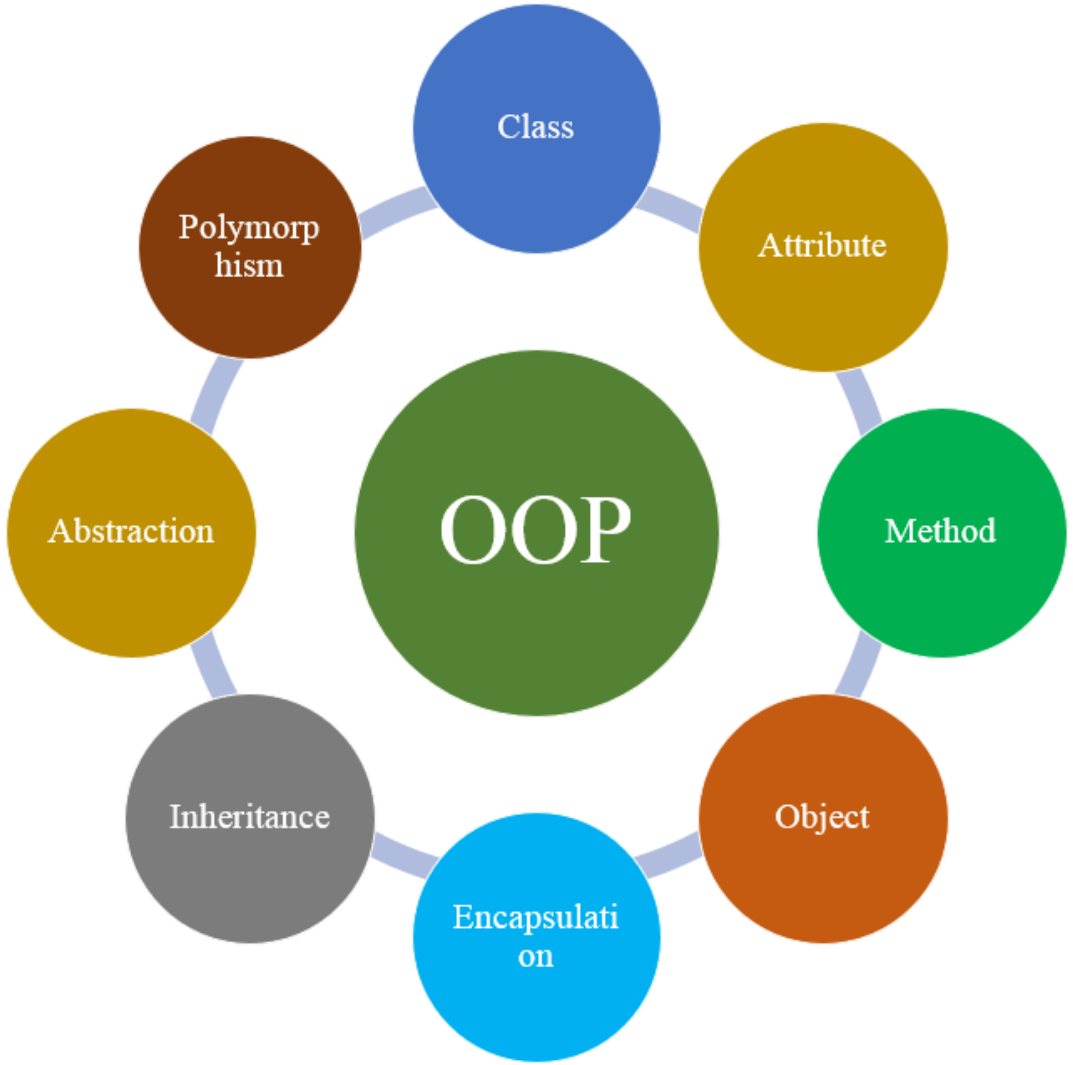
Bài 03

Các khái niệm trong lập trình hướng đối tượng

Nội dung bài học

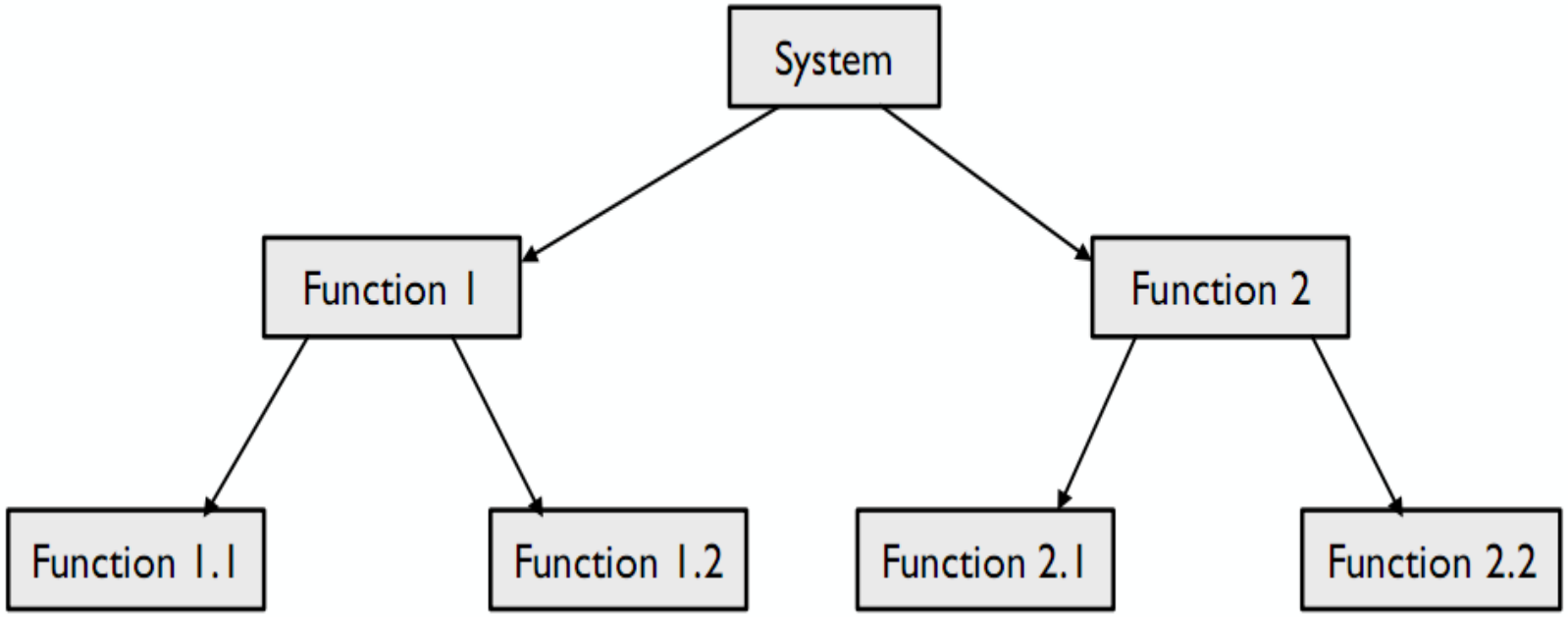


Nội dung bài học

- Khái niệm lập trình hướng thủ tục
- Khái niệm lập trình hướng đối tượng
- Lập trình hướng đối tượng trong JAVA
  1. Class Object
  2. Attribute Method Static
  3. Access modifier
  4. Getter Setter Constructor Destructor
  5. Interface Abstract Class
  6. Overload Override

Lập trình hướng cấu trúc

- Một hệ thống bao gồm một vài HÀM cụ thể
- Một hệ thống gồm các hệ thống con nhỏ hơn



Các hàm giao tiếp với nhau để chia sẽ dữ liệu và chuyển tiếp các tham số **"GHÉP NỐI LỎNG – LIÊN KẾT CHẶT"**

Lập trình hướng đối tượng

- Để khắc phục những hạn chế của FOP
- Giải pháp của vấn đề là tổ chức hệ thống xung quanh các đối tượng
- Một hệ thống bao gồm các **đối tượng** và **mối quan hệ** giữa chúng
- Các đối tượng giao tiếp với nhau bởi các **thông điệp, tin nhắn**
- **Không có biến toàn cục**

FOP II POP

Functional approach v.s. object-oriented approach

- Functional approach
  - System = algorithms + data structures
- Object-oriented approaches
  - System = Σ objects
  - Object = algorithms + data structures

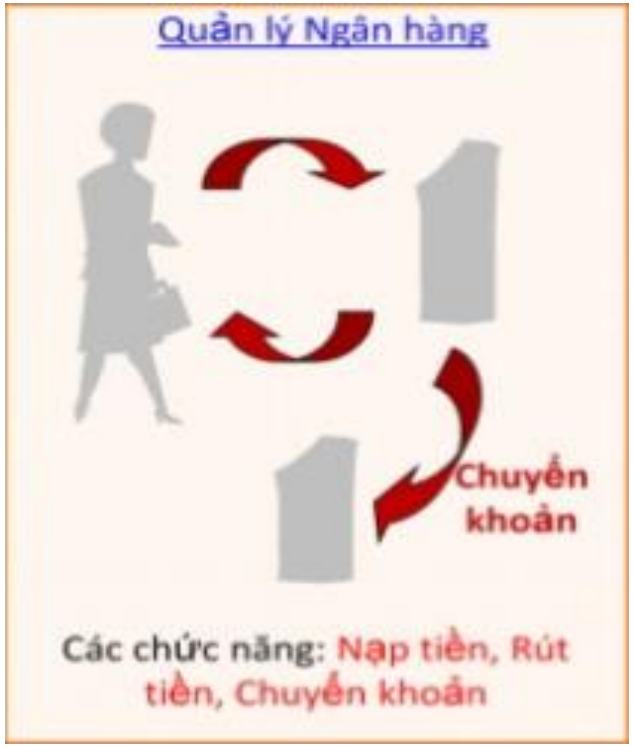
BÀI TOÁN VÍ DỤ

Bài toán quản lý giao dịch giữa **Khách Hàng** với **Ngân Hàng**

- **Khách hàng (Thông tin khách hàng = Data)** có các giao dịch với ngân hàng
  - **Nạp tiền** vào **tài khoản**
  - **Rút tiền** từ tài khoản
  - **Chuyển khoản** giữa các tài khoản
- Giải quyết bài toán theo 2 phương pháp
  - Hướng thủ tục – **P**rocedural **P**rograming
  - Hướng đối tượng – **O**bject **O**riented **P**rograming

Lập trình hướng cấu trúc

- Quan tâm đến **chức năng** (cấu trúc, thủ tục) cần thực hiện
- Chương trình lớn được **chia thành các chức năng** (thủ tục) nhỏ hơn sao cho "**ghép nối lỏng và liên kết chặt**"
- Phần lớn các chức năng **sử dụng chung dữ liệu**



Lập trình hướng cấu trúc

Vấn đề gặp phải của lập trình hướng thủ tục với các hệ thống phức tạp

- Vấn đề quản lý dữ liệu phức tạp
- Các chức năng liên kết chặt chẽ với nhau. Khó thay đổi chức năng
- Vấn để mở rộng chức năng hay sử dụng lại module đã viết
- Vấn đề quản lý quá nhiều chức năng

Cần module hóa các chức năng.

Mỗi chức năng nên tập trung giải quyết vấn đề chính

Phân cấp chức năng ghép nối càng lỏng càng tốt

Phương pháp lập trình hướng đối tượng

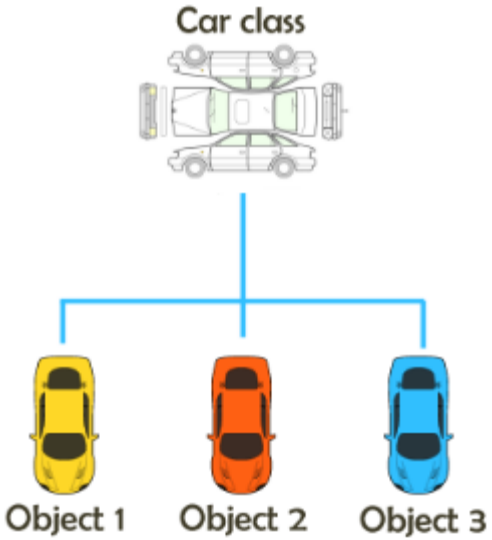
Lập trình hướng đối tượng

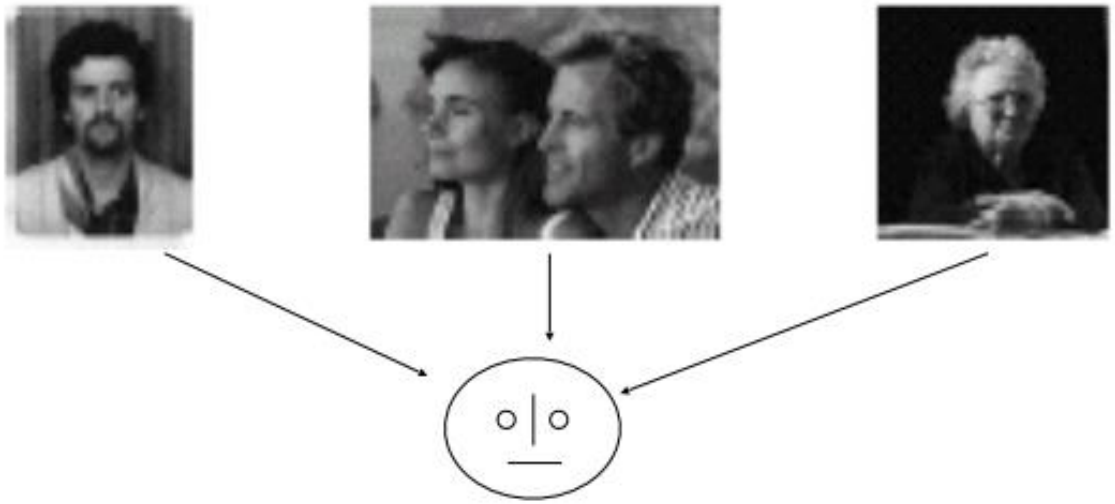
- Chương trình được chia thành các **đối tượng(thực thể)** tồn tại trong thế giới thực
- Mỗi đối tượng chịu trách nhiệm quản lý riêng dữ liệu và chức năng của nó
- Các đối tượng tương tác với nhau thông qua các **thông điệp** (chức năng)

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

CLASS

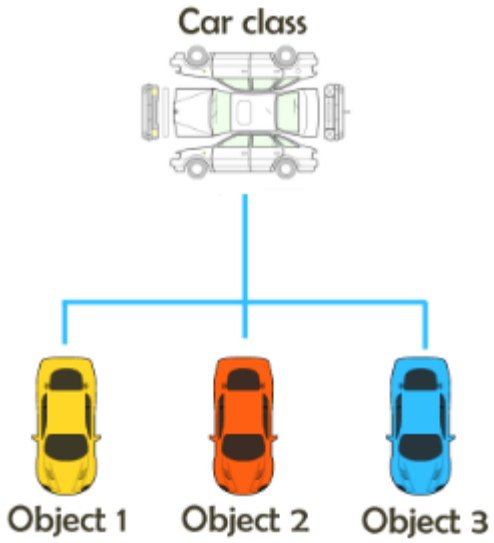
- Class là một khái niệm **trừu tượng** để chỉ **tập hợp các object** có chung:
  - Tính chất: **thuộc tính** -- Hành vi: **phương thức**
  - Mối quan hệ với các class khác

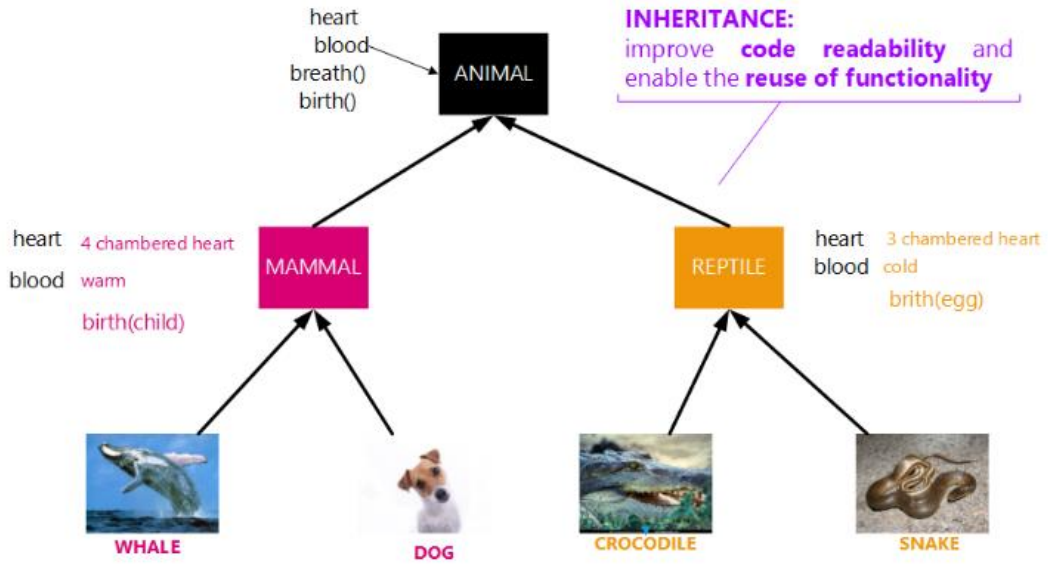




OBJECT

- Object là một khái niệm dùng để mô tả các thực thể **tồn tại trong thế giới thực**.
  - VD: Student "Le Na" là một object
  - Student không phải là một object





OBJECT

- An object is an instance of a class
- An value is an instance of an attribute
- A link between objects is an instance of the relationship between classes
- Too complicated
- **Ex:** Dog bites cat

PHÂN BIỆT CLASS VÀ OBJECT ?

CLASS OBJECT

```
public class Rectangle {
    // attributes
    private int width;
    private int height;

    // default: empty constructor
    public Rectangle() {
    }

    // constructor with 2 parameters
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    // setter
    public void setWidth(int width) {
        this.width = width;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    // getter
    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }
```

Khai báo thuộc tính của Class

- Cú pháp:

```
// attributes
private int width;
private int height;
```

[AccessModifier] [static] [final] DataType attributeName;

Khai báo phương thức của Class

- Cú pháp:

```
// setter
public void setWidth(int width) {
    this.width = width;
}

public void setHeight(int height) {
    this.height = height;
}

// getter
public int getWidth() {
    return width;
}

public int getHeight() {
    return height;
}

@Override
public String toString() {
    return "w" + this.width + " h" + this.height;
}
```

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
	YES	YES	NO	NO

Getter Setter

```
public  class HinhChuNhat{

    // Inform attributes
    public int chieuDai;
    private int chieuRong;
    public int chieuMua;
    protected String chieuNhoEm;
    // Getter
    public int getChieuDai() {
        return chieuDai;
    }
    // Setter
    public void setChieuDai(int chieuDai) {
        this.chieuDai = chieuDai;
    }
```

Getter setter dùng để làm gì ???

Biến this: truy cập đến bản thân đối tượng hiện tại. Phân biệt this và super ???

- Phương thức setter:
  - Truy cập vào đối tượng và gán giá trị cho thuộc tính của đối tượng
- Phương thức getter:
  - Truy cập vào đối tượng và lấy giá trị của đối tượng

Getter Setter vs Access modifier

```
public class Car {
    private String color;
    private String model;

    public String getColor() {
        return color;
    }

    public String getModel() {
        return model;
    }

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.getColor());
```

```
public class Car {
    public String color;
    public String model;

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.color);
```

Tại sao chúng ta không khai báo access modifier là public để lấy dữ liệu từ external class một cách đơn giản – code ngắn

Mà phải sử dụng getter setter phiền phức ?

Constructor

```
// Constructor
public HinhChuNhat(){
    super();
}
// Constructor load
public HinhChuNhat(int chieuDai, int chieuRong){
    this.chieuDai = chieuDai;
    this.chieuRong = chieuRong;
}
```

Auto load

Có thể dùng thay cho setter.
Nên dùng cái nào constructor || getter-setter.
Tại sao ?

- Phương thức khởi tạo - constructor:
  - Là một phương thức đặc biệt của Class, được gọi tự động khi tạo một thể hiện của lớp
  - Cùng tên với Class và không có giá trị trả về
  - Được gọi đến khi dùng từ khóa new
  - Một class có mặc định một constructor không có tham số. Có thể định nghĩa nhiều constructor cho lớp

Destructor

- Tại sao lại cần phần sử dụng hàm hủy ?
  - Trong những dự án với các chương trình lớn đòi hỏi phải **kiểm soát được các luồng dữ liệu tạm thời trên RAM** , giúp **giảm nguy cơ mất dữ liệu** khi đang thao tác hoặc làm **tăng hiệu năng giúp chương trình của ta chạy nhanh hơn mượt hơn**.
  - Để kiểm soát được các ô nhớ tạm trên RAM chúng ta cần tìm hiểu cơ chế **alias** thu gom rác **garbage collection** trong JAVA
- Bài toán ví dụ

```
class KhachHang{
    private String name;
    public void setName(String name)
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
}
```

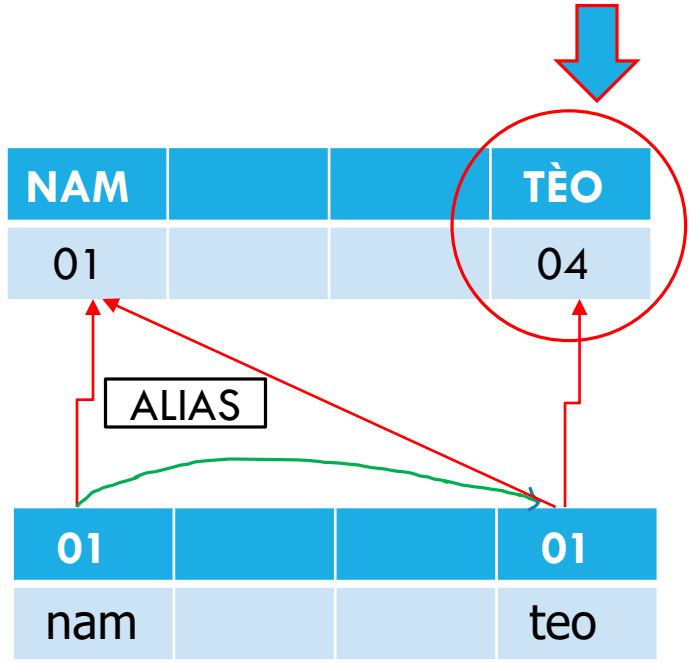
```
public class KhachHangDemo {
    public static void main(String[] args) {
        KhachHang nam = new KhachHang();
        nam.setName("Nam");

        KhachHang teo = new KhachHang();
        teo.setName("Tèo");

        System.out.println("1=>"+nam.getName());
        System.out.println("2=>"+teo.getName());

        teo = nam ;
        System.out.println("3=>"+teo.getName());

        teo.setName("Lạc trôi");
        System.out.println("4=>"+teo.getName());
        System.out.println("5=>"+nam.getName());
    }
}
```



- Trong Java ko có khái niệm hàm hủy

Hashcode/Equals

❖ Equals:

- So sánh 2 đối tượng
- Dựa trên giá trị của các thuộc tính trong mỗi đối tượng
- 2 đối tượng bằng nhau nếu được lưu trữ trong cùng địa chỉ vùng nhớ hoặc cùng value (Same memory address)

❖ Hashcode

- Trả về 1 số nguyên integer thể hiện địa chỉ vùng nhớ của đối tượng
- Phương thức mặc định trả về integers là khác nhau cho mỗi instance
- Được sử dụng chủ yếu trong collection với Hashtable, HashMap, HashSet
- Để loại bỏ những phần tử trùng nhau trong một danh sách, tập hợp.
- Có thể override để cụ thể hóa chức năng.

Question

```
public class Car {
    private String color;
    private String model;

    public String getColor() {
        return color;
    }

    public String getModel() {
        return model;
    }

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.getColor());
```

```
public class Car {
    public String color;
    public String model;

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.color);
```

Khi khai báo các thuộc tính, phương thức là non-static.

Thuộc tính, phương thức thuộc phạm vi của class hay đối tượng.

Question

```
public class Car {
    private String color;
    private String model;

    public String getColor() {
        return color;
    }

    public String getModel() {
        return model;
    }

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.getColor());
```

```
public class Car {
    public String color;
    public String model;

Car object1 = new Car("yellow", "Audi");
Libs.printf("object 1 color", object1.color);
```

Tạo 100 mẫu xe Audi với các màu sắc khác nhau

Trường hợp 1 ngày nào đó. Chủ sở hữu đổi tên hãng xe. Không phải là Audi mà sang tên khác

Giải quyết vấn đề ?

Khái niệm static

- The static can be:
  - Variable
  - Method
  - Block
  - Nested class
- Tại sao phải có static
  - Quản lý bộ nhớ
  - Static variable được lưu trữ tại một vùng nhớ cố định trên RAM
  - Thuộc phạm vi của class.
  - *"The static keyword belongs to the class than instance of the class"*

Khái niệm static

❖ Static variable

- Chỉ phụ thuộc vào lớp không phụ thuộc đối tượng
- **Chỉ khởi tạo một lần khi chương trình bắt đầu thực thi**
- Truy cập trực tiếp bằng tên class mà không cần bất kì đối tượng nào

  Syntax: ClassName.staticVariable

  Ex: public static int width; // Class Box

  Box.width = 5; // setter method

- Có thể truy cập thuộc tính phương thức static thông qua đối tượng của class. Nhưng vì static thuộc phạm vi class. Nên nếu một đối tượng thay đổi giá trị static thì tất cả các đối tượng còn lại cũng sẽ mang giá trị của thuộc tính static đó.
- Nên dùng class truy cập thuộc tính, phương thức static

Khái niệm static

❖ Static method

```
private static String hoTen;
private int namSinh;

public void thongTin(){
    int soTuoi = Calendar.getInstance().get(Calendar.YEAR) - namSinh;
    System.out.println("Hoten: " + hoTen);
    System.out.println("NamSinh: " + namSinh);
    System.out.println("DoTuoi: " + soTuoi);
}
```

```
private static String hoTen;
private int namSinh;

public static void thongTin(){
    int soTuoi = Calendar.getInstance().get(Calendar.YEAR) - namSinh;
    System.out.println("Hoten: " + hoTen);
    System.out.println("NamSinh: " + namSinh);
    System.out.println("DoTuoi: " + soTuoi);
}
```

NONSTATIC IN STATIC => FAIL

STATIC IN NONSTATIC => OK

NONSTATIC IN NONSTATIC => Absolutely !

STATIC IN STATIC => Absolutely !

- Static và NonStatic: Thằng nào được tạo ra trước

Difference Between Static and non-Static Variable in Java

The variable of any class are classified into two types;

- Static or class variable
- Non-static or instance variable

Static variable in Java

Memory for static variable is created only one in the program at the time of loading of class. These variables are preceded by static keyword. tatic variable can access with class reference.

Non-static variable in Java

Memory for non-static variable is created at the time of create an object of class. These variable should not be preceded by any static keyword Example: These variables can access with object reference.

Difference between non-static and static variable

	Non-static variable	Static variable
1	These variable should not be preceded by any static keyword Example: `class A { int a; }`	These variables are preceded by static keyword. **Example** `class A { static int b; }`
2	Memory is allocated for these variable whenever an object is created	Memory is allocated for these variable at the time of loading of the class.
3	Memory is allocated multiple time whenever a new object is created.	Memory is allocated for these variable only once in the program.
4	Non-static variable also known as instance variable while because memory is allocated whenever instance is created.	Memory is allocated at the time of loading of class so that these are also known as class variable.
5	Non-static variable are specific to an object	Static variable are common for every object that means there memory location can be sharable by every object reference or same class.
6	Non-static variable can access with object reference. **Syntax** `obj_ref.variable_name`	Static variable can access with class reference. **Syntax** `class_name.variable_name`

Note: static variable not only can be access with class reference but also some time it can be accessed with object reference.

OOP

- Bài tập
  - Thiết kế bài toán mua hàng khi khách hàng vào mua **điện thoại** tại của hàng thế giới di động. Sau khi hoàn tất mua hàng. Thông tin khách hàng phải được lưu trữ tại hệ thống.

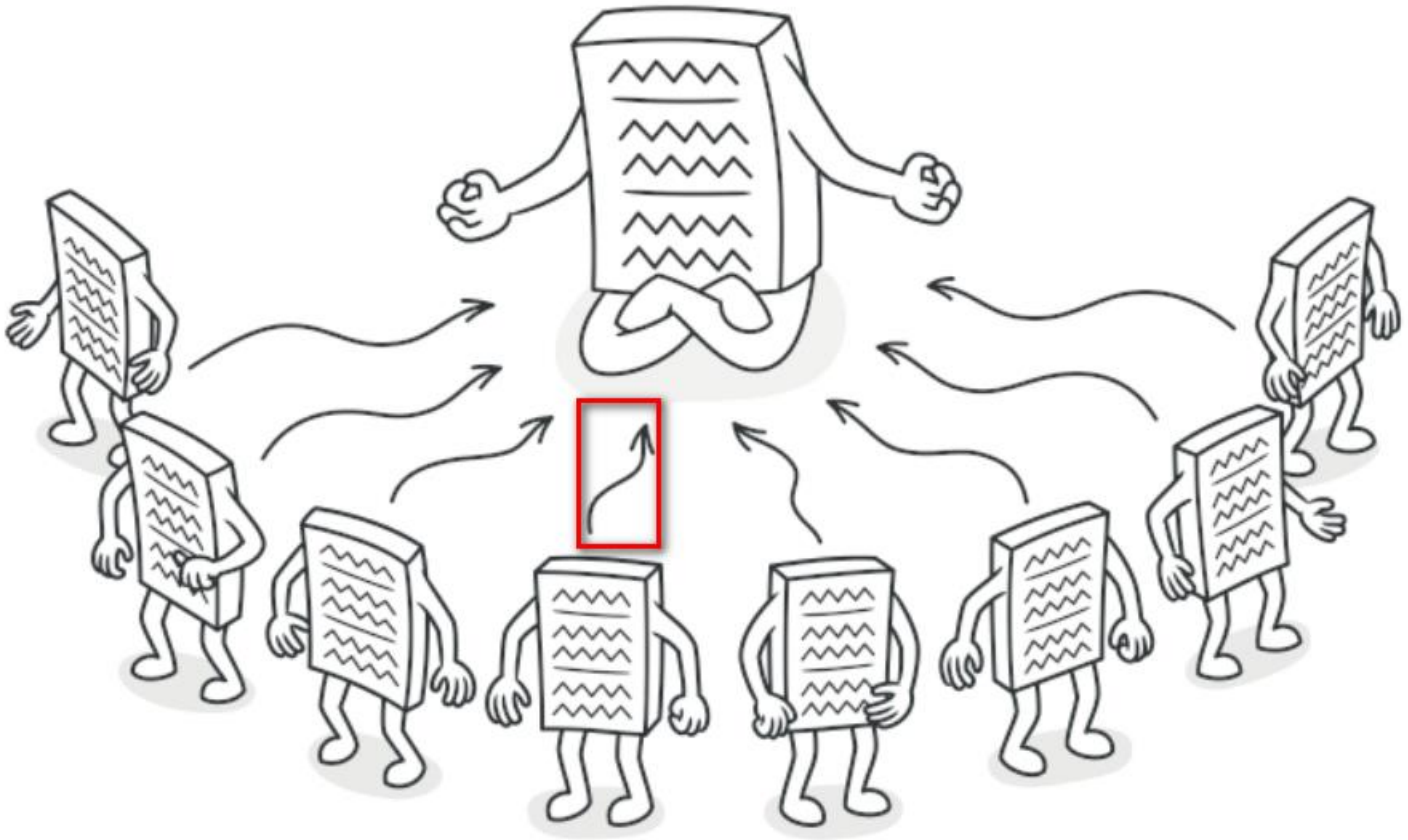
    Trong ngày 05/05. Mặt hàng trên 590.000 sẽ được giảm giá 10%
  - Khách hàng phải biết được thông tin thiết bị mà mình đã mua.
  - Thông tin **khách hàng**: **họ tên**, số điện thoại, chứng minh nhân dân, địa chỉ
  - Thông tin **thiết bị**: mã thiết bị, hệ điều hành, màu sắc, giá.
  - Chương trình:
    - **Khởi tạo dữ liệu cho cửa hàng**
    - **Khởi tạo thông tin khách hàng**
    - Thực hiện mua hàng
      - **Tính tổng tiền mà khách hàng phải trả.**
      - Xuất ra thông tin toàn bộ sản phẩm mà khách hàng đã mua để xuất hóa đơn

Ứng dụng static

- Helper – Utilities class
- Design Pattern: Builder, Singleton
  - Singleton: is a design solution where an application wants to have one and only one instance of any class
  - Builder: is a design pattern that allows for the step-by-step creation of complex objects using the correct sequence of actions

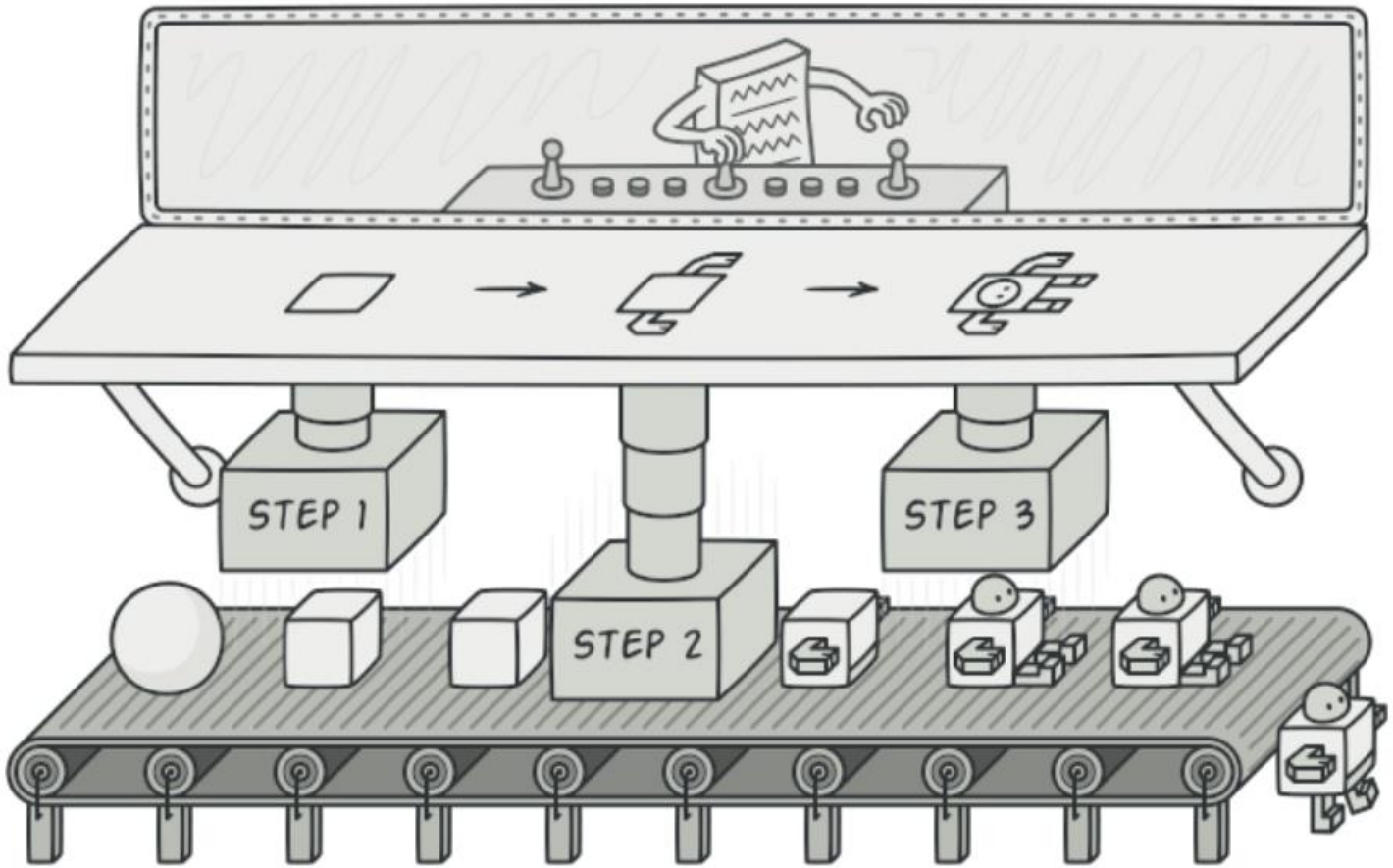
Ứng dụng static

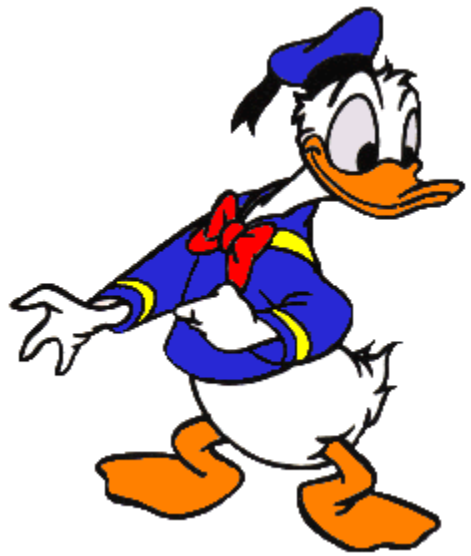
- Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.



Ứng dụng static

- Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.





END