Bài 04

Các tính chất trong lập trình hướng đối tượng

Nội dung bài học

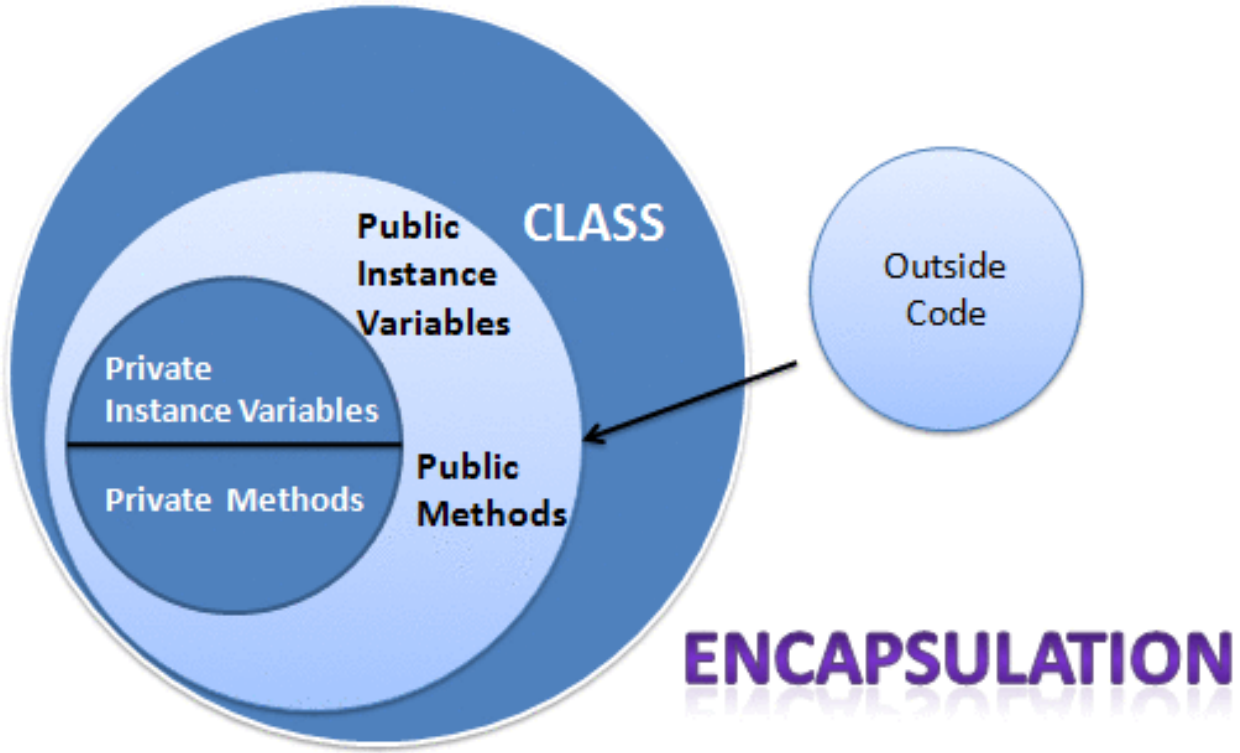
❖ Ôn tập các kiến thức đã học

- FOP – OOP
- Class, object, attributes, methods
- Access modifier getter setter
- Constructor, destructor
- Static, Non Static, this

❖ Tính chất lập trình hướng đối tượng

1. Đóng gói - Encapsulation
2. Đa hình - Polymorphism
3. Thừa kế - Inheritance
4. Trừu tượng – Abstraction

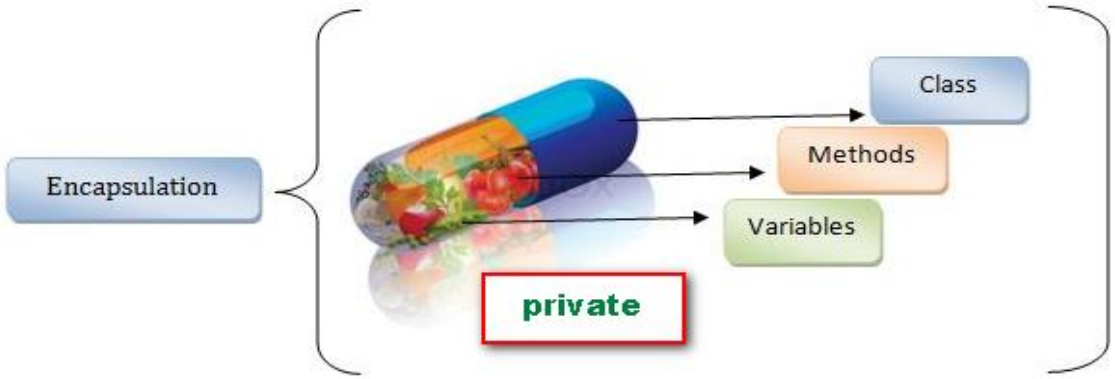
Encapsulation



Encapsulation

- Là kĩ thuật **ẩn dấu thông tin** bên trong class.
- Được sử dụng để **đóng gói dữ liệu** vào một class, package, project để dễ quản lý
- Bảo vệ các trạng thái bên trong của một đối tượng, việc chỉnh sửa đối tượng được thực hiện thông qua các phương thức
- Không tiện cho người không có trách nhiệm truy cập trực tiếp được - **che dấu dữ liệu**
- Ngoài ra có thể gom thành các pakage, module, namespace tùy ngôn ngữ
- Chỉ cần input => output: No talk active, tương tự query.
- Giảm độ phức tạp trong phát triển phần mềm

Encapsulation



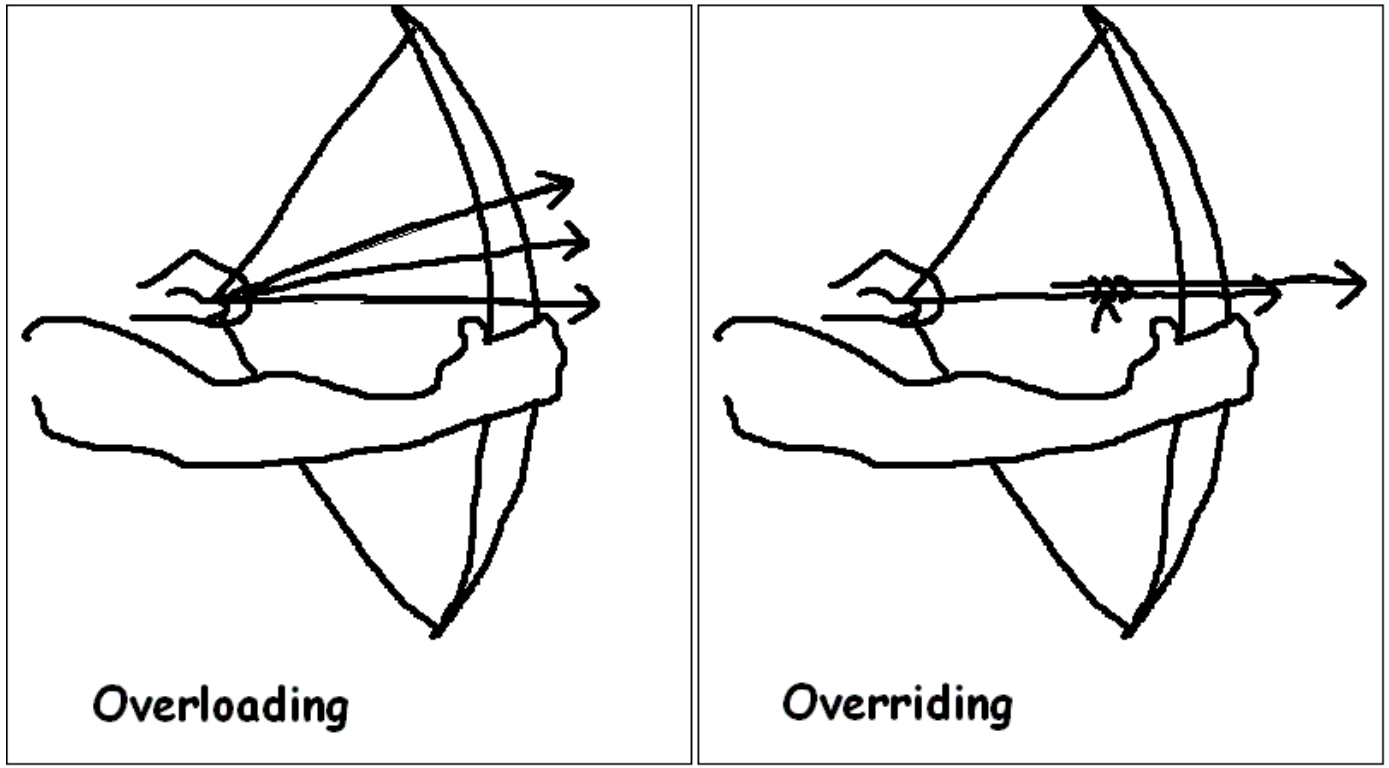

```
public class Medicine {

    private String item; // Thành Phần
    private String mark; // Mục đích

    // create getter setter
    // Nhập số lượng thuốc
    private int inputQuantity(){
        @SuppressWarnings("resource")
        Scanner ip = new Scanner(System.in);
        return Integer.parseInt(ip.nextLine());
    }
```

Polymorphism

- Khái niệm
  - Trong từng trường hợp, hoàn cảnh khác nhau thì **đối tượng, phương thức** có hình thái chức năng khác nhau.
  - Đối tượng, phương thức có tính đa hình được xem như là đối tượng, phương thức đặc biệt. Vì có lúc nó là đối tượng này, lúc là đối tượng khác
- Đa hình trong phương thức
- Đa hình trong đối tượng

Polymorphism

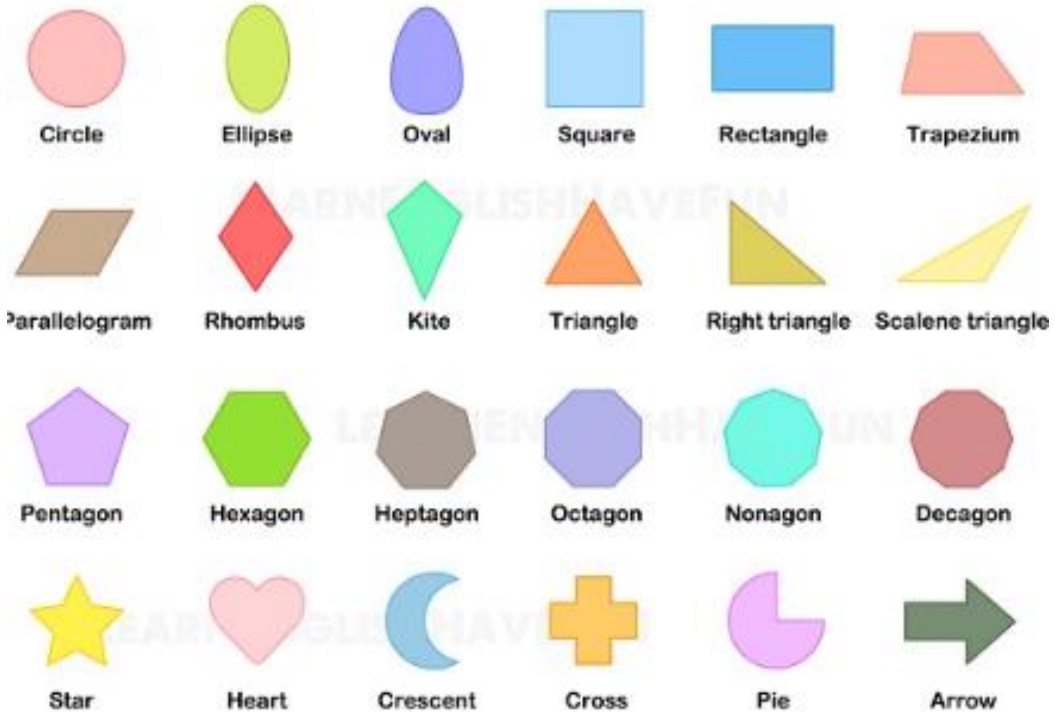
- Đa hình trong phương thức



Polymorphism

- Đa hình trong đối tượng
  - C++: “Con trỏ kiểu lớp cha có thể dùng để trỏ đến đối tượng kiểu lớp con”
  - Như vậy khi khai báo chỉ cần khai báo đối tượng item có kiểu lớp cha, **còn sau đó nó trỏ đến đối tượng phương thức của “ai”, “đứa con nào” thì kệ cha-con nó**.
  - Phương pháp xác định kiểu con vịt. “Nếu animal đi như vịt thì coi là nó là vịt ” ☺ Confuse ☺
  - Một con người thì **có thể là** đàn ông hoặc là phụ nữ

Polymorphism



Polymorphism

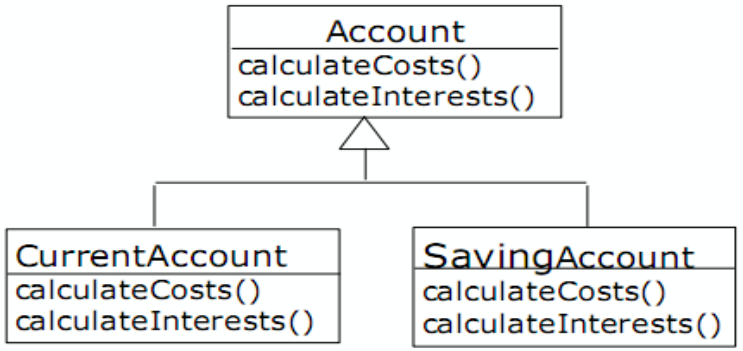
- Đa hình trong đối tượng

```java
public class Shape {
    public void paint(){
        System.out.println("Painting !!");
    }
}
```

```java
public class Circle extends Shape{
    public void paint(){
        System.out.println("Painting circle!!");
    }
}
```

```java
public static void main(String[] args) {
    Shape objSQ = new Square();
    Shape objCC = new Circle();
    // What is result !!! Try it more time
    objSQ.paint();
    objCC.paint();
}
```

```java
public class Square extends Shape{
    public void paint(){
        System.out.println("Painting square!!");
    }
}
```



Polymorphism

- Đặt câu hỏi
  - Tại sao phải dùng đa hình.
  - Tại sao không khai báo trực tiếp KDL như tham chiếu.

Inheritance

❖ Khái niệm kế thừa

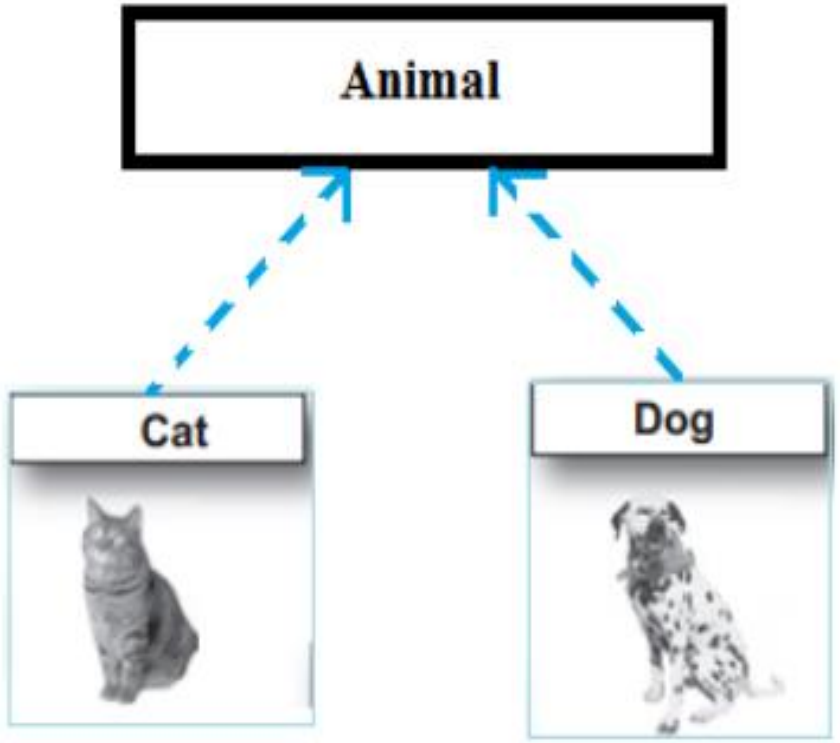
- Là một tính chất quan trọng trong OOP. Nó là một cơ chế trong Java cho phép 1 class được phép kế thừa những **đặc điểm** (fields, methods) của 1 class khác.
- Super class: Chứa đặc điểm được thừa kế (base, parent class).
- Sub class: Kế thừa từ super class, có thể thêm các đặc điểm khác ngoài các đặc điểm từ super class (derived class, extended class, child class)

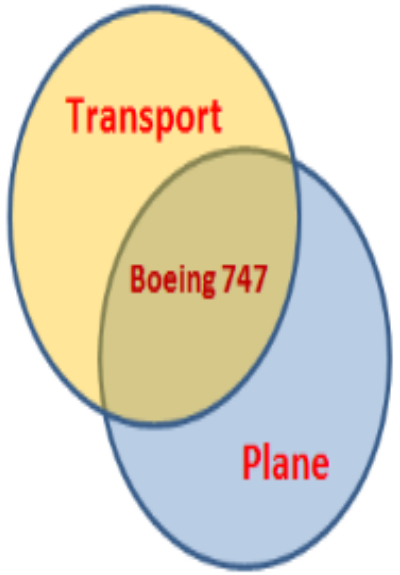
Inheritance

❖ Lợi ích của việc kế thừa

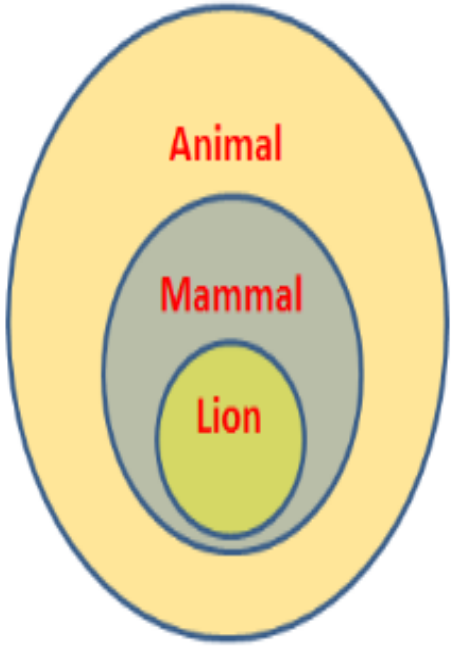
- Tăng khả năng reuse code
- Sub class có thể override lại các đặc điểm (phương thức) của lớp cha, phụ thuộc vào access modifier (**permissons**) trong lớp cha
- Sub class có thể định nghĩa thêm thuộc tính và phương thức mới của riêng phù hợp với mục đích của nó.
- **Không thể override lại các thuộc tính**
- Có 3 dạng kế thừa chính, super class là: **Class, Interface, Abstract Class**
- **Tổng quát hóa – chuyên biệt hóa**

Inheritance





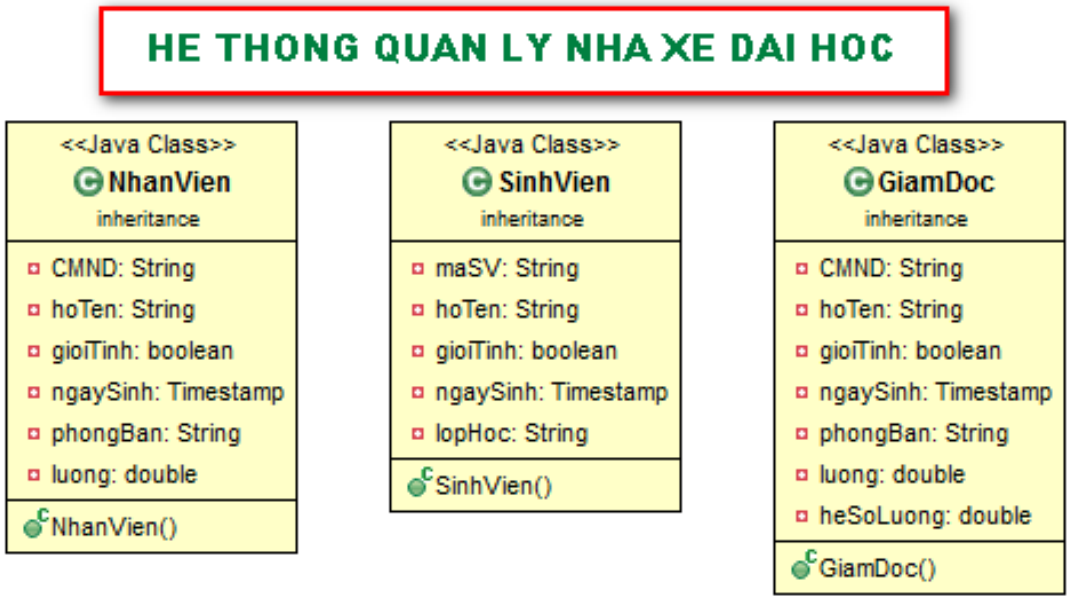
Multiple Inheritance

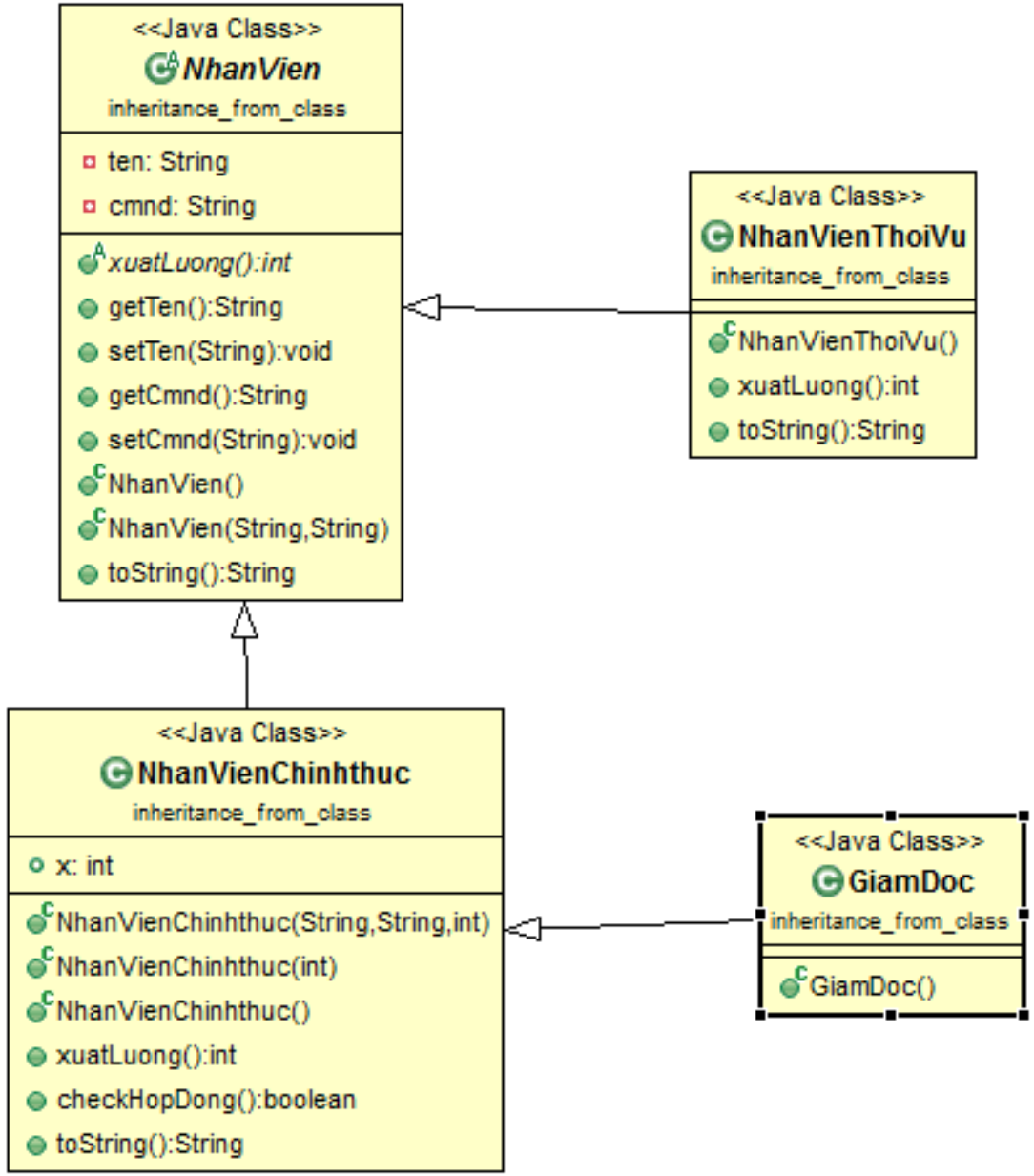


Inheritance (Lion extends Mammal)
(Mammal extends Animal)

Inheritance

http://www.objectaid.com/





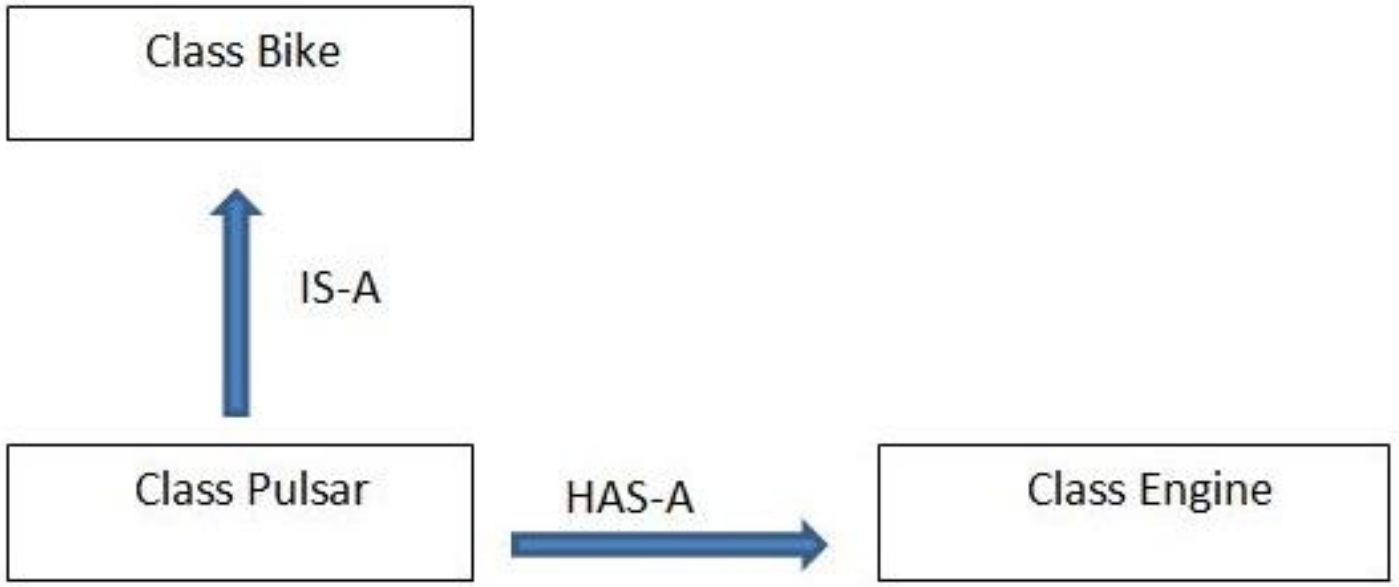
Inheritance

❖ Kết luận

- Tại sao phải dùng thừa kế.
- Thừa kế tất cả các thuộc tính, **phương thức**.
- **No variable overriding, only method overriding**

Inheritance

- Has-A and Is-A relationship



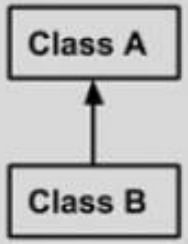
Inheritance

❖ Vấn đề

- Đối với dự án có khoảng 100, 1000 objects
- Mỗi đối tượng có rất nhiều thuộc tính, điều đó thực sự khó khăn để tạo ra 1 super class để các lớp con cùng thừa kế, tăng độ phức tạp cho code
- Thông thường một đối tượng sẽ chứa các thông tin (thuộc tính) riêng của nó, hoạt động độc lập và không quan tâm đến các đối tượng khác đang chứa thông tin gì.
- Convention. Trên 2 - 3 thuộc tính chung mới tạo super class (optional)
- Liên kết chặt
- Mỗi đối tượng chỉ nên chứa các thuộc tính, getter setter, constructor, hashcode, equals methods. Các chức năng thực hiện liên quan đến đối tượng sẽ được xử lý trong services, utils, controller, dao, manager (convention code)
- Chỉ tạo các lớp cha có chung thuộc tính trong phạm vi 1 vài đối tượng. Không nên tạo lớp cha cho toàn bộ project.

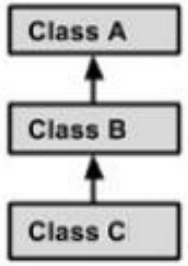
Inheritance

Single Inheritance



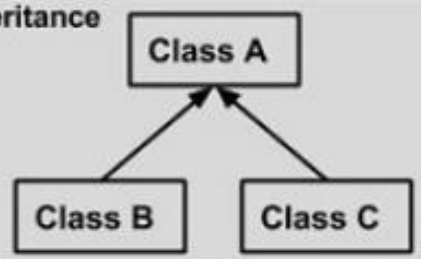

```
public class A {
   .......
}
public class B extends A {
   .........
}
```

Multi Level Inheritance



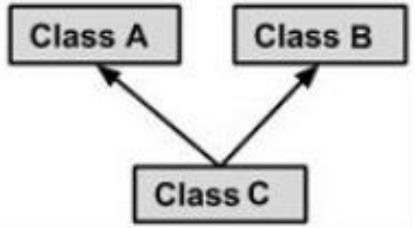

```
public class A { ..................}
public class B extends A {.....................}
public class C extends  B {..................... }
```

Hierarchical Inheritance




```
public class A { ..................}
public class B extends A {.....................}
public class C extends A {..................... }
```

Multiple Inheritance




```
public class A { ..................}
public class B {.....................}
public class C extends  A,B {
   ....................
} // Java does not support mutiple Inheritance
```

Super class

- Interface
- Abstract class
- Class
- What is the difference between them ?

Interface

- Là một reference type trong Java. Tương tự với Class
- Là tập hợp các **abstract methods**[method with no body]
- Class inherit from Interface thực chất kế thừa các abstract method từ interface
- Tạo interface tương tự với việc tạo class. Nhưng class mô tả attributes và methods của object. Interface chỉ chứa những behaviors (abstract methods) mà class kế thừa sẽ định nghĩa, thực thi nó.
- Không thể khởi tạo một interface, not contain any constructors
- Thông thường: chỉ dùng interface để chứa các hàm trừu tượng

Interface

Interface

```java
public interface TrendManager {
    /**...*/
    List<ItemTrendDTO> getWarehouseTrends(Long whId, ControlListTypeEnum type, Long languageId);

    /**...*/
    List<ItemTrendDTO> getWarehouseTrendsPV(Long whId, Long advDateId, Long languageId);

    /**...*/
    Map<Long, List<ItemTrendDTO>> getStoreTrends(final Long countryId, Long whId, ControlListTypeEnum type, Long languageId, List<Long> st

    /**...*/
    Map<Long, List<ItemTrendDTO>> getStoreTrendsPV(final Long countryId, Long whId, Long advDateId, Long languageId, List<Long> storeIds);

    /**...*/
    void saveWarehouseTrends(List<TrendValueDTO> trendValues);

    /**...*/
    void saveStoreTrends(List<TrendValueDTO> trendValues);

    /**...*/
    void cleanUpTrendData(Set<Long> whIds, Long advDateId, LocalDate from, LocalDate until);

    /**...*/
    void cleanUpPVItemTrend(Long removedPVItem);

    /**...*/
    void cleanUpTrendData(Long whId);
}
```

Interface

```java
public class TrendManagerImpl implements TrendManager {

    private static final int NUMBER_OF_DAYS_IN_PAST = 3;
    private static final int NUMBER_OF_DAYS_IN_FUTURE = 4;

    private final TrendDao trendDao;
    private final TimeProvider timeProvider;
    private final EnumMap<ControlListTypeEnum, ItemTrendProvider> itemTrendProviders = new EnumMap<>(ControlListTypeEnum.class);

    @Inject
    public TrendManagerImpl(final TrendDao trendDao, final TimeProvider timeProvider) {...}

    /**
     * {@inheritDoc}
     *
     * @param whId       Id of warehouse
     * @param type       Control list type of items
     * @param languageId Id of language
     * @return
     */
    @Override
    public List<ItemTrendDTO> getWarehouseTrends(final Long whId, final ControlListTypeEnum type, final Long languageId) {
        final LocalDate from = timeProvider.today().minusDays(NUMBER_OF_DAYS_IN_PAST);
        final LocalDate until = timeProvider.today().plusDays(NUMBER_OF_DAYS_IN_FUTURE);

        final ItemTrendProvider itemTrendProvider = itemTrendProviders.get(type);

        if (Objects.nonNull(itemTrendProvider)) {
            return itemTrendProvider.getWarehouseTrends(whId, languageId, from, until);
        }

        return Collections.emptyList();
    }
```

Interface

- Advanced
  - Along with abstract methods, an interface may also contain constants, **default methods**, **static methods**. Method bodies exist only for default methods and static methods.
- https://www.tutorialspoint.com/java/java_interfaces.htm
- Từ Java8. Một interface có thể chứa các phương thức có thân hàm tương tự như class, abstract class

```
public interface Interface01 {
    public void method01();

    default void log(String str) {
        System.out.println("Interface 01: ===> " + str);
    }

    static void display() {
        System.out.println("Interface 01: ===> display");
    }

}
```

Abstract class

- Một class với từ khóa abstract được xem như là abstract class. Abstract class có thể chứa abstract (method with no body, declaration only, no definition) hoặc non-abstract method.
- Abstraction: Là quá trình che dấu thông tin chi tiết và chỉ thể hiện những chức năng cần thiết đến người dùng
- “shows only essential things to the user and hides the internal details” e.g sending message
- There are two ways to achieve abstraction in java
  - Abstract class (0 to 100%)
  - Interface (100%)
- Có thể khởi tạo một abstract class
- Có thể chứa constructor và static methods

Abstract class

Abstract class

```java
public abstract class TrendPropagationRow extends WebMarkupContainer {
    private static final long serialVersionUID = -6183396948088963622L;

    private final List<TextField<String>> takeOverableTrends;

    protected TrendPropagationRow(final String id) {
        super(id);

        takeOverableTrends = new ArrayList<>();

        // sysdate and 4 days in the future are able to take over
        for (int i = 0; i < 5; i++) {
            final TextField<String> field = new TextField<>("bulk-edit-" + i, Model.of());
            field.setType(String.class);
            add(field);

            takeOverableTrends.add(field);
        }

        add(new WfmAjaxSubmitLinkButton<Void, Void>("propagate-btn") {
            private static final long serialVersionUID = 42608103131413970l0L;

            @Override
            public void onSubmit(final AjaxRequestTarget target, final Form<?> form) { onTakeOver(target); }

            @Override
            protected void onError(final AjaxRequestTarget target, final Form<?> form) { onTakeOver(target); }
        });
    }

    private void onTakeOver(final AjaxRequestTarget target) {
        final List<String> trends = takeOverableTrends.stream()
                .map(FormComponent::getConvertedInput)
                .collect(Collectors.toList());
        if (hasAnyChanges(trends)) {
            onSubmit(trends, target);
        }
    }

    protected abstract void onSubmit(final List<String> trends, final AjaxRequestTarget target);

    private boolean hasAnyChanges(final List<String> trends) {
```

Abstract class

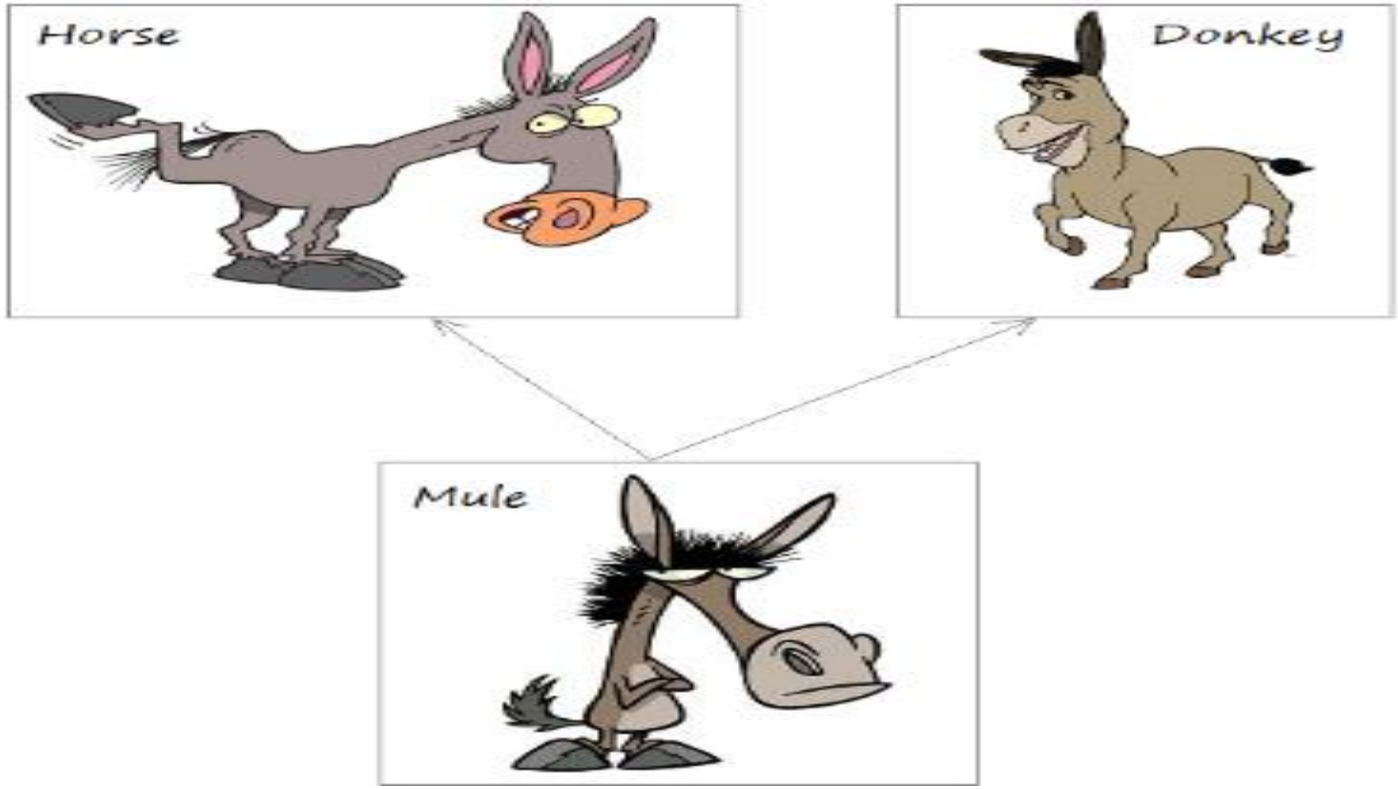
```
private TrendPropagationRow createPropagationRow() {
    final TrendPropagationRow propagationRow = new TrendPropagationRow("propagation-row") {
        private static final long serialVersionUID = 6614775284015165 93L;

        @Override
        protected void onSubmit(final List<String> trends, final AjaxRequestTarget target) {
            for (int i = 0; i < trends.size(); i++) {
                final String trendString = trends.get(i);
                if (StringUtils.isNotEmpty(trendString)) {
                    if (trendString.matches(NUMBER_PATTERN)) {
                        final Integer trend = Integer.parseInt(trendString);
                        final Iterator<? extends ItemTrend> iter = dataProvider.iterator(0, dataProvider.size());
                        while (iter.hasNext()) {
                            final ItemTrend itemTrend = iter.next();
                            if (itemTrend.isSelected()) {
                                final TrendValue trendValue = itemTrend.getTrendValues().get(i + 3);
                                if (trendValue.isEditable()) {
                                    trendValue.setEditValue(trend);
                                    onTakeOver(trendValue);
                                }
                            }
                        }
                    }
                }
            }
            clearInputs();
            target.add(selectAll);
            target.add(dataBox);
        }
```

Phân biệt Interface, Abstract class, Class trong tính chất kế thừa ???

Inheritance

Trong Java có hỗ trợ tính chất đa thừa kế không ?

Cho ví dụ demo



Multiple - Inheritance

La là con của **lừa đực và ngựa cái**. La thì kiên nhẫn,có chân cứng cáp và sống lâu hơn ngựa và chúng thường được xác định là ít cứng đầu, nhanh hơn và thông minh hơn lừa. Được đánh giá cao về độ to lớn, la thường nặng **khoảng giữa 370kg và 460 kg**.

Functional Interface

- Là Interface chỉ có duy nhất một abstract method
- Xuất hiện từ 03/2014 trong JAVA 8
- Annotation: @FunctionalInterface
- Có thể sử dụng thay thế với Lambda expression để code ngắn gọn hơn

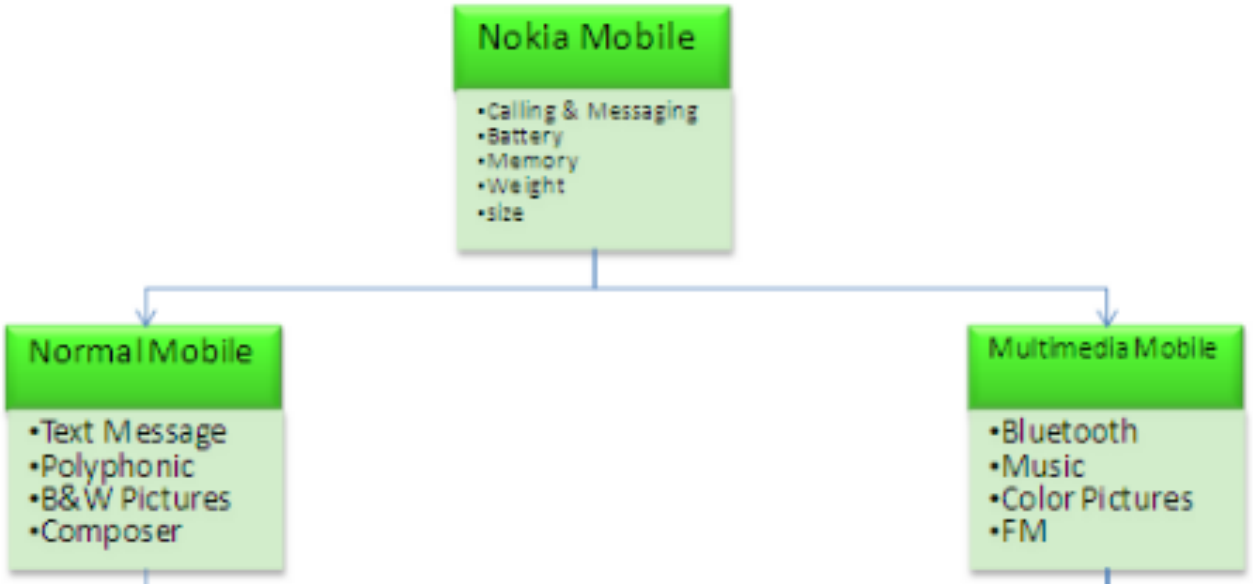
Abstraction

❖ Đối tượng trong thực tế rất phức tạp

- Cần đơn giản hóa, bỏ những chi tiết không cần thiết
- Chỉ "trích xuất" lấy những thông tin liên quan, quan trọng đối với bài toán.
- Những thông tin đó phải "**nhìn thấy được**"

Abstraction

- Lớp trừu tượng là lớp có chứa các hàm trừu tượng : biết tên hàm chưa có định nghĩa
- Trừu tượng hoá các đối tượng Nokia



Bài tập

- Viết chương trình thực hiện đa năng hóa toán tử
  - int operate(int a, int b): one abstract method only
  - Hàm có thể thực hiện các chức năng
    - +
    - -
    - *
    - /

Interview questions

- Tính chất lập trình hướng đối tượng
  - Nêu cách mình hiểu về mỗi tính chất trong OOP
  - Tính chất nào là quan trọng nhất
- Phân biệt Class và Object
- Tại sao phải dùng Interface và Abstract Class
  - Phân biệt interface || abstract || class
  - Interface method and static methods
- Phân biệt overload override
- Phân biệt static và final
- Phân biệt this super
- Đa kế thừa

Access Modifier

Access Modifier	Within class	Within package	Outside package by subclass only	Outside package
Public	YES	YES	YES	YES
Private	YES	NO	NO	NO
Protected	YES	YES	YES	NO
Default	YES	YES	NO	NO

BÀI TẬP

Bài 2: Viết chương trình quản lý nhân sự của một công ty gồm các loại nhân sự sau đây.

a. Giám đốc gồm các thuộc tính: họ tên, năm sinh, hệ số lương, hệ số chức vụ.

b. Trường phòng gồm các thuộc tính: họ tên, năm sinh, hệ số lương, sô lượng nhân viên quản lý

c. Nhân viên gồm các thuộc tính: họ tên, ngày sinh, hệ số lương, tên đơn vị(phòng/ban)

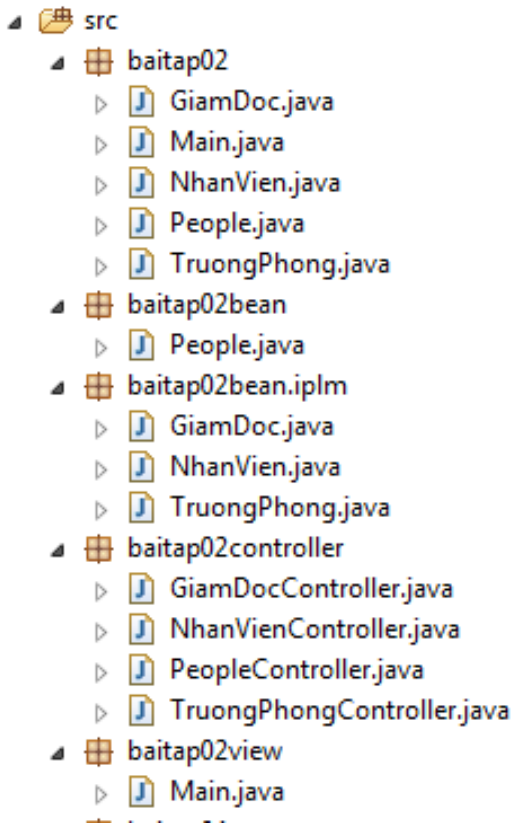
Chương trình thực hiện các công việc sau đây.

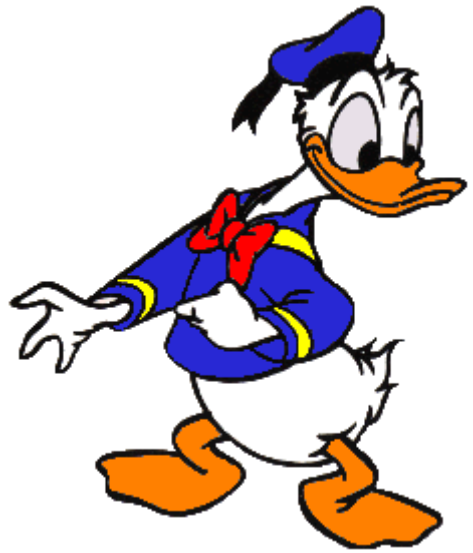
+ Nhập dữ liệu gồm 1 Giám đốc, 1 Trưởng phòng, 2 Nhân viên
+ Hiển thị thông tin các nhân sự có trong công ty
+ Tính và in ra mức lương của từng loại nhân sự , biết rằng :

Nhân viên:	Lương = (hệ số lương + hệ số chức vụ(nếu có))	* 1250000
Trưởng phòng:	Lương = (hệ số lương + hệ số chức vụ(nếu có))	* 2200000
Giám đốc:	Lương = (hệ số lương + hệ số chức vụ(nếu có))	* 3000000

Gợi ý: Tạo class People gồm các thuộc tính: hoTen, namSinh, heSoLuong, heSoChucVu
phương thức: tinhLuong()

Tạo lần lượt các class GiamDoc, TruongPhong, NhanVien kế thừa từ class people và định nghĩa lại hoặc thêm một số thuộc tính, phương thức trong lớp con. Dữ liệu nhập từ bàn phím.





END