

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

□ □ □ □ □



BÀI TẬP NHÓM MÔN HỌC KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM

MIOStore - Assignment 3

Giảng viên hướng dẫn : Trần Đình Quế

Nhóm môn học : 03

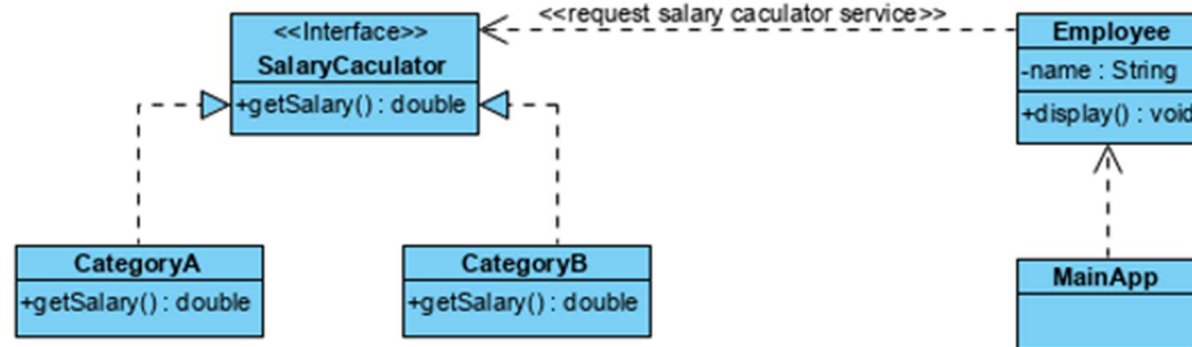
Nhóm bài tập : 08

Danh sách thành viên :

Lê Thị Hoa	B16DCCN151
Phạm Ngọc Hoàng	B16DCCN159
Nguyễn Đình Thắng	B16DCCN319
Phan Quang Thành	B16DCCN331
Trần Nhật Minh	B16DCCN519

1. Sử dụng interface để thiết kế chương trình tính lương nhân viên của một số phòng ban

➤ Thiết kế:



➤ Code:

Tên class	Code
Class SalaryCalculator	<pre>package Question1; public interface SalaryCalculator { double getSalary(); }</pre>

Class CategoryA	<pre>package Question1; public class CategoryA implements SalaryCalculator { double baseSalary; double OT; public CategoryA(double baseSalary, double OT) { this.baseSalary = baseSalary; this.OT = OT; } @Override public double getSalary() { return baseSalary + OT; } }</pre>
Class CategoryB	<pre>package Question1; public class CategoryB implements SalaryCalculator { final static double commission = 0.02; double baseSalary; double salesAmt; public CategoryB(double baseSalary, double salesAmt) { this.baseSalary = baseSalary; this.salesAmt = salesAmt; } @Override public double getSalary() { return baseSalary + commission * salesAmt; } }</pre>

Class Employee	<pre>package Question1; public class Employee { SalaryCalculator empType; String name; public Employee(SalaryCalculator empType, String name) { this.empType = empType; this.name = name; } public void display() { System.out.println("Name: " + name); System.out.println("Salary: " + empType.getSalary()); } }</pre>
Class MainApp	<pre>package Question1; public class MainApp { public static void main(String[] args) { SalaryCalculator salaryCalculator = new CategoryA(baseSalary: 1000, OT: 200); Employee employee = new Employee(salaryCalculator, name: "Thang"); employee.display(); salaryCalculator = new CategoryB(baseSalary: 2000, salesAmt: 800); employee = new Employee(salaryCalculator, name: "Nam"); employee.display(); } }</pre>

Kết quả chạy code

```
Name: Thang
Salary: 1200.0
Name: Nam
Salary: 2016.0

Process finished with exit code 0
```

- **Kinh nghiệm:**

- ❖ Phân tích ví dụ:

- Đầu tiên, chúng ta sẽ thấy là chúng ta cần tính lương cho các nhân viên dù là bất kỳ phòng ban nào -> có phương thức `getSalary()` tính lương là hành động cần thực hiện giống nhau -> tạo một class interface đa năng để tính lương theo phòng ban được.
 - Mỗi Category có một cách tính khác nhau -> tạo các lớp Category implements interface để tính lương theo cách khác nhau
 - Mỗi nhân viên thì sẽ thuộc các phòng ban khác nhau và có kiểu tính lương khác nhau -> tạo Nhân viên với hai thuộc tính tên và cách tính lương

- ❖ Áp dụng sau này:

- Xác định các chức năng chung mà có thể xây dựng interface.
 - Xây dựng các class implements interface đó với các cách thức thực hiện phương thức abstract theo cách khác nhau
 - Khởi tạo các đối tượng chính có thuộc tính là các class trên

- ❖ Đặc điểm của interface:

- Một interface không chứa bất cứ hàm Constructor nào.
 - Tất cả các phương thức của interface đều là trừu tượng.
 - Một interface không thể chứa một trường nào trừ các trường vừa static và

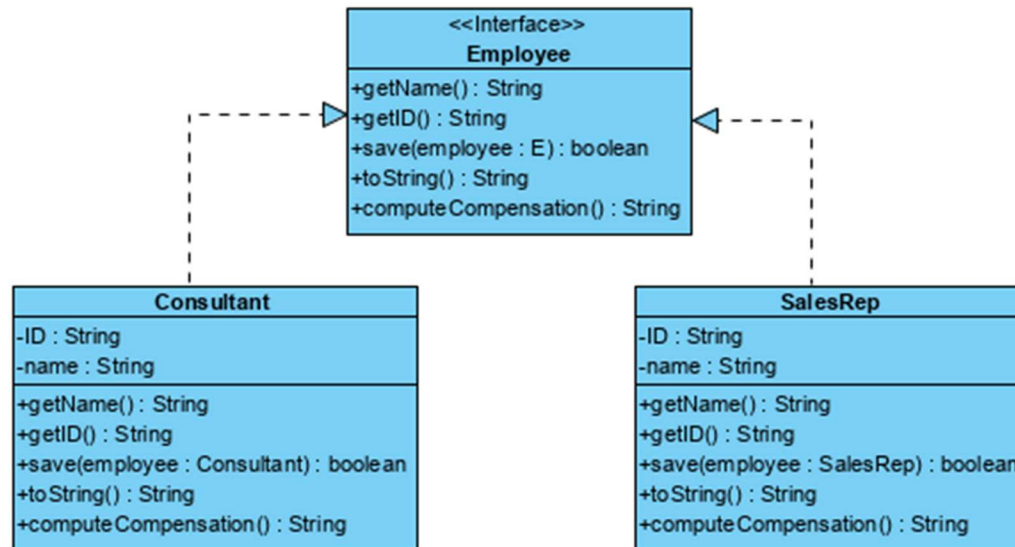
final.

- Một interface không thể kế thừa từ lớp, nó được implements bởi một lớp.
- Một interface có thể kế thừa từ nhiều interface khác.

2. Sử dụng abstract hoặc interface để thiết kế class Employee với các phương thức:

- Lưu dữ liệu employee
- Hiển thị dữ liệu employee
- Truy cập các thuộc tính của employee
- Tính toán

- Mô hình



- Code
 - Interface

Tên class	Code
Class Employee	<pre> package Question2.Interface; public interface Employee { String getName(); String getId(); void save(); String toString(); String computeCompensation(); } </pre>

Class
Consultant

```
package Question2.Interface;

public class Consultant implements Employee {
    private String id;
    private String name;

    public Consultant(String id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String getName() { return this.name; }

    @Override
    public String getId() { return this.id; }

    @Override
    public void save() { System.out.println("Save Consultant done"); }

    @Override
    public String computeCompensation() { return "Consultant"; }
}
```


Class SalesRep

```
package Question2.Interface;

public class SalesRep implements Employee {
    private String id;
    private String name;

    public SalesRep(String id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String getName() { return this.name; }

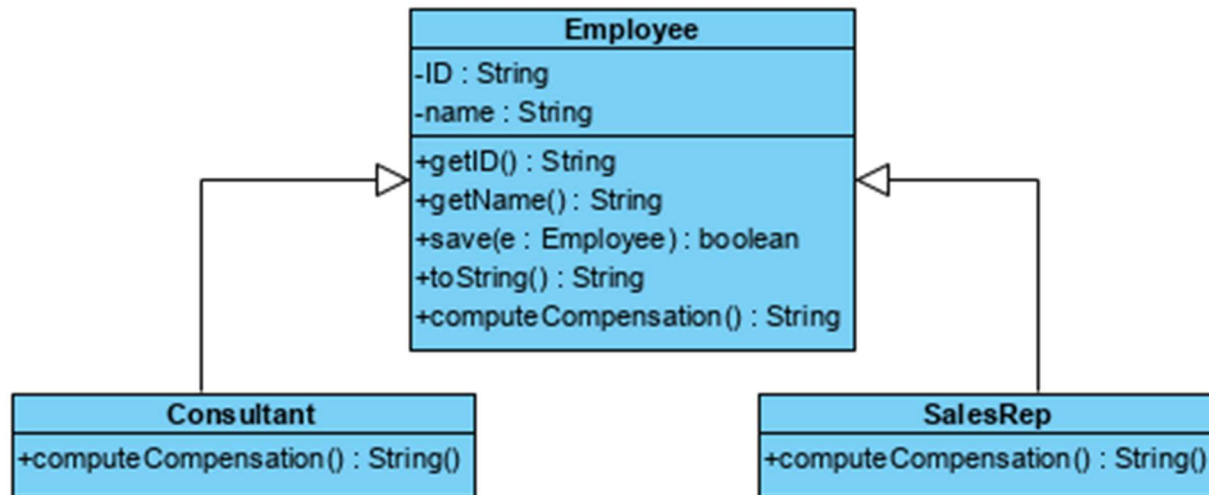
    @Override
    public String getId() { return this.id; }

    @Override
    public void save() { System.out.println("Save salesRep done"); }

    @Override
    public String computeCompensation() { return "salesRep"; }
}
```

Class MainApp	<pre>package Question2.Interface; public class MainApp { public static void main(String[] args) { Employee consultant = new Consultant(id: "1", name: "Thang"); Employee salesRep = new SalesRep(id: "2", name: "Nam"); System.out.println(consultant.toString()); System.out.println(consultant.computeCompensation()); System.out.println(salesRep.toString()); System.out.println(salesRep.computeCompensation()); } }</pre>
Kết quả chạy code	<pre>Question2.Interface.Consultant@4554617c Consultant Question2.Interface.SalesRep@74a14482 salesRep Process finished with exit code 0</pre>

- Abstract class



Tên class	Code
-----------	------

Class
Employee

```
package Question2.Abstract;

public abstract class Employee {
    private String name;
    private String id;

    public String getName() {
        return this.name;
    }

    public String getId() {
        return this.id;
    }

    public void save() { System.out.println("Do something"); }

    public String toString() {
        return id + " " + name;
    }

    public Employee(String name, String id) {
        this.name = name;
        this.id = id;
    }

    public abstract String computeCompensation();
}
```

Class Consultant	<pre> package Question2.Abstract; public class Consultant extends Employee { public Consultant(String id, String name) { super(id, name); } @Override public String computeCompensation() { return "Consultant"; } } </pre>
Class SalesRep	<pre> package Question2.Abstract; public class SalesRep extends Employee { public SalesRep(String id, String name) { super(id, name); } @Override public String computeCompensation() { return "SalesRep"; } } </pre>
Class MainApp	<pre> package Question2.Abstract; public class MainApp { public static void main(String[] args) { Employee consultant = new Consultant(id: "1", name: "Thang"); Employee salesRep = new SalesRep(id: "2", name: "Name"); Employee employee = new Employee(name: "3", id: "Hoang") { @Override public String computeCompensation() { return "abc do something"; } }; System.out.println(consultant.toString()); System.out.println(consultant.computeCompensation()); System.out.println(salesRep.toString()); System.out.println(salesRep.computeCompensation()); System.out.println(employee.toString()); System.out.println(employee.computeCompensation()); } } </pre>

Kết quả chạy code	<pre>Thang 1 Consultant Name 2 SalesRep Hoang 3 abc do something Process finished with exit code 0</pre>
-------------------	---

❖ **Kinh nghiệm:**

- Sự khác nhau giữa interface và abstract class

Abstract class	Interface
0) Là một lớp	Không phải là một lớp
1) Abstract class có phương thức abstract (không có thân hàm) và phương thức non-abstract (có thân hàm).	Interface chỉ có phương thức abstract . Từ java 8, nó có thêm các phương thức default và static .
2) Abstract class không hỗ trợ đa kế thừa .	Interface có hỗ trợ đa kế thừa
3) Abstract class có các biến final, non-final, static and non-static .	Interface chỉ có các biến static và final .
4) Abstract class có thể cung cấp nội dung cài đặt cho phương thức của interface .	Interface không thể cung cấp nội dung cài đặt cho phương thức của abstract class .
5) Abstract chứa cả phương thức trừu tượng và phương thức thành viên	Interface chỉ chứa các phương thức trừu tượng
6) Một phương thức trong abstract không sử dụng chỉ định từ truy xuất là private	Một phương thức trong interface không sử dụng chỉ định từ truy xuất là public
5) Từ khóa abstract được sử dụng để khai báo abstract class.	Từ khóa interface được sử dụng để khai báo interface.
6) Ví dụ: <pre>public abstract class Shape { public abstract void draw(); }</pre>	Ví dụ: <pre>public interface Drawable { void draw(); }</pre>

○ **Đặc điểm của abstract class**

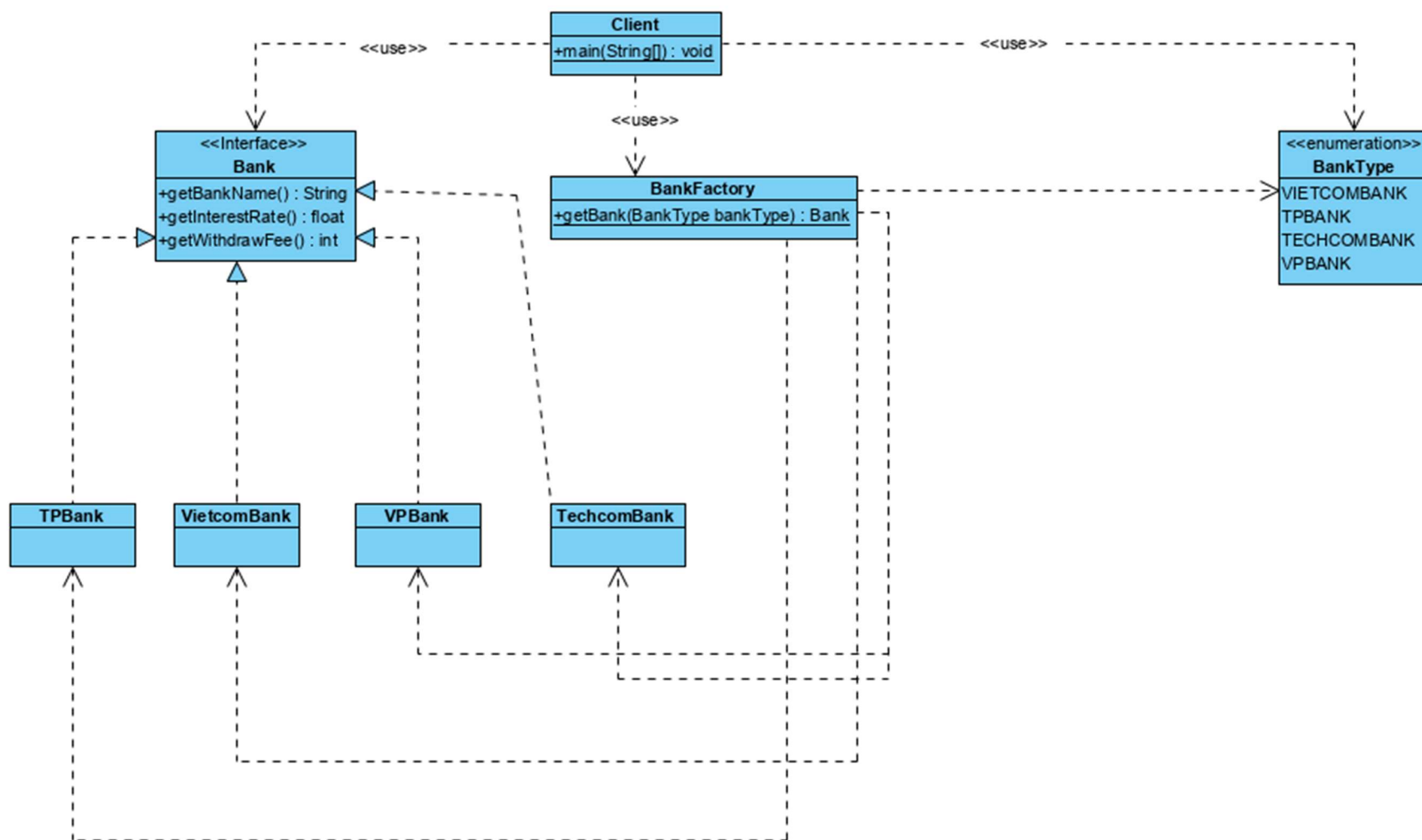
- Lớp trừu tượng có thể có các phương thức abstract hoặc non-abstract.
- Lớp trừu tượng có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.

3. Factory method

a) Kết quả chạy code

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java.exe" ...  
Bank name: TPBank  
  
Process finished with exit code 0
```

b) Sửa code để thêm 2 services



Code:

Tên class	Code
-----------	------

Class Bank	<pre>package Question3; public interface Bank { public String getBankName(); public float getInterestRate(); public int getWithdrawFee(); }</pre>
Class TP Bank	<pre>package Question3; public class TPBank implements Bank { @Override public String getBankName() { return "TPBank"; } @Override public float getInterestRate() { return 0.1f; } @Override public int getWithdrawFee() { return 100000; } }</pre>
Class TechcomBank	<pre>package Question3; public class TechcomBank implements Bank { @Override public String getBankName() { return "TechcomBank"; } @Override public float getInterestRate() { return 0.2f; } @Override public int getWithdrawFee() { return 220000; } }</pre>

Class
VietcomBank

```
package Question3;

public class VietcomBank implements Bank {
    @Override
    public String getBankName() { return "VietcomBank"; }

    @Override
    public float getInterestRate() { return 0.6f; }

    @Override
    public int getWithdrawFee() { return 8720000; }
}
```

Class Factory

```
package Question3;

public class BankFactory {
    @ public static final Bank getBank(BankType bankType) {
        switch (bankType) {
            case TPBANK:
                return new TPBank();
            case VPBANK:
                return new VPBank();
            case TECHCOMBANK:
                return new TechcomBank();
            case VIETCOMBANK:
                return new VietcomBank();
            default:
                throw new IllegalArgumentException("Not support this bank");
        }
    }
}
```

Enum BankType	<pre>package Question3; public enum BankType { VIETCOMBANK, TPBANK, TECHCOMBANK, VPBANK }</pre>
Class Client	<pre>package Question3; public class Client { public static void main(String[] args) { Bank bank = BankFactory.getBank(BankType.TECHCOMBANK); System.out.println("Bank name: " + bank.getBankName()); System.out.println("Interest: " + bank.getInterestRate()); System.out.println("Whithdraw fee: " + bank.getWithdrawFee()); bank = BankFactory.getBank(BankType.TPBANK); System.out.println("Bank name: " + bank.getBankName()); System.out.println("Interest: " + bank.getInterestRate()); System.out.println("Whithdraw fee: " + bank.getWithdrawFee()); bank = BankFactory.getBank(BankType.VIETCOMBANK); System.out.println("Bank name: " + bank.getBankName()); System.out.println("Interest: " + bank.getInterestRate()); System.out.println("Whithdraw fee: " + bank.getWithdrawFee()); bank = BankFactory.getBank(BankType.VPBANK); System.out.println("Bank name: " + bank.getBankName()); System.out.println("Interest: " + bank.getInterestRate()); System.out.println("Whithdraw fee: " + bank.getWithdrawFee()); } }</pre>

Kết quả chạy
code

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java.exe" ...  
Bank name: TechcomBank  
Interest: 0.2  
Whithdraw fee: 220000  
Bank name: TPBank  
Interest: 0.1  
Whithdraw fee: 100000  
Bank name: VietcomBank  
Interest: 0.6  
Whithdraw fee: 8720000  
Bank name: VPBank  
Interest: 0.3  
Whithdraw fee: 530000  
  
Process finished with exit code 0
```

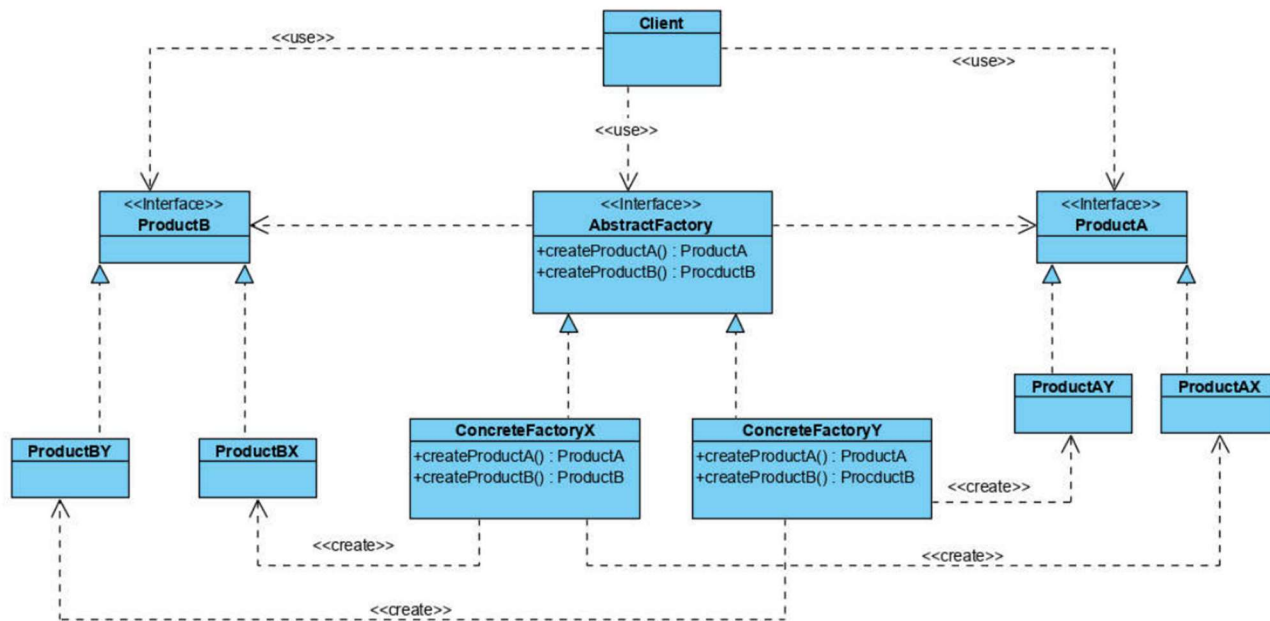
4. Tìm hiểu abstract factory pattern và builder pattern

❖ Abstract Factory pattern

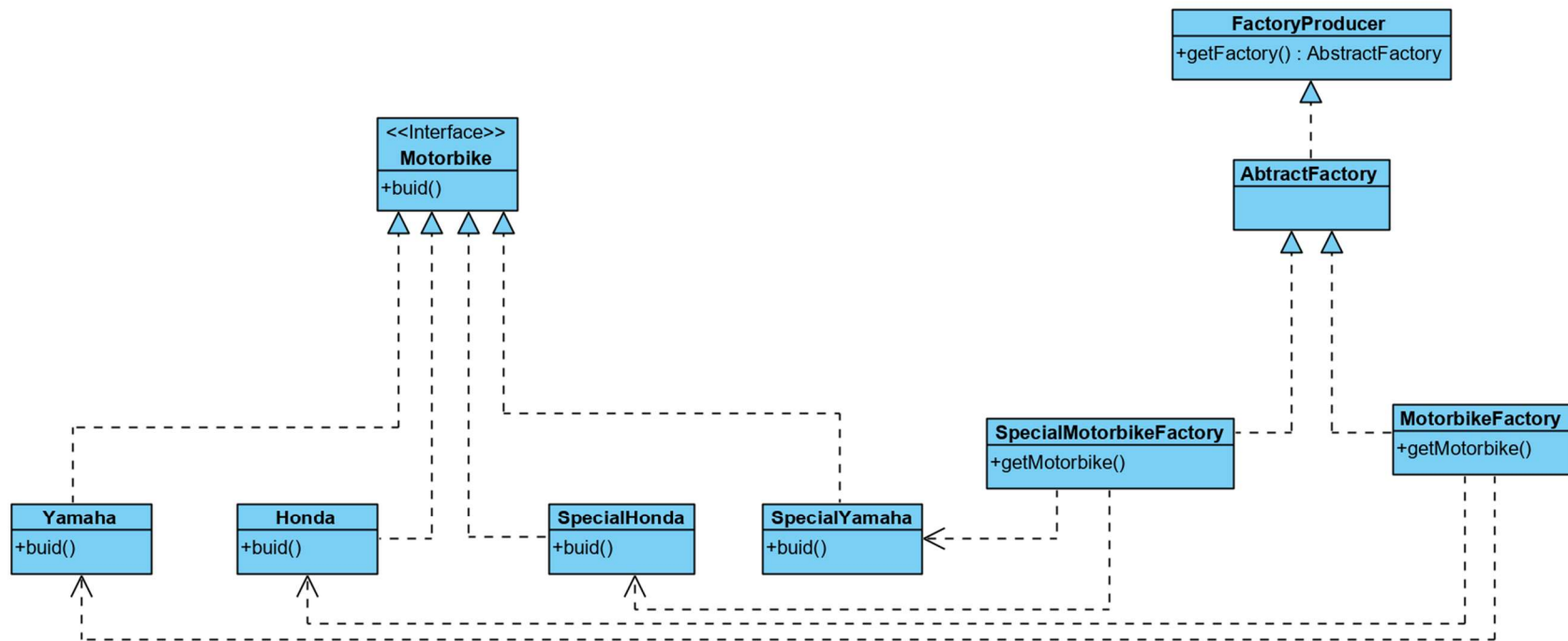
- Link tìm hiểu:

1. <https://www.uml-diagrams.org/design-pattern-abstract-factory-uml-class-diagram-example.html>
2. https://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm
3. <https://www.geeksforgeeks.org/abstract-factory-pattern/>
4. <https://www.javatpoint.com/abstract-factory-pattern>

- Pattern này hoạt động xung quanh một super-factory hay còn được gọi là factory của factory – cái mà tạo ra các factory khác.
- Trong pattern này, một interface chịu trách nhiệm tạo ra một factory của các object có liên quan mà không chỉ định rõ các lớp của chúng. Mỗi factory được tạo có thể cung cấp các đối tượng theo Factory pattern.
- Abstract diagram



- Diagram cho ứng dụng



• Code

Tên class	Code
Abstract class AbstractFactory	<pre> package Question4; public abstract class AbstractFactory { abstract Motorbike getMotobike(MotorbikeType motorbikeType); } </pre>

Class
MotorbikeFactory

```
package Question4;

public class MotorbikeFactory extends AbstractFactory {
    @Override
    @Motorbike getMotobike(MotorbikeType motorbikeType) {
        switch (motorbikeType) {
            case HONDA:
                return new Honda();
            case YAMAHA:
                return new Yamaha();
        }
        return null;
    }
}
```

Class
SpecialMotorbikeFacto
ry

```
package Question4;

public class SpecialMotorbikeFactory extends AbstractFactory {
    @Override
    @Motorbike getMotobike(MotorbikeType motorbikeType) {
        switch (motorbikeType) {
            case HONDA:
                return new SpecialHonda();
            case YAMAHA:
                return new SpecialYamaha();
        }
        return null;
    }
}
```


Class FactoryProducer	<pre> package Question4; public class FactoryProducer { @ public static AbstractFactory getFactory(MotorbikeVersion motorbikeVersion) { switch (motorbikeVersion) { case NORMAL: return new MotorbikeFactory(); case SPECIAL: return new SpecialMotorbikeFactory(); } return null; } } </pre>
Enum MotobikeType	<pre> package Question4; public enum MotorbikeType { ⚡ HONDA, YAMAHA } </pre>
Enum MotobikeVersion	<pre> package Question4; public enum MotorbikeVersion { NORMAL, SPECIAL } </pre>
Interface Motobike	<pre> package Question4.AbstracFactory; public interface Motorbike { void buidMotobike(); } </pre>

Class Honda	<pre> package Question4; public class Honda implements Motorbike { @Override public void buidMotobike() { System.out.println("Buid a honda motobike"); } } </pre>
Class Yamaha	<pre> package Question4; public class Yamaha implements Motorbike { @Override public void buidMotobike() { System.out.println("Buid a yamaha motobike"); } } </pre>
Class SpecialHonda	<pre> package Question4; public class SpecialHonda implements Motorbike { @Override public void buidMotobike() { System.out.println("Buid a special honda motobike"); } } </pre>
Class SpecialYamaha	<pre> package Question4; public class SpecialYamaha implements Motorbike { @Override public void buidMotobike() { System.out.println("Buid a special yamaha motobike"); } } </pre>

Class MainApp	<pre> package Question4; public class MainApp { public static void main(String[] args) { AbstractFactory normalFactory = FactoryProducer.getFactory(MotorbikeVersion.NORMAL); Motorbike motorbike = normalFactory.getMotobike(MotorbikeType.HONDA); motorbike.buidMotobike(); Motorbike motorbike2 = normalFactory.getMotobike(MotorbikeType.YAMAHA); motorbike2.buidMotobike(); AbstractFactory specialFactory = FactoryProducer.getFactory(MotorbikeVersion.SPECIAL); Motorbike motorbike3 = specialFactory.getMotobike(MotorbikeType.HONDA); motorbike3.buidMotobike(); Motorbike motorbike4 = specialFactory.getMotobike(MotorbikeType.YAMAHA); motorbike4.buidMotobike(); } } </pre>
Kết quả chạy code	<pre> C:\Users\thangbook\.jdk\corretto-1.8.0_252\bin\java.exe ... Buid a honda motobike Buid a yamaha motobike Buid a special honda motobike Buid a special yamaha motobike Process finished with exit code 0 </pre>

❖ Buidler pattern

○ Tổng quan

▪ Link tìm hiểu:

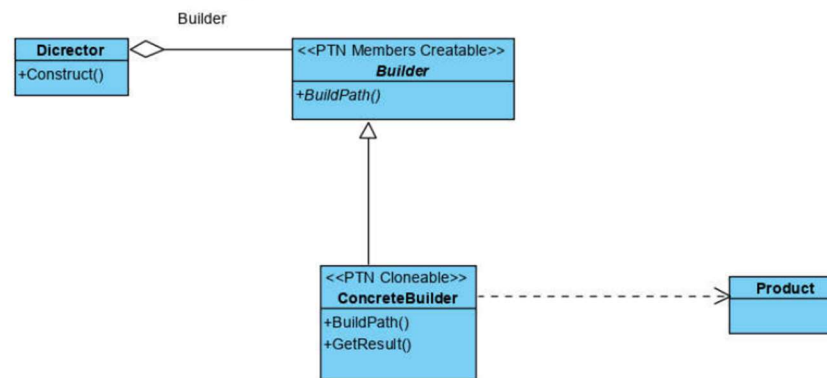
- https://sourcemaking.com/design_patterns/builder/java/2
- <https://www.visual-paradigm.com/tutorials/builderpattern.jsp>
- <https://www.javatpoint.com/builder-design-pattern>
- https://www.tutorialspoint.com/design_pattern/builder_pattern.htm
- <https://howtodoinjava.com/design-patterns/creational/builder-pattern-in-java/>
- <https://www.geeksforgeeks.org/builder-design-pattern/>

▪ Là pattern dành cho việc xây dựng các đối tượng phức tạp

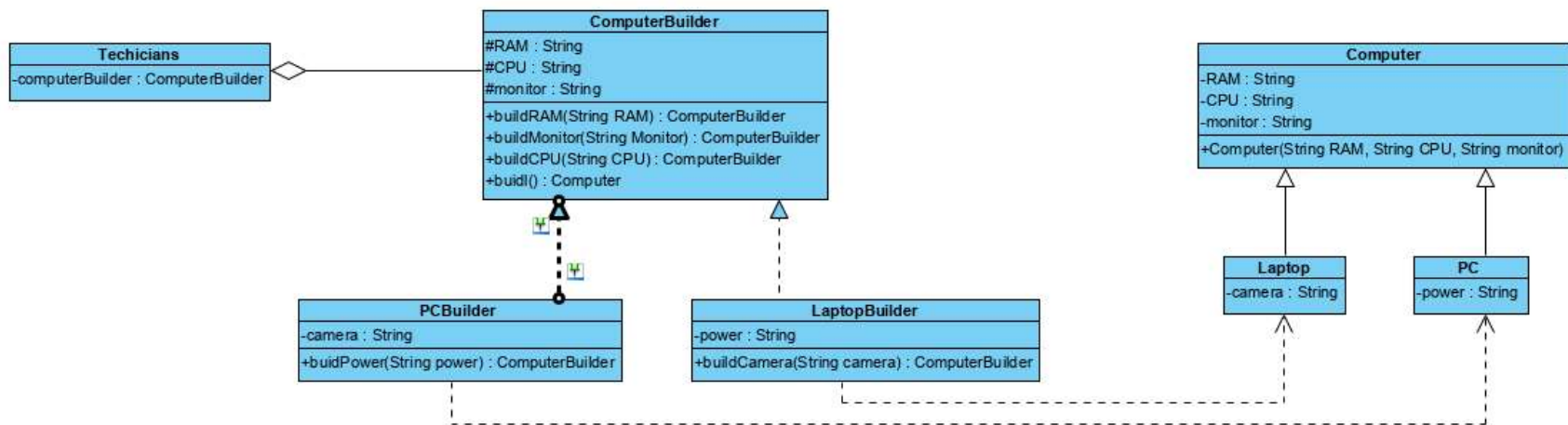
- Pattern này giải quyết vấn đề với số lượng lớn các tham số tùy chọn và trạng thái không nhất quán bằng cách cung cấp cách xây dựng đối tượng từng bước và cung cấp một phương thức sẽ trả về đối tượng cuối cùng.

- Diagram

- Basic



- Áp dụng



- Code

Tên class	Code
Class Computer	<pre>package Question4.Builder; public class Computer { private final String RAM; private final String CPU; private final String monitor; public Computer(String RAM, String CPU, String monitor) { this.RAM = RAM; this.CPU = CPU; this.monitor = monitor; } @Override public String toString() { return "Cake " + "RAM='" + RAM + '\'' + ", CPU='" + CPU + '\'' + ", monitor='" + monitor + '\''; } }</pre>

Class PC

```
package Question4.Builder;

public class PC extends Computer {
    private final String power;

    public PC(String RAM, String CPU, String monitor, String power) {
        super(RAM, CPU, monitor);
        this.power = power;
    }

    @Override
    public String toString() {
        return super.toString() + " power " + power;
    }
}
```

Class Laptop

```
package Question4.Builder;

public class Laptop extends Computer {
    private final String camera;

    public Laptop(String RAM, String CPU, String monitor, String camera) {
        super(RAM, CPU, monitor);
        this.camera = camera;
    }

    @Override
    public String toString() {
        return super.toString() + " camera " + camera;
    }
}
```

Interface ComputerBuilder

```
package Question4.Builder;

public abstract class ComputerBuilder {
    protected String RAM;
    protected String CPU;
    protected String monitor;

    public ComputerBuilder buildRAM(String RAM) {
        this.RAM = RAM;
        return this;
    }

    public ComputerBuilder buildCPU(String CPU) {
        this.CPU = CPU;
        return this;
    }

    public ComputerBuilder buildMonitor(String monitor) {
        this.monitor = monitor;
        return this;
    }

    public abstract Computer build();
}
```

Class PCBuilder

```
package Question4.Builder;

public class PCBuilder extends ComputerBuilder {
    private String power;

    public ComputerBuilder buildPower(String power) {
        this.power = power;
        return this;
    }

    @Override
    public Computer build() {
        return new PC(RAM, CPU, monitor, power);
    }
}
```

Class LaptopBuilder	<pre> package Question4.Builder; public class LaptopBuilder extends ComputerBuilder { private String camera; public ComputerBuilder buildCamera(String camera) { this.camera = camera; return this; } @Override public Computer build() { return new Laptop(RAM, CPU, monitor, camera); } } </pre>
Class Technicians	<pre> package Question4.Builder; public class Technicians { public static void main(String[] args) { PCBuilder pcBuilder = new PCBuilder(); LaptopBuilder laptopBuilder = new LaptopBuilder(); Computer computer1 = pcBuilder.buildPower("400W").buildCPU("i7-8700").buildMonitor("dell 2020").buildRAM("16GB").build(); System.out.println(computer1.toString()); Computer computer2 = laptopBuilder.buildCamera("xxx").buildCPU("i5-8500").buildMonitor("abc").buildRAM("32GB").build(); System.out.println(computer2.toString()); } } </pre>
Kết quả chạy code	<pre> "C:\Program Files\Java\jdk-13\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ Cake RAM='16GB', CPU='i7-8700', monitor='dell 2020' power 400W Cake RAM='32GB', CPU='i5-8500', monitor='abc' camera xxx Process finished with exit code 0 </pre>

5.

❖ Lợi ích và sử dụng của từng pattern:

- **Factory pattern:**

- Lợi ích:

- ✓ Cung cấp hướng tiếp cận với Interface thay thì các implement
 - ✓ Che giấu sự phức tạp của việc khởi tạo các đối tượng với người dùng (client)

- ✓ Độc lập giữa việc khởi tạo đối tượng và hệ thống sử dụng, ...
- Được sử dụng khi:
 - ✓ Chúng ta có một super class với nhiều class con và dựa trên đầu vào, chúng ta cần trả về một class con.
 - ✓ Mô hình này giúp chúng ta đưa trách nhiệm của việc khởi tạo một lớp từ phía người dùng (client) sang lớp Factory.
 - ✓ Chúng ta không biết sau này sẽ cần đến những lớp con nào nữa. Khi cần mở rộng, hãy tạo ra sub class và implement thêm vào factory method cho việc khởi tạo sub class này.

• ***Abstract Factory Pattern:***

- Lợi ích:
 - ✓ Các lợi ích của Abstract Factory Pattern cũng tương tự như Factory Pattern như: cung cấp hướng tiếp cận với Interface thay thì các implement, che giấu sự phức tạp của việc khởi tạo các đối tượng với người dùng (client), độc lập giữa việc khởi tạo đối tượng và hệ thống sử dụng, ...
 - ✓ Giúp tránh được việc sử dụng điều kiện logic bên trong Factory Pattern. Khi một Factory Method lớn (có quá nhiều sử lý if-else hay switch-case), chúng ta nên sử dụng theo mô hình Abstract Factory để dễ quản lý hơn (cách phân chia có thể là gom nhóm các sub-class cùng loại vào một Factory).
 - ✓ Abstract Factory Pattern là factory của các factory, có thể dễ dàng mở rộng để chứa thêm các factory và các sub-class khác.
 - ✓ Dễ dàng xây dựng một hệ thống đóng gói (encapsulate): sử dụng được với nhiều nhóm đối tượng (factory) và tạo nhiều product khác nhau.
- Được sử dụng khi:
 - ✓ Tạo các đối tượng tương tự nhau

- ✓ Cung cấp phương thức hoàn chỉnh (có thể tổng hợp thành library) để sinh các đối tượng
- ✓ Tạo các đối tượng đặc biệt từ các lớp cha
- ✓ Dễ dàng tạo extends system từ system cũ

- ***Builder Pattern:***

- Lợi ích:

- ✓ Hỗ trợ, loại bớt việc phải viết nhiều constructor.
 - ✓ Code dễ đọc, dễ bảo trì hơn khi số lượng thuộc tính (property) bắt buộc để tạo một object từ 4 hoặc 5 property.
 - ✓ Giảm bớt số lượng constructor, không cần truyền giá trị null cho các tham số không sử dụng.
 - ✓ Ít bị lỗi do việc gán sai tham số khi mà có nhiều tham số trong constructor: bởi vì người dùng đã biết được chính xác giá trị gì khi gọi phương thức tương ứng.
 - ✓ Đối tượng được xây dựng an toàn hơn: bởi vì nó đã được tạo hoàn chỉnh trước khi sử dụng.
 - ✓ Cung cấp cho bạn kiểm soát tốt hơn quá trình xây dựng: chúng ta có thể thêm xử lý kiểm tra ràng buộc trước khi đối tượng được trả về người dùng.
 - ✓ Có thể tạo đối tượng immutable.

- Được sử dụng khi:

- ✓ Tạo một đối tượng phức tạp: có nhiều thuộc tính (nhiều hơn 4) và một số bắt buộc (required), một số không bắt buộc (optional).
 - ✓ Khi có quá nhiều hàm constructor, bạn nên nghĩ đến Builder.
 - ✓ Muốn tách rời quá trình xây dựng một đối tượng phức tạp từ các phần tạo nên đối tượng.

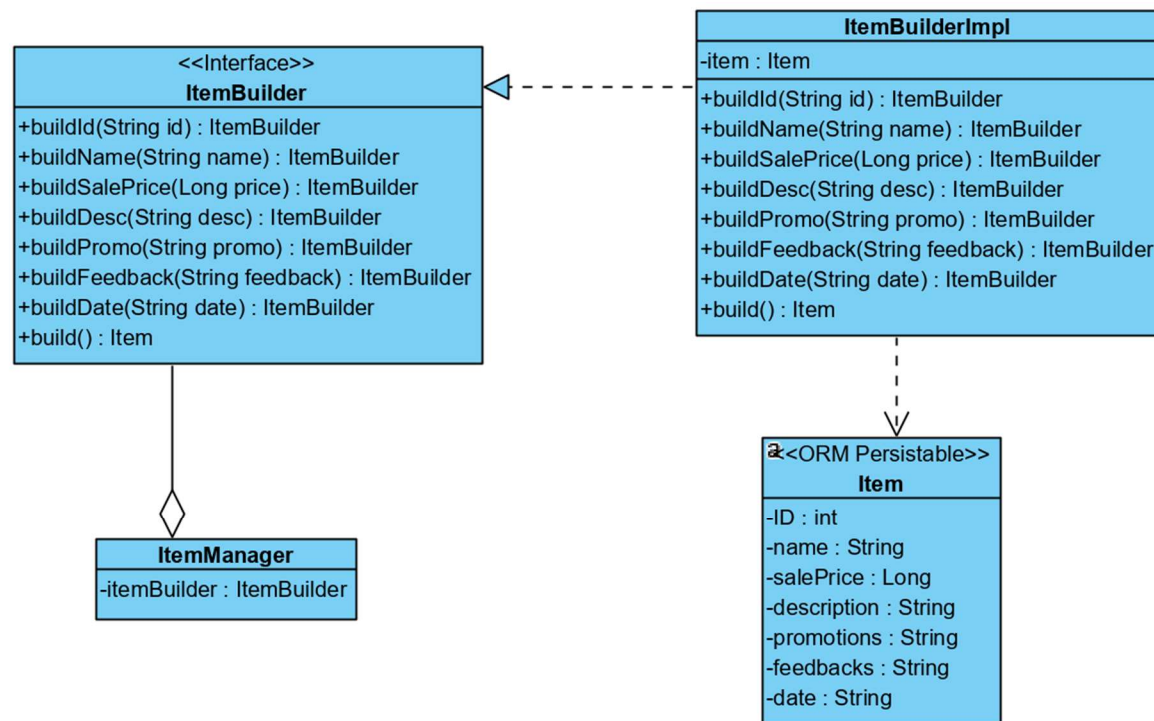
✓ Muốn kiểm soát quá trình xây dựng.

✓ Khi người dùng (client) mong đợi nhiều cách khác nhau cho đối tượng được xây dựng.

❖ Ứng dụng vào ứng dụng của mình

Chức năng	Pattern
Make Order	Builder Pattern
Manage Item	Builder Pattern
Chekout	Factory Pattern
Sign up	Builder Pattern
View Computer	Builder Pattern

❖ Áp dụng builder pattern cho class item:



❖ Code:

Tên class	Code
Class Item	<pre>package Question5; public class Item { private String id; private String desc; private long salePrice; private String date; private String name; private String promo; private String feedback; public Item(String id, String desc, long salePrice, String date, String name, String promo, String feedback) { this.id = id; this.desc = desc; this.salePrice = salePrice; this.date = date; this.name = name; this.promo = promo; this.feedback = feedback; } }</pre>
Interface ItemBuilder	<pre>package Question5; public interface ItemBuilder { ItemBuilder buildId(String id); ItemBuilder buildName(String name); ItemBuilder buildDesc(String desc); ItemBuilder buildDate(String date); ItemBuilder buildFeedback(String feedback); ItemBuilder buildPromo(String promo); ItemBuilder buildSalePrice(long salePrice); Item build(); }</pre>

Class ItemBuilderImpl

```
package Question5;

public class ItemBuilderImpl implements ItemBuilder {
    private String id;
    private String desc = "";
    private long salePrice;
    private String date = "";
    private String name = "";
    private String promo = "";
    private String feedback = "";

    @Override
    public ItemBuilder buildId(String id) {
        this.id = id;
        return this;
    }

    @Override
    public ItemBuilder buildName(String name) {
        this.name = name;
        return this;
    }

    @Override
    public ItemBuilder buildDesc(String desc) {
        this.desc = desc;
        return this;
    }
}
```

```
@Override
public ItemBuilder buildDate(String date) {
    this.date = date;
    return this;
}

@Override
public ItemBuilder buildFeedback(String feedback) {
    this.feedback = feedback;
    return this;
}

@Override
public ItemBuilder buildPromo(String promo) {
    this.promo = promo;
    return this;
}

@Override
public ItemBuilder buildSalePrice(long salePrice) {
    this.salePrice = salePrice;
    return this;
}

@Override
public Item build() {
    return new Item(id, desc, salePrice, date, name, promo, feedback);
}
```

Class ItemManager

```
package Question5;

public class ItemManager {
    public static void main(String[] args) {
        ItemBuilder itemBuilder1 = new ItemBuilderImpl();
        Item item1 = itemBuilder1.buidlId("1").buidlDate("20/10").buidlDesc("Giam gia").build();
        System.out.println(item1);
        ItemBuilder itemBuilder2 = new ItemBuilderImpl();
        Item item2 = itemBuilder2.buidlId("2").buidlFeedback("Tot").buidlSalePrice(20).build();
        System.out.println(item2);
    }
}
```

Kết quả chạy code

```
C:\Users\thangbook\.jdk\corretto-1.8.0_252\bin\java.exe ...
Item{id='1', desc='Giam gia', salePrice=0, date='20/10', name='', promo='', feedback=''}
Item{id='2', desc='', salePrice=20, date='', name='', promo='', feedback='Tot'}
Process finished with exit code 0
```