**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

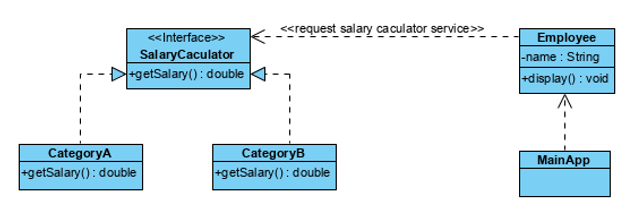
Phạm Ngọc Hoàng B16DCCN159

**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**

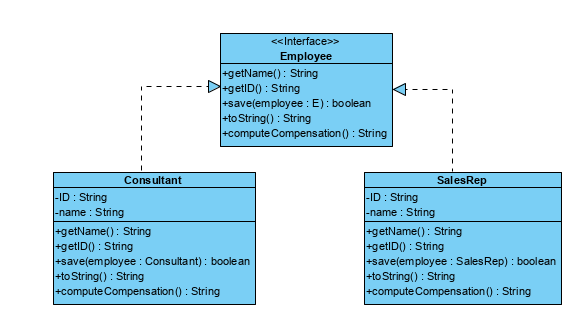
* Thiết kế:
* Code:

|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Class SalaryCalculator |  |
| Class CategoryA |  |
| Class CategoryB |  |
| Class Employee |  |
| Class MainApp |  |
| Kết quả chạy code |  |

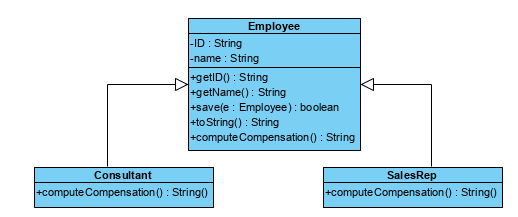
* **Kinh nghiệm**:
* Phân tích ví dụ:  
  ▪ Đầu tiên, chúng ta sẽ thấy là chúng ta cần tính lương cho các nhân viên dù  
  là bất kỳ phòng ban nào -> có phương thức getSalary() tính lương là hành  
  động cần thực hiện giống nhau -> tạo một class interface đa năng để tính  
  lương theo phòng ban được.  
  ▪ 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 Contructor 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.

1. **Sử dụng abstract hoặc interface để thiết kế class Employee với các phương**

**thức:**

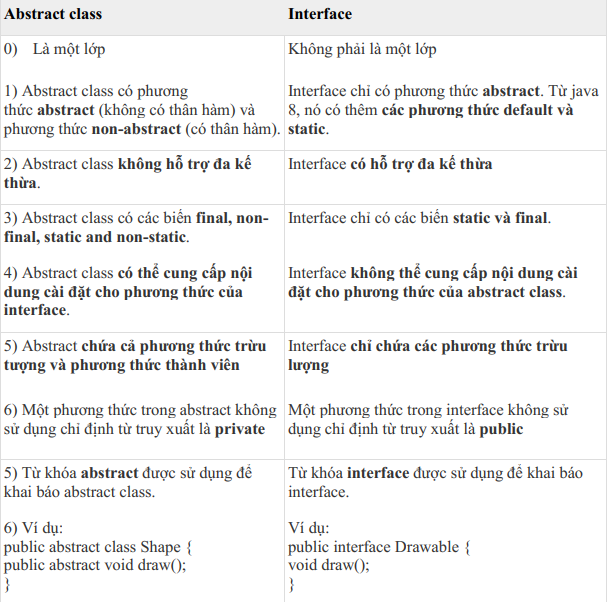
* Lưu dữ liệu employee
* Hiển thị dữ liệu employee
* Truy cập các thuộc tính của employee
* Tính toán
* Mô hình
* Code
  + Interface

|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Class Employee |  |
| Class Consultant |  |
| Class SalesRep |  |
| Class MainApp |  |
| Kết quả chạy code |  |

* Abstract class

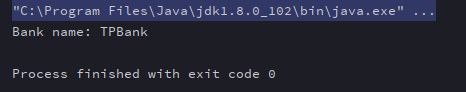
|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Class Employee |  |
| Class Consultant |  |
| Class SalesRep |  |
| Class MainApp |  |
| Kết quả chạy code |  |

* **Kinh nghiệm:**
  + **Sự khác nhau giữa interface và abstract class**

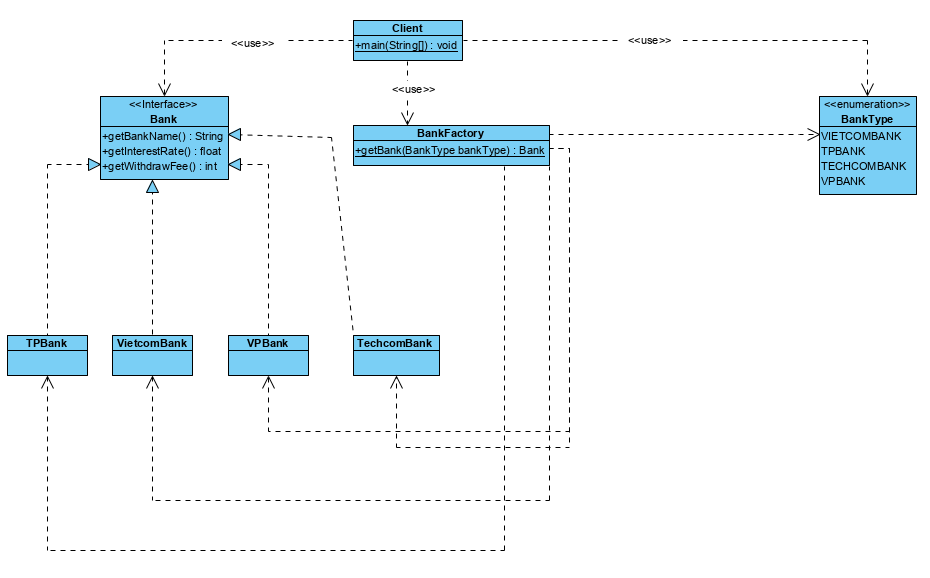


* + **Đặc điểm của abstract class**
    - Lớp trừu tượng có thể có các phương thức abstract hoặc non-abtract.
    - Lớp trừu tượng có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.

1. **Factory method**
2. Kết quả chạy code

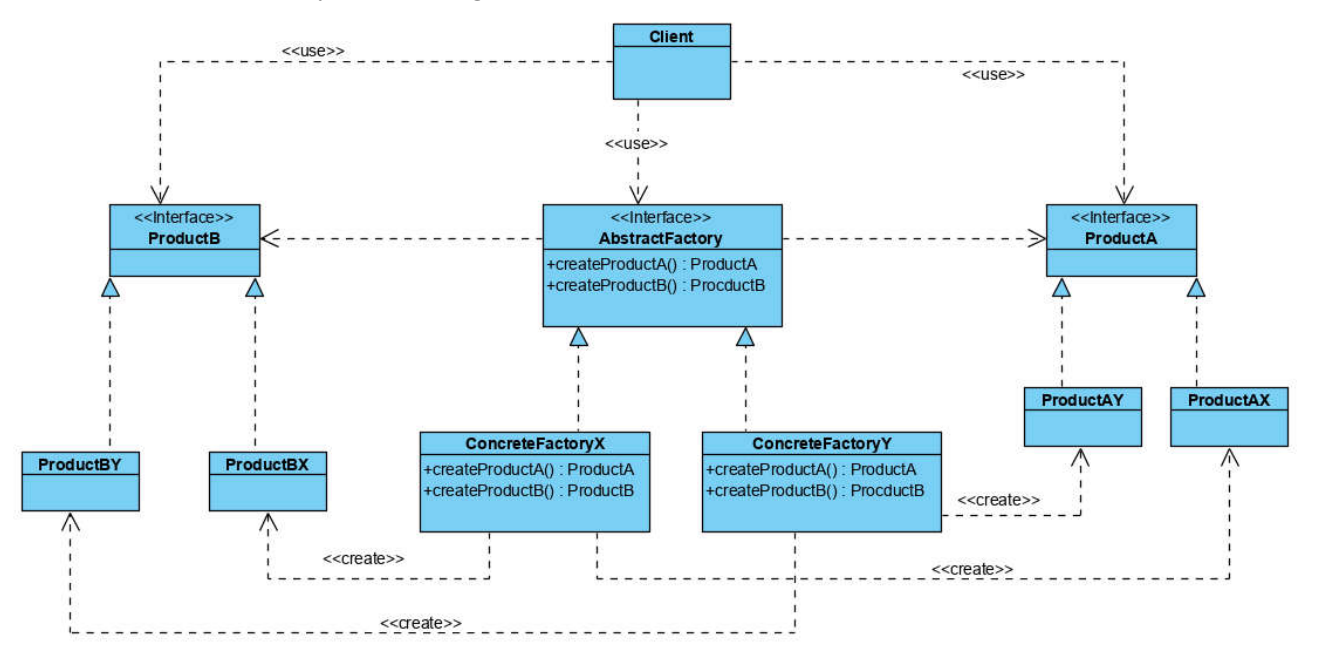


1. Sửa code để thêm 2 services

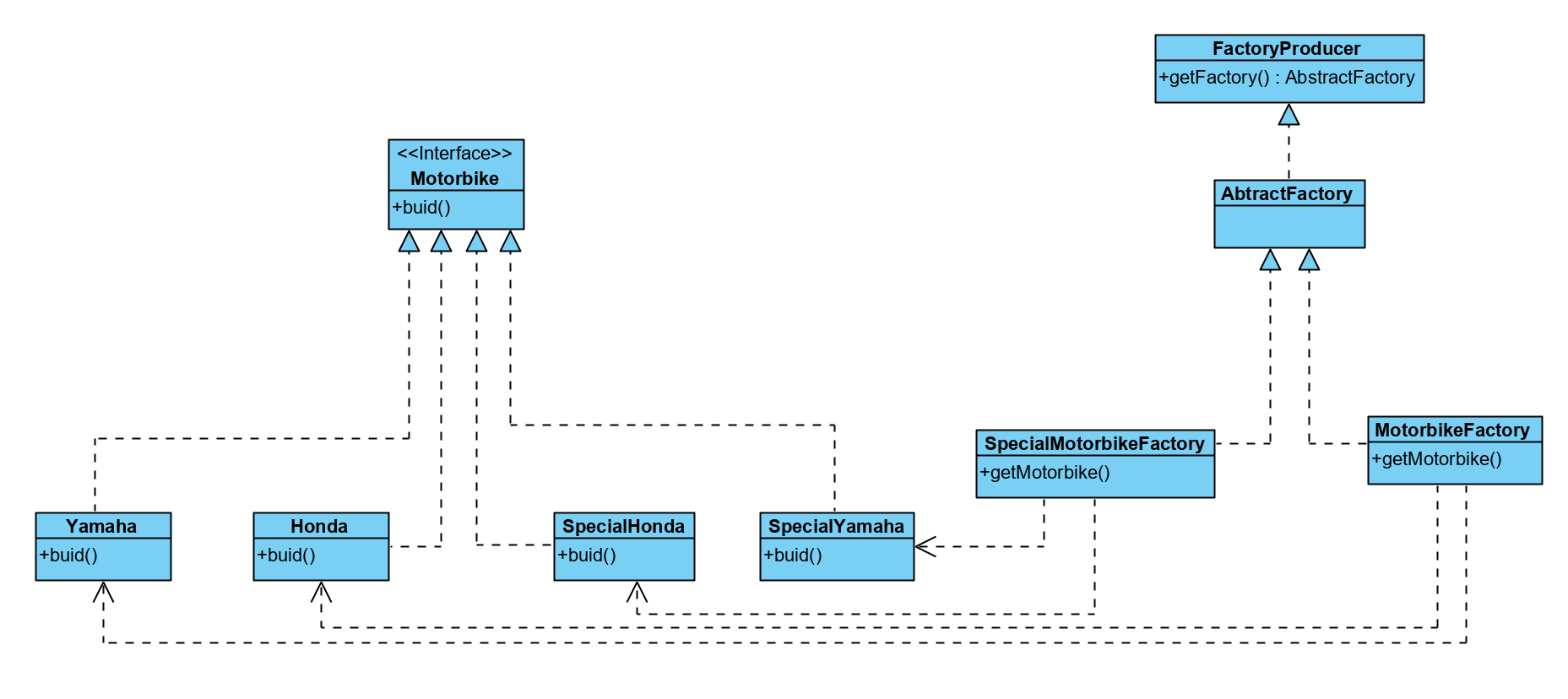
Code:

|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Class Bank |  |
| Class TP Bank |  |
| Class TechcomBank |  |
| Class VietcomBank |  |
| Class Factory |  |
| Enum BankType |  |
| Class Client |  |
| Kết quả chạy code |  |

1. **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



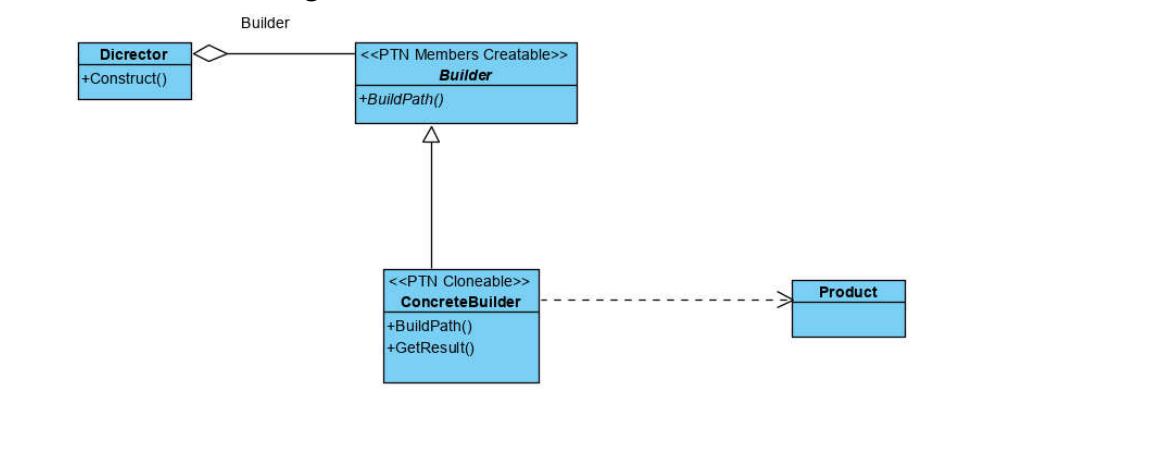
* + - Diagram cho ứng dụng

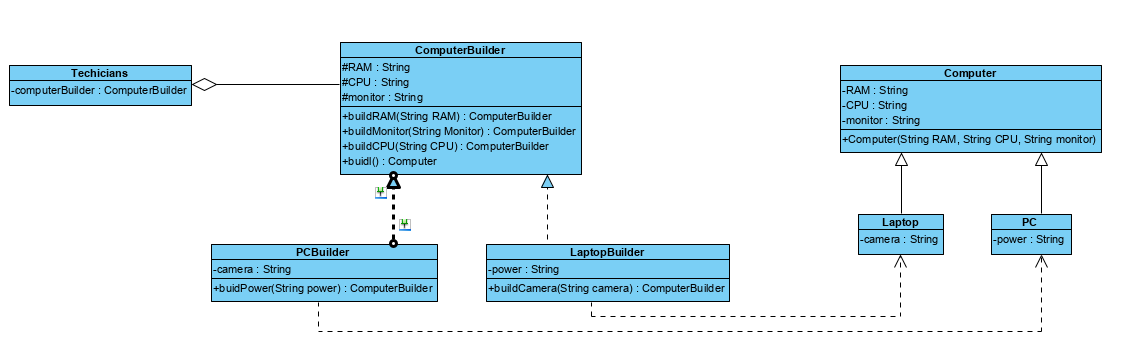


* + - Code

|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Abstract class AbstractFactory |  |
| Class MotorbikeFactory |  |
| Class SpecialMotorbikeFactory |  |
| Class FactoryProducer |  |
| Enum MotobikeType |  |
| Enum MotobikeVersion |  |
| Interface Motobike |  |
| Class Honda |  |
| Class Yamaha |  |
| Class SpecialHonda |  |
| Class SpecialYamaha |  |
| Class MainApp |  |
| Kết quả chạy code |  |

* 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



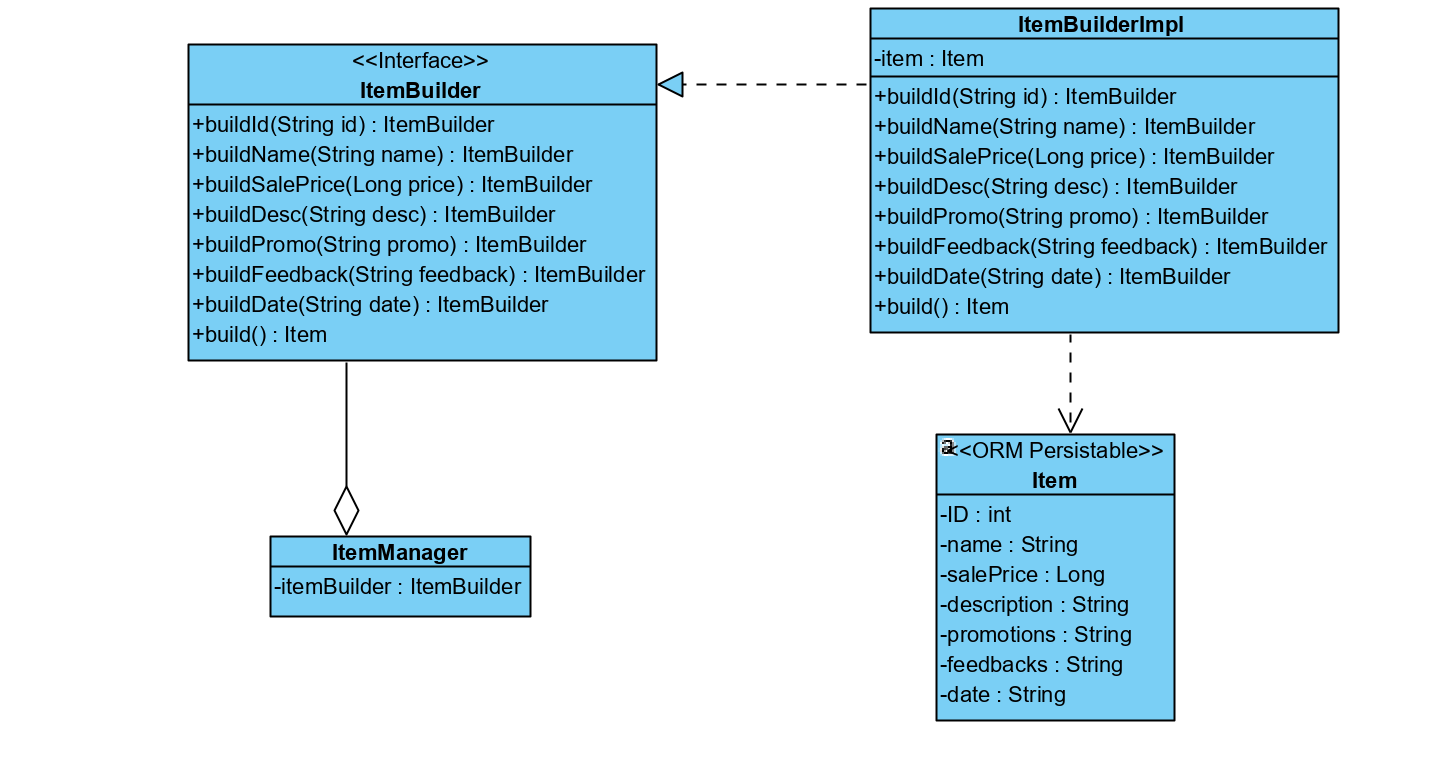
* + - Áp dụng
    - Code

|  |  |
| --- | --- |
| **Tên class** | **Code** |
| Class Computer |  |
| Class PC |  |
| Class Laptop |  |
| Interface ComputerBuilder |  |
| Class PCBuilder |  |
| Class LaptopBuilder |  |
| Class Technicians |  |
| Kết quả chạy code |  |

* + 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 dạng 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:
        + Taọ 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 (propery) 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 (requried), 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 |  |
| Interface ItemBuilder |  |
| Class ItemBuilderImpl |  |
| Class ItemManager |  |
| Kết quả chạy code |  |