERVICENAME=" + dbName;

```

□ Bước 2: Tạo chuỗi URL



Qui trình kết nối CSDL



□ Bước 3: Thiết lập kết nối

```

String username = "jay_debesee";
String password = "secret";
Connection connection =
    DriverManager.getConnection(oracleURL,
        username,
        password);

DatabaseMetaData dbMetaData =
    connection.getMetaData();
String productName =
    dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =
    dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);

```

□ Xem thông tin về CSDL đã kết nối



Qui trình kết nối CSDL



□ Bước 4: Tạo đối tượng Statement

```
Statement statement =
    connection.createStatement();
String query =
    "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet =
    statement.executeQuery(query);
```

□ Bước 5: Thực thi câu truy vấn

- Để thực thi câu truy vấn với tác vụ insert, update, delete thì dùng phương thức executeUpdate()



Qui trình kết nối CSDL



□ Bước 6: Xử lý kết quả

```
while(resultSet.next()) {
    System.out.println(resultSet.getString(1) + " " +
        resultSet.getString(2) + " " +
        resultSet.getString(3));
}
```

- Chỉ số cột bắt đầu bằng 1, tức cột đầu tiên là 1
- Giao diện ResultSet cung cấp phương thức getXXX() với đối số là chỉ số hoặc tên cột để truy cập giá trị của cột đó



Qui trình kết nối CSDL



□ Bước 7: Đóng kết nối

```
connection.close();
```

- Luôn luôn đóng kết nối sau khi thực thi xong kết nối để quá trình kết nối an toàn
- Quá trình kết nối là tốn chi phí



Qui trình kết nối CSDL



```
import java.sql.*

public class JDBCInformation {
    static String userid="scott", password = "tiger";
    static String url = "jdbc:odbc:example";
    static Connection con = null;
    public static void main(String[] args) throws Exception {
        Connection con = getJDBCConnection();
        if(con!= null){
            System.out.println("Got Connection.");
            DatabaseMetaData meta = con.getMetaData();
            System.out.println("Driver Name: "+meta.getDriverName());
            System.out.println("Driver Version:
                "+meta.getDriverVersion());
        }else{
            System.out.println("Could not Get Connection");
        }
    }
}
```



Qui trình kết nối CSDL



```

public static Connection getJDBCConnection(){

    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    } try {
        con = DriverManager.getConnection(url, userid,
password);
    } catch(SQLException ex) {
        System.err.println("SQLException: " +
ex.getMessage());
    }
    return con;
}

}

```



Qui trình kết nối CSDL



❑ Lớp PreparedStatement

- Đối tượng PreparedStatement chứa phát biểu truy vấn đã được biên dịch trước
- Đối tượng PreparedStatement không những sử dụng câu truy vấn không có đối số mà còn những câu truy vấn có đối số
 - Có thể sử dụng cùng một câu truy vấn, nhưng ứng dụng với các giá trị khác nhau



Qui trình kết nối CSDL



❑ Dùng Statement

```
String updateString = "UPDATE COFFEES SET SALES = 75 " +
    "WHERE COF_NAME LIKE 'Colombian'";
stmt.executeUpdate(updateString);

PreparedStatement updateSales = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
```

❑ Dùng PreparedStatement



Qui trình kết nối CSDL



❑ Phiên giao dịch (Transaction)

- Mặc định khi câu lệnh truy vấn thực thi hoàn thành thì tự động cập nhật dữ liệu vào CSDL (commit)
 - Gọi là chế độ auto-commit
- Phiên giao dịch là cách cho phép một hay nhiều phát biểu được nhóm lại thành một phiên giao dịch
 - Có thể chuyển chế độ auto-commit sang trạng thái ẩn (false)



```

//turn off commit mode
con.setAutoCommit(false);
//create prepared statement
PreparedStatement updateSales = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 50);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
//create another prepared statement
PreparedStatement updateTotal = con.prepareStatement(
    "UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE ?");
updateTotal.setInt(1, 50);
updateTotal.setString(2, "Colombian");
updateTotal.executeUpdate();
//commit a transaction
con.commit();
//return default auto-commit mode
con.setAutoCommit(true);

```



Qui trình kết nối CSDL

❑ Connect pooling

- Là một kỹ thuật để nâng cao hiệu quả truy xuất CSDL
- Một ứng dụng có khi chi phí mở kết nối lớn hơn nhiều chi phi thực thi. Do đó, kỹ thuật Connect pooling lưu trữ các kết nối trước đó một khoảng thời gian và hạn chế tạo ra kết nối mới
- Phương pháp này hiện thực một proxy để chặn tất cả các kết nối đến CSDL



Qui trình kết nối CSDL



❑ Connect pooling

- Cụ thể lớp Connection có phương thức getConnection() trả về một đối tượng proxy
 - Thực tế, những kết nối không kết nối trực tiếp đến CSDL nhưng sẽ chia sẻ luồng kết nối
- Kết nối mới tạo ra khi không tồn tại kết nối tương tự trước đó hoặc kết nối tương tự đã vượt quá thời gian lưu trữ cho phép (timeout)





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 7: *Bảo mật ứng dụng*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung

1. Tổng quan Bảo mật ứng dụng
2. Xác thực người dùng (Authentication)
3. Phân quyền người dùng (Authorization)
4. Thiết kế bảo mật Ứng dụng



Tổng quan bảo mật ứng dụng



Nội dung



1. Tổng quan Bảo mật ứng dụng
2. Xác thực người dùng (Authentication)
3. Phân quyền người dùng (Authorization)
4. Thiết kế bảo mật Ứng dụng



Xác thực người dùng (Authentication)



Xác thực người dùng (Authentication)



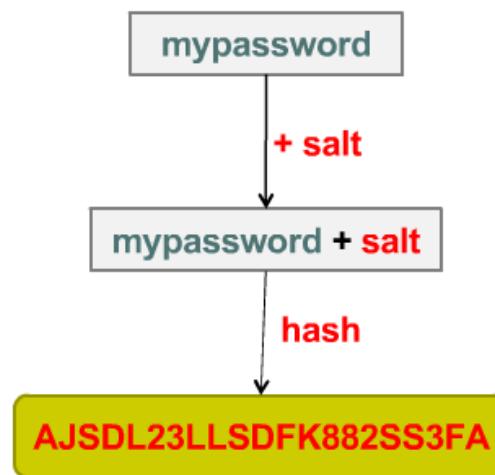
Xác thực người dùng (Authentication)



Xác thực người dùng (Authentication)



Xác thực người dùng (Authentication)



Xác thực người dùng (Authentication)



Xác thực người dùng (Authentication)



password +



A	B	C	D	E	F	G	H
126	306	406	671	418	588	373	305
726	140	216	216	312	512	597	457
189	569	963	237	947	511	272	295
334	440	125	478	224	161	908	679
364	260	376	246	498	877	738	998
762	682	871	641	932	792	899	336
789	110	854	427	804	268	842	381
164	977	857	782	886	940	100	354



Xác thực người dùng (Authentication)



```
String password= "myPassword";  
  
MessageDigest md =  
MessageDigest.getInstance("MD5");  
  
md.reset();  
  
md.update(password );  
  
byte byteData[] = md.digest();  
  
String hashPassword =  
DatatypeConverter.printHexBinary(md5);
```



Nội dung



1. Tổng quan Bảo mật ứng dụng
2. Xác thực người dùng (Authentication)
3. Phân quyền người dùng (Authorization)
4. Thiết kế bảo mật Ứng dụng



Phân quyền người dùng (Authorization)



Phân quyền người dùng (Authorization)



Nội dung

1. Tổng quan Bảo mật ứng dụng
2. Xác thực người dùng (Authentication)
3. Phân quyền người dùng (Authorization)
4. Thiết kế bảo mật Ứng dụng



Thiết kế bảo mật Ứng dụng

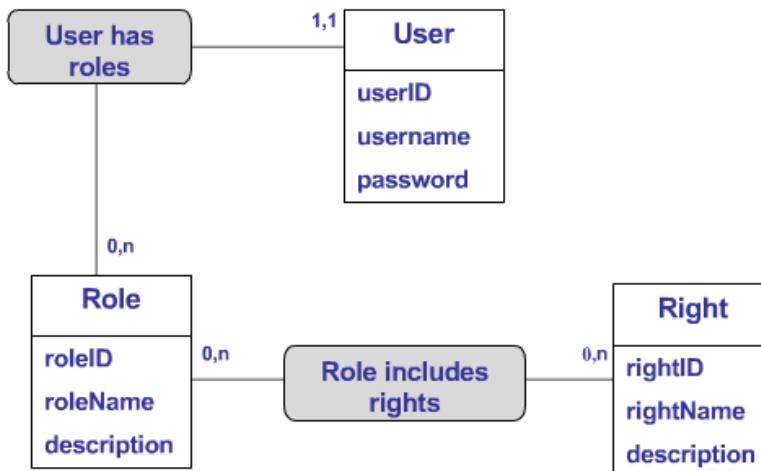


Thiết kế bảo mật Ứng dụng



□ Thiết kế

- Password phải được mã hóa trước khi lưu vào database bằng thuật toán mã hóa 1 chiều





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 8: *Design Pattern - p1*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



1. Giới thiệu

2. Phân loại

3. Decorator

4. Proxy

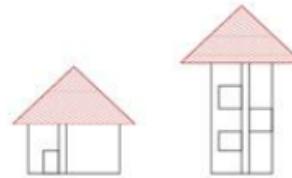


Giới thiệu



❑ Ngữ cảnh

- Kiến trúc sư Christopher Alexander (“Timeless way of building”, 1979) đã phát triển ý tưởng để xác định các đối tượng khi đưa ra: Một giải pháp cho một vấn đề trong một ngữ cảnh - “A solution to a problem in a context”.



- Để xác định một mẫu (pattern), cần đưa ra lý do và cách thức để xây dựng từng giải pháp.



Giới thiệu



- ❑ Design Pattern lần đầu được giới thiệu bởi Gang of Four (GoF) : Gamma Erich (PhD thesis), Richard Helm, Ralph Johnson và John Vlissides (1995).
- ❑ Đây là một nét “văn hóa mới” trong cộng đồng lập trình
- ❑ Pattern là thiết bị cho phép các chương trình chia sẻ kiến thức về thiết kế. Khi xây dựng chương trình, có nhiều vấn đề gấp phải và nó xuất hiện lại thường xuyên.
- ❑ Thư viện các pattern là nơi để chúng ta có thể tìm thấy “Các giải pháp lập trình tốt nhất.



Giới thiệu



❑ Khái niệm

- Mẫu thiết kế (Design Pattern) là vấn đề thông dụng cần giải quyết và là cách giải quyết vấn đề đó trong một ngữ cảnh cụ thể
- Mẫu thiết kế không đơn thuần là một bước nào đó trong các giai đoạn phát triển phần mềm mà nó đóng vai trò là sáng kiến để giải quyết một vấn đề thông dụng nào đó.
- Mẫu thiết kế sẽ giúp cho việc giải quyết vấn đề nhanh, gọn và hợp lý hơn.
- Mẫu thiết kế còn được sử dụng nhằm cô lập các thay đổi trong mã nguồn, từ đó làm cho hệ thống có khả năng tái sử dụng cao.



Giới thiệu Design Pattern



□ Lý do nên dùng Design Pattern

- Tái sử dụng: Việc thiết kế một phần mềm hướng đối tượng phục vụ cho mục đích dùng lại là rất khó; cần phải xác định được có những lớp đối tượng nào, quan hệ giữa chúng ra sao, có kế thừa hay không,... Thiết kế phải đảm bảo không chỉ giải quyết được các vấn đề hiện tại, mà còn có thể tiến hành mở rộng trong tương lai. Vì vậy, nếu phần mềm không có một thiết kế tốt, việc sau này khi mở rộng phần mềm lại phải thiết kế lại từ đầu rất có thể xảy ra.
- Kinh nghiệm quý báu: Design Pattern là những kinh nghiệm đã được đúc kết từ những người đi trước, việc sử dụng Design Pattern sẽ giúp chúng ta giảm được thời gian và công sức suy nghĩ ra các giải pháp để giải quyết những vấn đề đã có lời giải.



Nội dung



1. Giới thiệu
2. Phân loại
3. Decorator
4. Proxy



Phân loại



□ Hệ thống các mẫu design pattern
được chia thành 3 nhóm dựa theo vai
trò

Creational	Structural	Behavioral
Abstract Factory, Factory, Builder, Prototype, Singleton	Adapter, Bridge, Composite, Decorator, Facade, Proxy, Flyweight	Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor.



Phân loại



□ Trong đó:

- Nhóm Creational (nhóm kiến tạo)
 - Hỗ trợ cho việc **khởi tạo đối tượng** trong hệ thống: khởi tạo một đối tượng cụ thể từ một định nghĩa trừu tượng (abstract, class, interface).
 - Giúp khắc phục những vấn đề khởi tạo đối tượng, hạn chế sự phụ thuộc vào platform



Phân loại



● Nhóm Structural (nhóm cấu trúc)

- Các lớp đối tượng kết hợp với nhau tạo thành cấu trúc lớn hơn
- Cung cấp cơ chế xử lý những lớp không thể thay đổi, ràng buộc muộn (late binding) và giảm kết nối (lower coupling) giữa các thành phần và cung cấp các cơ chế khác để kế thừa.
- Diễn tả một cách có hiệu quả cả việc phân chia hoặc kết hợp các phần tử trong một ứng dụng



Phân loại



● Nhóm Behavioral (nhóm hành vi)

- Cách thức để các lớp và đối tượng có thể giao tiếp với nhau
- Che giấu hiện thực của đối tượng, che giấu giải thuật, hỗ trợ việc thay đổi cấu hình đối tượng một cách linh động.
- Có liên quan đến luồng điều khiển của hệ thống. Một vài cách của tổ chức điều khiển bên trong một hệ thống có thể mang lại các lợi ích cả về hiệu suất lẫn khả năng bảo trì hệ thống đó



Nội dung



1. Giới thiệu
2. Phân loại
3. Decorator
4. Proxy



Decorator



□ Giới thiệu

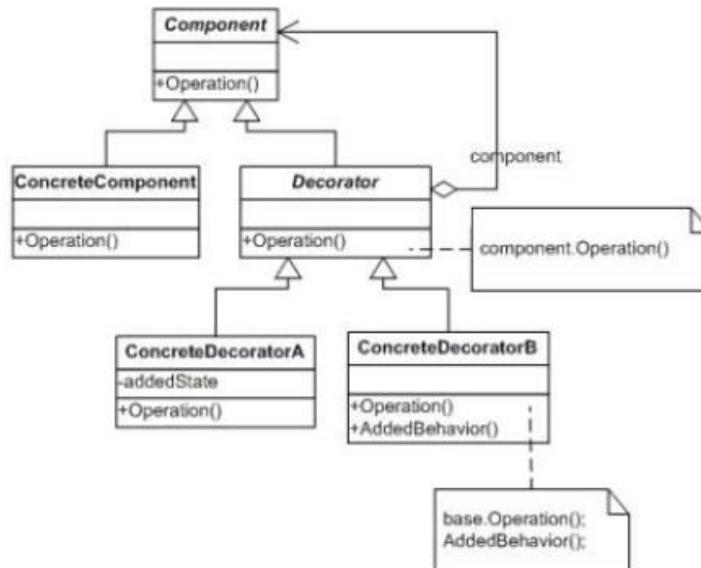
- Là mẫu dùng để bổ sung trách nhiệm cho đối tượng tại thời điểm thực thi.
- Được xem là sự thay thế hiệu quả cho phương pháp kế thừa trong việc bổ sung trách nhiệm cho đối tượng và mức tác động là ở mức đối tượng thay vì ở mức lớp như phương pháp kế thừa.



Decorator



Cấu trúc mẫu



Decorator



```
public interface Component {  
    void Operation();  
}
```

```
public class Concrete
implements Component {

@Override
public void Operation() {
    System.out.println("This
is Concrete Component");
}

}
```

```
public abstract class Decorator  
implements Component {  
  
    protected Component  
decoratedComponent;  
  
    public Decorator(Component  
decoratedComponent) {  
  
        this.decoratedComponent =  
decoratedComponent;  
  
    }  
  
    public void Operation() {  
  
decoratedComponent.Operation();  
  
    }  
}
```



Decorator

```
public class ConcreteDecoratorA extends Decorator {
    public ConcreteDecoratorA(Component decoratedComponent) {
        super(decoratedComponent);
    }
    @Override
    public void Operation() {
        decoratedComponent.Operation();
        addedState(decoratedComponent);
    }
    private void addedState(Component decoratedComponent) {
        System.out.println("This is Concrete Decorator A");
    }
}
// làm tương tự cho ConcreteDecoratorB
```



Decorator

- Trong đó:

- Component: là một interface chứa các phương thức ảo (ở đây là defaultMethod)
- ConcreteComponent: là một lớp kế thừa từ Component, cài đặt các phương thức cụ thể (defaultMethod được cài đặt tường minh)
- Decorator: là một lớp ảo kế thừa từ Component đồng thời cũng chứa 1 thê hiện của Component, phương thức defaultMethod trong Decorator sẽ được thực hiện thông qua thê hiện này.
- ConcreteDecoratorX: là các lớp kế thừa từ Decorator, khai báo tường minh các phương thức, đặc biệt trong các lớp này khai báo tường minh các “trách nhiệm” cần thêm vào khi run-time



Decorator



□ Áp dụng

- Khi muốn thay đổi động mà không ảnh hưởng đến người dùng, không phụ thuộc vào giới hạn các lớp con
- Khi muốn thành phần có thể thêm vào hoặc rút bỏ đi khi hệ thống đang chạy
- Có một số đặc tính phụ thuộc mà muốn ứng dụng một cách động và muốn kết hợp chúng vào trong một thành phần



Decorator



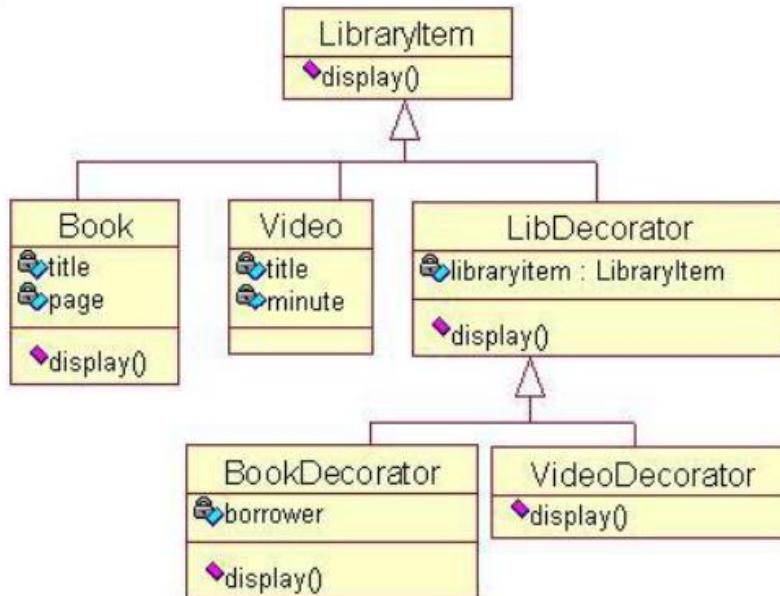
□ Ví dụ 1:

- Trong thư viện có các tài liệu: sách, video... Các loại tài liệu này có các thuộc tính khác nhau, phương thức hiển thị của giao tiếp LibraryItem sẽ hiển thị các thông tin này. Giả sử, ngoài các thông tin thuộc tính trên, đôi khi ta muốn hiển thị độc giả mượn một cuốn sách (chức năng hiển thị độc giả này không phải lúc nào cũng muốn hiển thị), hoặc muốn xem đoạn video(không thường xuyên).



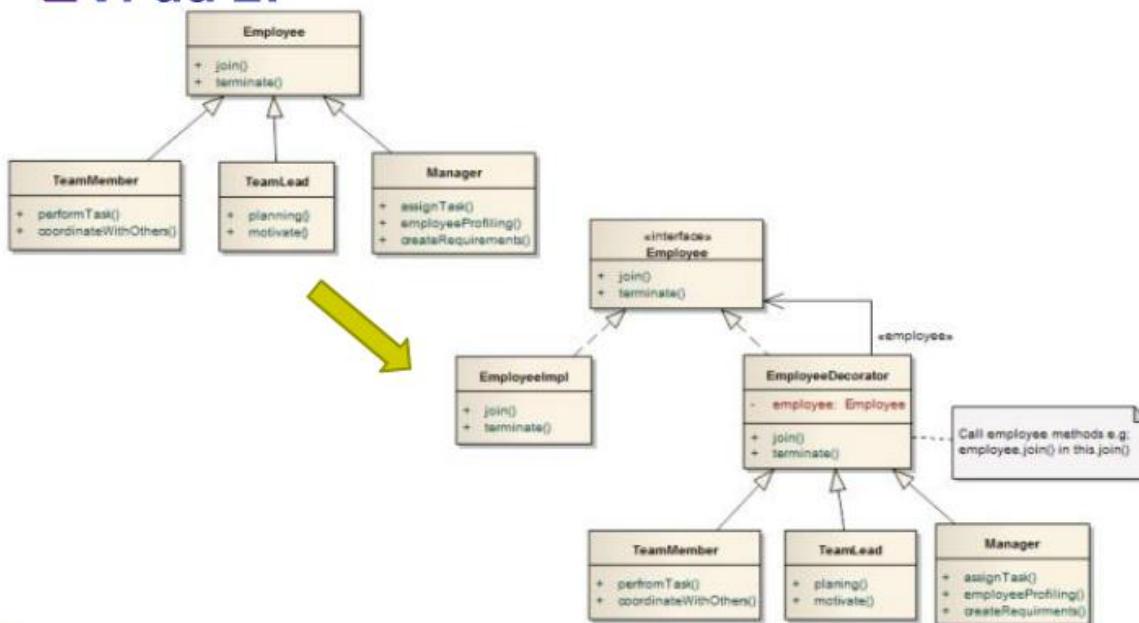
Decorator

❑ Ví dụ 1



Decorator

❑ Ví dụ 2:



Nội dung



1. Giới thiệu
2. Phân loại
3. Decorator
4. Proxy



Proxy



□ Giới thiệu

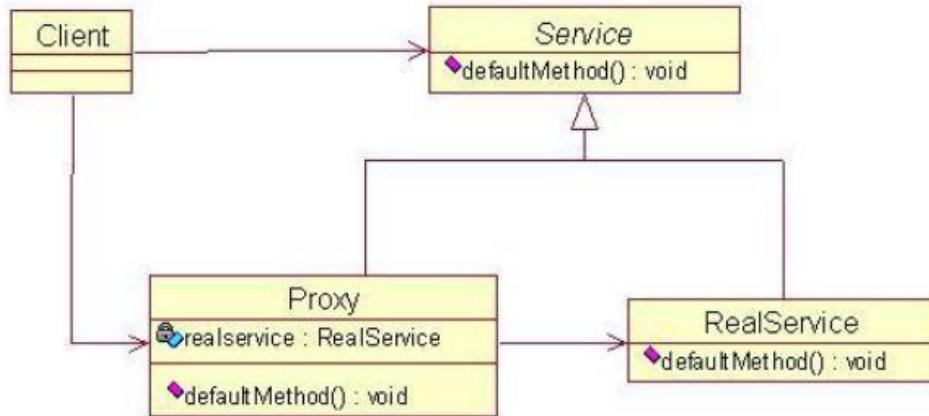
- Đại diện một đối tượng phức tạp bằng một đối tượng đơn giản, vì các mục đích truy xuất, tốc độ và bảo mật
- Là một mẫu thiết kế mà ở đó tất cả các truy cập trực tiếp một đối tượng nào đó sẽ được chuyển hướng vào một đối tượng trung gian



Proxy



□ Cấu trúc mẫu



Proxy



```

public interface Service {
    void defaultMethod();
}
  
```

```

public class RealService implements Service {
    // khởi tạo RealService
    public RealService(...){
        ...
    }
    @Override
    public void defaultMethod() {
        // code của việc sẽ thực hiện
        // trong RealService
    }
    ...
}
  
```

```

public class Proxy implements Service{
    private RealService realService;
    public Proxy (...){
        ...
    }
    @Override
    public void defaultMethod() {
        if(realService == null){
            realService = new
            RealService(...);
        }
        realService.defaultMethod ();
    }
}
  
```



Proxy



- Trong đó:

- **Service**: là giao tiếp định nghĩa các phương thức chuẩn cho một dịch vụ nào đó
- **RealService**: là một thực thi của giao tiếp Service, lớp này sẽ khai báo tường minh các phương thức của Service, lớp này xem như thực hiện tất cả các yêu cầu từ Service
- **Proxy**: kế thừa Service và sử dụng đối tượng của RealService



Proxy



□ Áp dụng

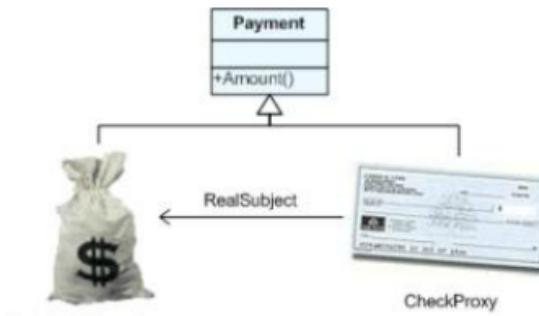
- Sử dụng mẫu Proxy khi bạn cần một tham chiếu phức tạp đến một đối tượng thay vì chỉ một cách bình thường
- Remote proxy – sử dụng khi bạn cần một tham chiếu định vị cho một đối tượng trong không gian địa chỉ(JVM)
- Virtual proxy – lưu giữ các thông tin thêm vào về một dịch vụ thực vì vậy chúng có thể hoãn lại sự truy xuất vào dịch vụ này
- Protection proxy – xác thực quyền truy xuất vào một đối tượng thực



Proxy



Ví dụ: Proxy cung cấp một vật thay thế hoặc một nơi lưu trữ để cung cấp quyền truy cập vào một đối tượng. Một tấm séc hay ngân phiếu là một proxy cho các quỹ trong một tài khoản. Tấm séc có thể được sử dụng thay cho tiền mặt khi mua sắm và kiểm soát truy cập tài khoản của người phát hành.





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

[Go Screen Capture](#)

LTV CÔNG NGHỆ JAVA

Module 2 – Bài 9: *Design Pattern – p2*

Ngành LT & CSDL

www.t3h.vn



2014

5014



Nội dung



1. Observer
2. Singleton
3. Factory



Observer



□ Giới thiệu

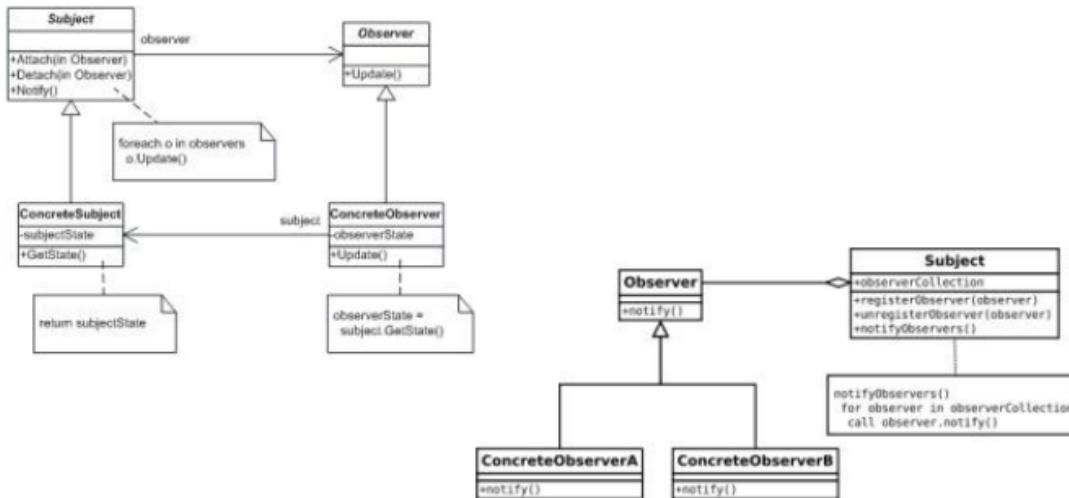
- Observer là một pattern được dùng trong trường hợp ta muốn cài đặt những lớp - đối tượng phụ thuộc vào đối tượng khác, gọi là các đối tượng quan sát (observer) và đối tượng được quan sát (observable).
- Khi trạng thái của đối tượng được quan sát thay đổi thì những đối tượng quan sát sẽ thực hiện hành động nào đó.
- Pattern này cũng được dùng khá phổ biến.





Observer

□ Cấu trúc mẫu



Observer

● Trong đó

- Observable - interface hoặc abstract class xác định các hoạt động để gắn hoặc gắn lại observer cho client, còn được gọi là Account
- ConcreteObservable - concrete Observable class. Nó duy trì trạng thái của đối tượng, khi có một thay đổi trong trạng thái xuất hiện, nó sẽ thông báo cho các Observer khác kèm theo.
- Observer - interface hoặc abstract class định nghĩa các operation được sử dụng để thông báo cho đối tượng này.
- ConcreteObserverA, ConcreteObserverB - lớp cụ thể được kế thừa từ lớp Observer



Observer



□ Áp dụng

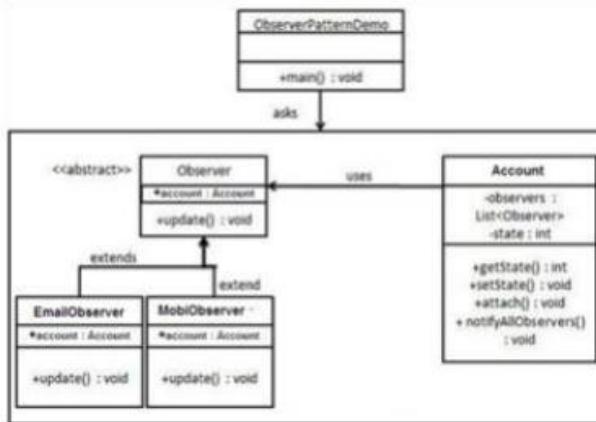
- Khi một trừu tượng (abstraction) có hai khía cạnh (aspect), cái nọ phụ thuộc vào cái kia
- Đóng gói các khía cạnh này trong các đối tượng riêng biệt cho phép thay đổi và tái sử dụng chúng một cách độc lập
- Khi có một thay đổi đến một đối tượng yêu cầu thay đổi các đối tượng khác



Observer



- Ví dụ: Đối tượng được quan sát sẽ là đối tượng Account, khi nó có thay đổi trạng thái thì các đối tượng quan sát là MobiPhone và Email sẽ thực hiện hành động tương ứng là gửi SMS và gửi email thông báo.



Observer

```
public class Account {
    private List<Observer> observers = new ArrayList<Observer>();
    private int state;
    public int getState() {
        return state;
    }
    public void setState(int state) {
        this.state = state;
        notifyAllObservers();
    }
    public void attach(Observer observer){
        observers.add(observer);
    }
    public void notifyAllObservers(){
        for (Observer observer : observers) {
            observer.update();
        }
    }
}
```



Observer

```
public abstract class Observer {
    protected Account account;
    public abstract void update();
}
```

```
public class EmailObserver extends Observer{
    public EmailObserver(Account account){
        this.account = account;
        this.account.attach(this);
    }
    @Override
    public void update() {
        // nội dung của Email
    }
}
```

```
public class MobiObserver extends Observer{
    public MobiObserver(Account account){
        this.account = account;
        this.account.attach(this);
    }
    @Override
    public void update() {
        // nội dung của Mobi
    }
}
```



Nội dung



1. Observer
2. Singleton
3. Factory



Singleton



□ Giới thiệu

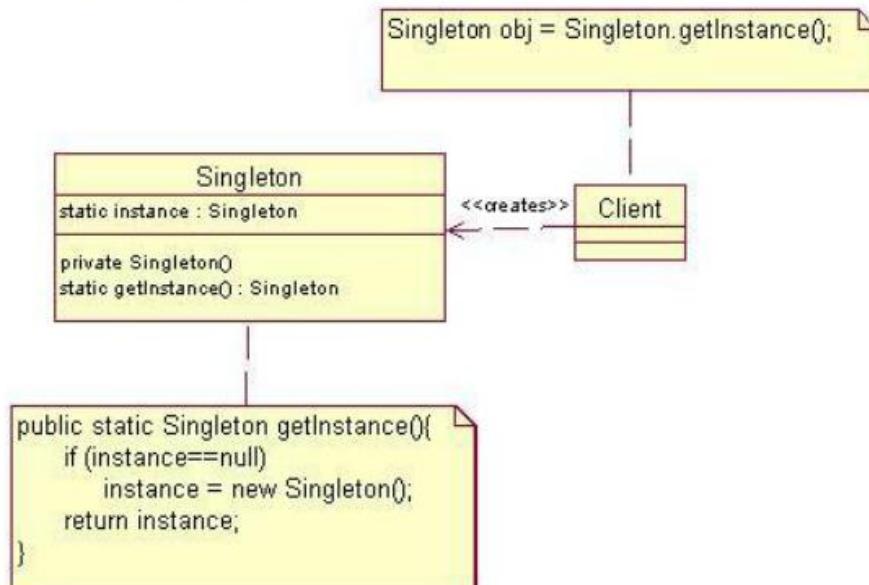
- Mẫu này được thiết kế để đảm bảo cho một lớp chỉ có thể tạo ra duy nhất một thể hiện của nó



Singleton



□ Cấu trúc mẫu



Singleton



```

public class Singleton {
    private static Singleton instance = new Singleton();
    private Singleton(){}
    public static Singleton getInstance(){
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
    ...
}
  
```

```

//client
SingleObject object = SingleObject.getInstance();
...
  
```



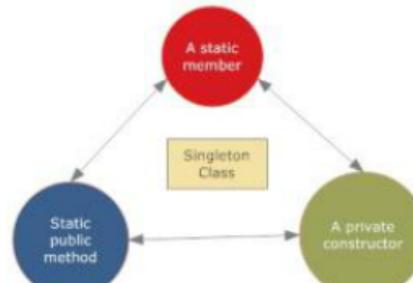
Singleton



- Trong đó

- **Singleton** cung cấp một phương thức tạo private, duy trì một thuộc tính tĩnh để tham chiếu đến một thể hiện của lớp Singleton này, và nó cung cấp thêm một phương thức tĩnh trả về thuộc tính tĩnh này

Singleton implementation



Singleton



- Áp dụng

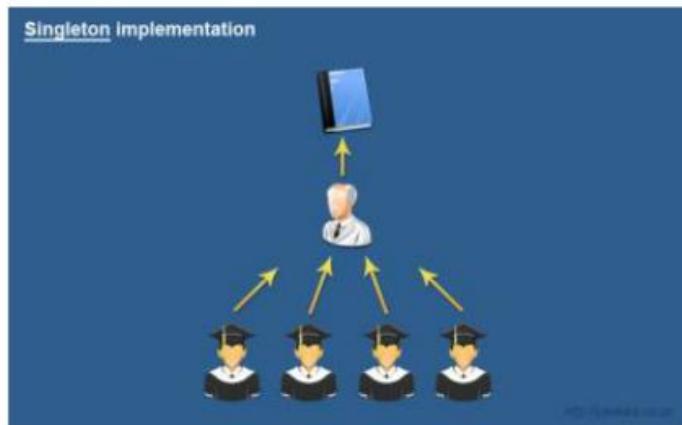
- Khi muốn lớp chỉ có 1 thể hiện duy nhất và nó có hiệu lực ở mọi nơi



Singleton



❑ Ví dụ:



Bằng cách sử dụng mẫu Singleton, lớp Teacher đã được khởi tạo chỉ một lần và sau đó mỗi đối tượng Student nhận được một tài liệu tham khảo của instance đó



Nội dung



1. Observer
2. Singleton
3. Factory



Factory



□ Giới thiệu

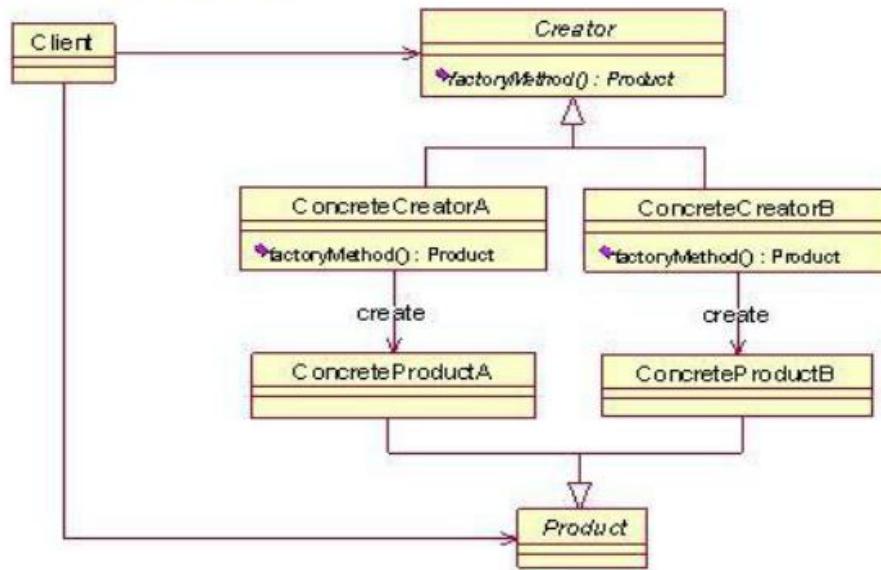
- Bản chất của mẫu thiết kế Factory là "Định nghĩa một giao diện (interface) cho việc tạo một đối tượng, nhưng để các lớp con quyết định lớp nào sẽ được tạo. "Factory method" giao việc khởi tạo một đối tượng cụ thể cho lớp con.



Factory



□ Cấu trúc mẫu



Factory

```
public interface Product {
    void create();
}

public class ConcreteProductA implements Product {
    ...
    @Override
    public void create() {
        // cách tạo ConcreteProductA
    }
}

public class ConcreteProductB implements Product {
    ...
    @Override
    public void create() {
        // cách tạo ConcreteProductB
    }
}
```



Factory

```
public interface Creator{
    Product factoryMethod();
}

public class ConcreteCreatorA implements Creator {
    ...
    @Override
    public Product factoryMethod() {
        // Trả về ConcreteProductA
    }
}

public class ConcreteCreatorB implements Creator {
    ...
    @Override
    public Product factoryMethod() {
        // Trả về ConcreteProductB
    }
}
```



Factory



● Trong đó

- Creator là lớp trừu tượng, khai báo phương thức factoryMethod() nhưng không cài đặt
- Product cũng là lớp trừu tượng
- ConcreteCreatorA và ConcreteCreatorB là 2 lớp kế thừa từ lớp Creator để tạo ra các đối tượng riêng biệt
- ConcreteProductA và ConcreteProductB là các lớp kế thừa của lớp Product, các đối tượng của 2 lớp này sẽ do 2 lớp ConcreteCreatorA và ConcreteCreatorB tạo ra



Factory



□ Áp dụng

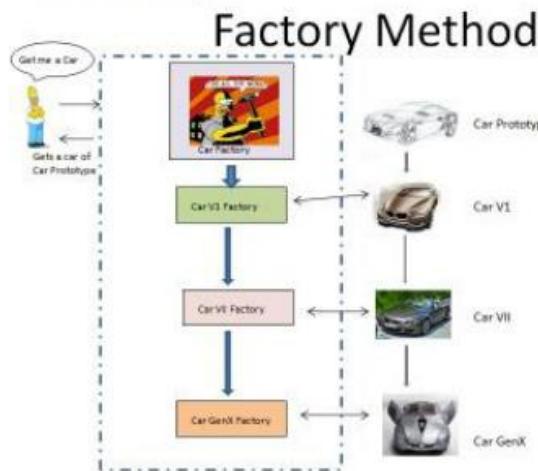
- Khi chúng ta có một lớp “cha” và nhiều lớp “con”, và dựa trên dữ liệu được cung cấp, chúng ta trả về một đối tượng là một trong những lớp “con”
- Khi muốn tạo ra một framework có thể mở rộng, có nghĩa là cho phép tính mềm dẻo trong một số quyết định như chỉ ra loại đối tượng nào được tạo ra
- Khi muốn 1 lớp con, mở rộng từ 1 lớp cha, quyết định lại đối tượng được khởi tạo
- Khi ta biết khi nào thì khởi tạo một đối tượng nhưng không biết loại đối tượng nào được khởi tạo
- Khi cần một vài khai báo chòng phương thức tạo với danh sách các tham số như nhau, điều mà Java không cho phép. Thay vì điều đó ta sử dụng các Factory Method với các tên khác nhau





Factory

❑ Ví dụ



Factory Method Pattern

