Design Patterns

TS. Huỳnh Hữu Nghĩa
huynhhuunghia@iuh.edu.vn

Nội dung

- Giới thiệu về Design Patterns
- Phân loại Design Patterns
- UML
- Một số Pattern thông dụng

Design Pattern là gì?

- Design pattern diễn tả các thực hành tốt nhất được sử dụng bởi các nhà phát triển phần mềm hướng đối tượng giàu kinh nghiệm.

- Design pattern là những giải pháp cho các vấn đề chung mà các nhà phát triển phần mềm phải đối mặt trong quá trình phát triển phần mềm.

- Những giải pháp này đã thu được bằng cách *thử* và *sai* bởi nhiều nhà phát triển phần mềm trong khoảng thời gian dài.

Sử dụng Design Pattern

Có 2 cách sử dụng chính trong việc phát triển phần mềm

- **Nền tảng chung cho các nhà phát triển.** *Design patterns cung cấp một thuật ngữ chuẩn và riêng biệt cho kịch bản cụ thể. Ví dụ, mẫu thiết kế singleton dùng để chỉ một đối tượng duy nhất và có thể nói rằng chương trình theo mẫu singleton.*

- **Thực hành tốt nhất.** *Design patterns đã được phát triển trong thời gian dài và chúng cung cấp các giải pháp cho các vấn đề nhất định phải đối mặt trong việc phát triển phần mềm. Học các mẫu này giúp những người phát triển chưa kinh nghiệm để học thiết kế phần mềm dễ dàng và nhanh hơn.*

Các kiểu Design Pattern

Gồm 23 kiểu design patterns chia làm 3 loại

- **Creational Patterns.** *Các mẫu thiết kế cung cấp cách để tạo các đối tương trong khi ẩn lôgic tạo, không phải khởi tạo các đối tượng trực tiếp sử dụng toán tử* **new**. *Điều này cho phép chương trình linh hoạt hơn trong quyết định các đối tượng cần được tạo ra cho một trường hợp sử dụng duy nhất.*
- **Structure Patterns.** *Các mẫu thiết kế liên quan đến cấu trúc class và object. Khái niệm kế thừa được sử dụng để tạo interfaces và các cách xác định để tạo các đối tượng có những tính năng mới.*
- **Behavioral Patterns.** *Các mẫu thiết kế này đặc biệt liên quan đến giao tiếp giữa các đối tượng.*

Các kiểu Design Pattern

Ngoài ra còn một loại khác

- **J2EE design patterns.** *Các mẫu thiết kế này đặc biệt liên quan đến tầng biểu diễn. Các mẫu này được ủng hộ bởi Sun Java Center.*

Unified Modeling Language (UML)

UML

UML cung cấp 12 lược đồ hướng tới biểu diễn phân tích các yêu cầu và thiết kế giải pháp của ứng dụng. Có thể phân làm 3 loại như sau:

- **Structure diagrams.** *UML cung cấp 4 lược đồ cấu trúc, nó có thể được sử dụng để biểu diễn cấu tĩnh của một ứng dụng.*

  1. Class diagrams
  2. Object diagrams
  3. Component diagram
  4. Deployment diagram

UML

UML cung cấp 12 lược đồ hướng tới biểu diễn phân tích các yêu cầu và thiết kế giải pháp của ứng dụng. Có thể phân làm 3 loại như sau:

- **Behavior diagrams.** *UML cung cấp 5 lược đồ hành vi, nó có thể được sử dụng để biểu diễn các khía cạnh hành vi của một ứng dụng.*

  1. Use Case diagrams
  2. Sequence diagrams
  3. Activity diagram
  4. Collaboration diagram
  5. Statechart diagram

UML

UML cung cấp 12 lược đồ hướng tới biểu diễn phân tích các yêu cầu và thiết kế giải pháp của ứng dụng. Có thể phân làm 3 loại như sau:

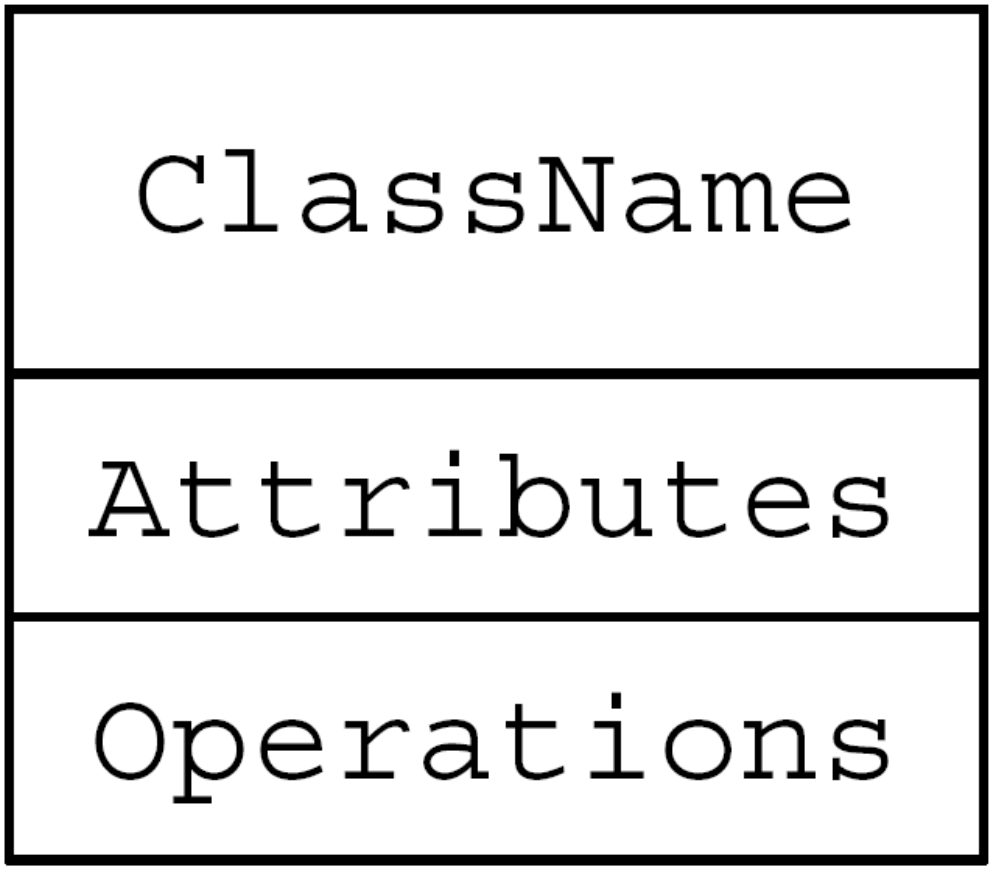
- **Model management diagrams.** *UML cung cấp 3 lược đồ quản lý mô hình, nó có thể được sử dụng để biểu diễn các mô đun ứng khác nhau được tổ chức và quản lý như thế nào.*

  1. Packages
  2. Subsystems
  3. Models

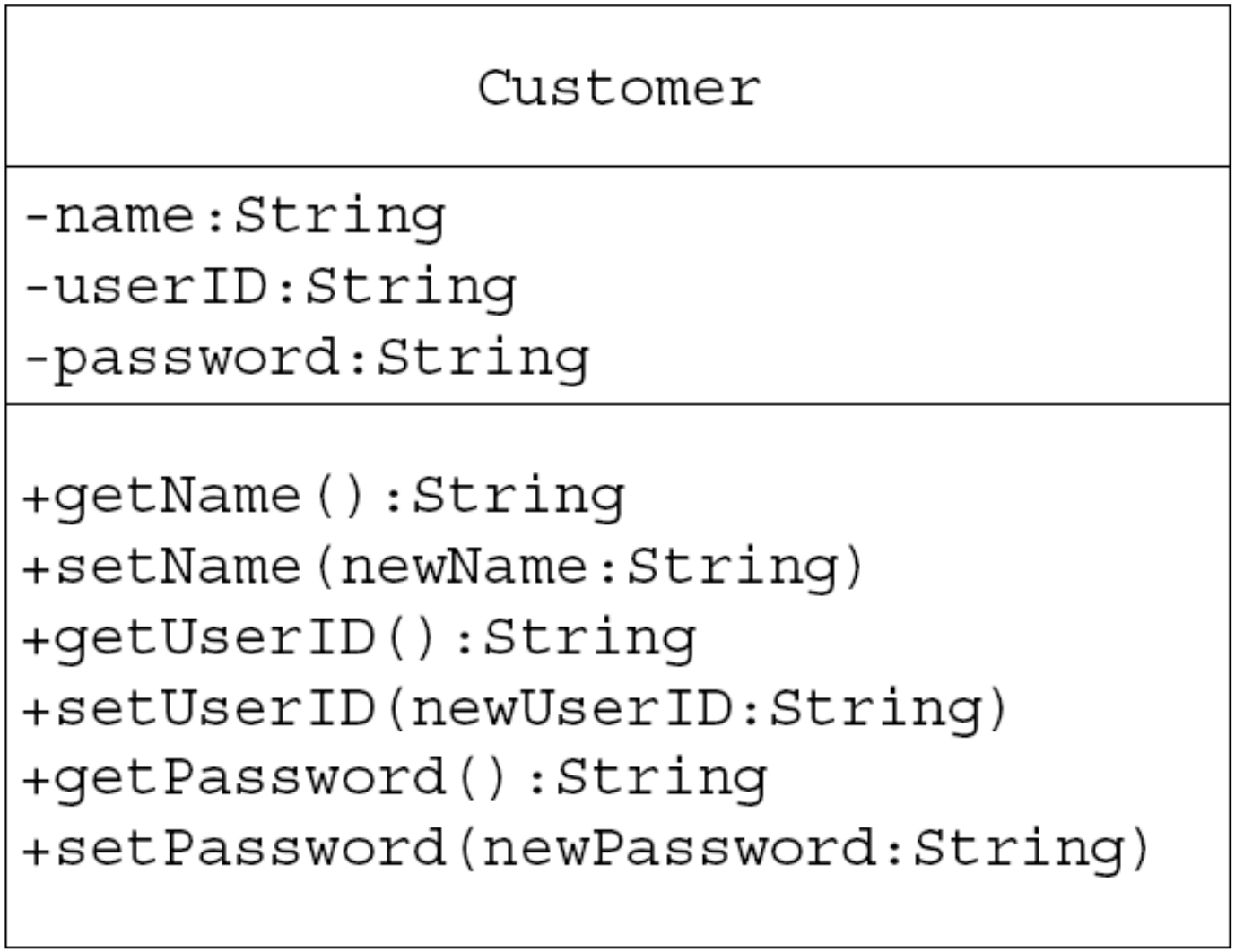
Class Diagrams

Class diagrams là thành phần của lược đồ cấu trúc và được sử dụng để mô tả cấu trúc tĩnh của hệ thống. Cấu trúc và hành vi của class và sự kết hợp của chúng với classes khác được miêu tả bên trong của class diagram.

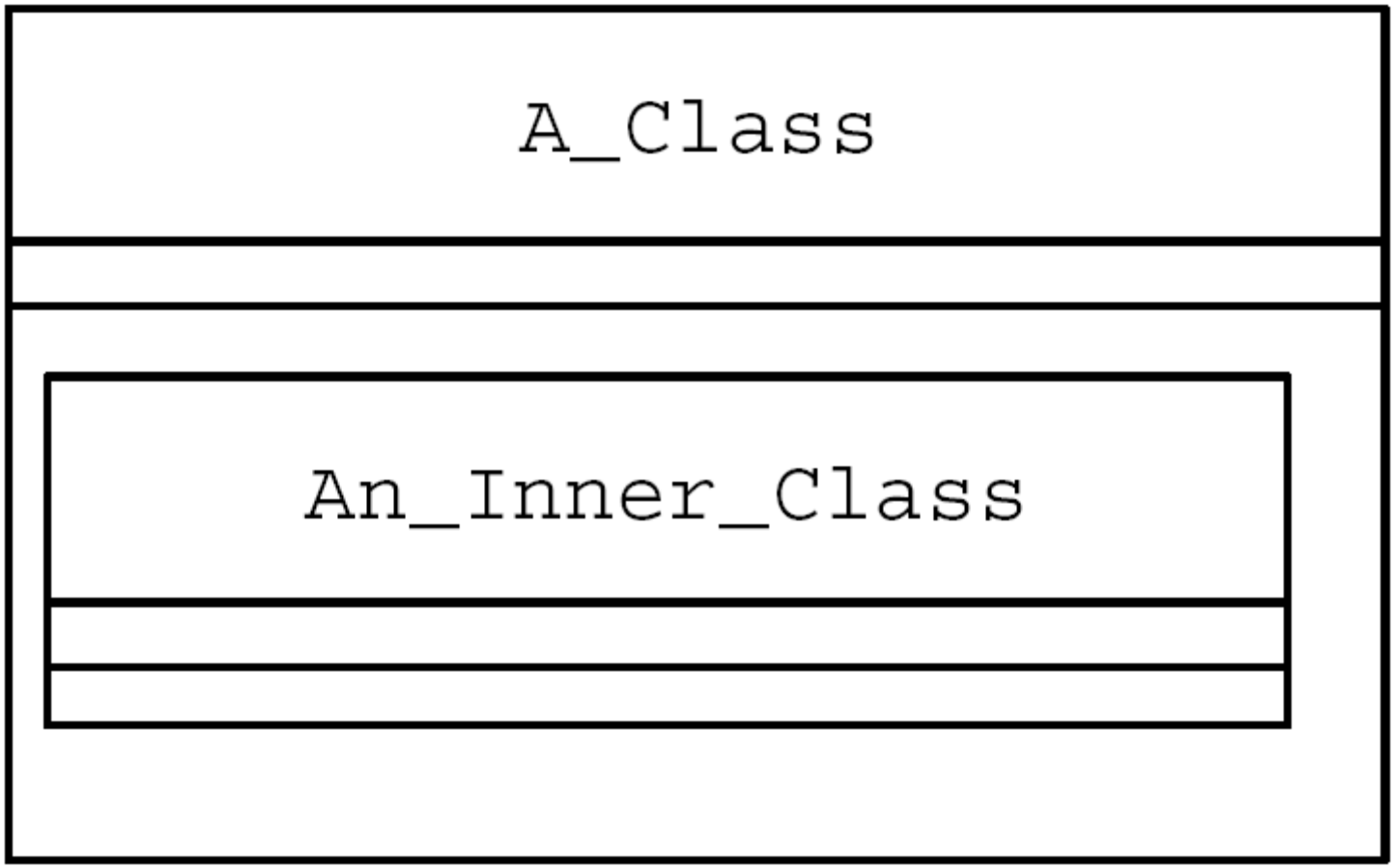
Class



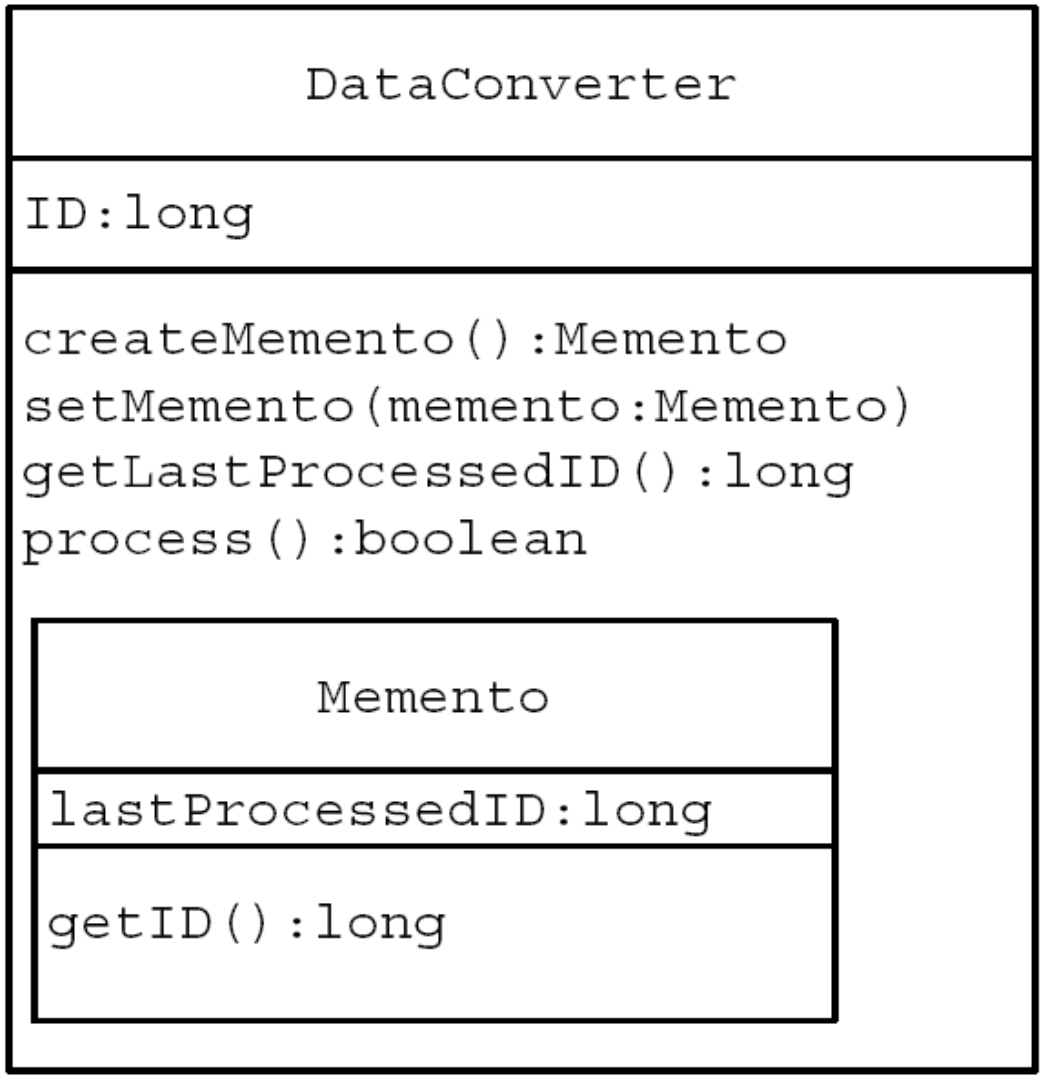
Class


Customer
-name:String
-userID:String
-password:String
+getName():String
+setName(newName:String)
+getUserID():String
+setUserID(newUserID:String)
+getPassword():String
+setPassword(newPassword:String)

Inner Class

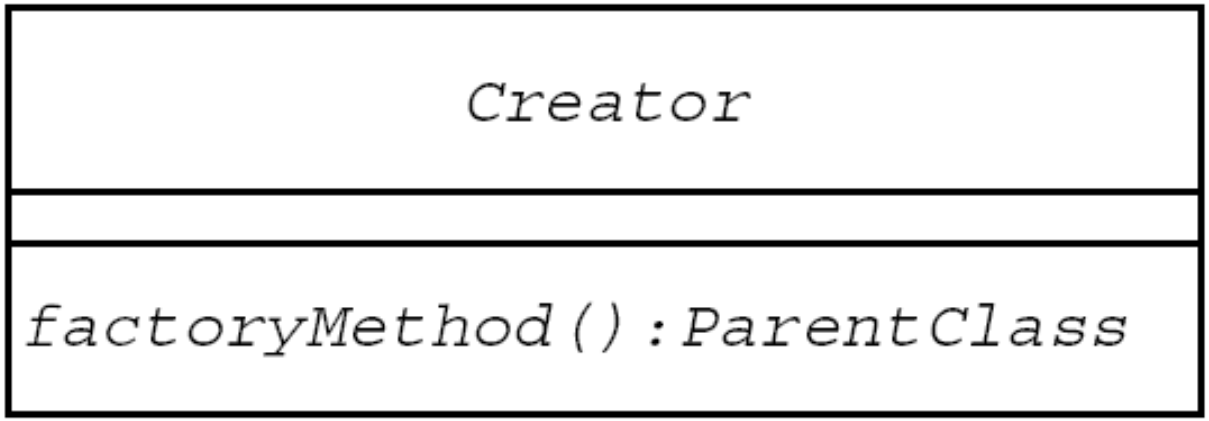


Inner Class


DataConverter
ID:long
createMemento():Memento
setMemento(memento:Memento)
getLastProcessedID():long
process():boolean
Memento
lastProcessedID:long
getID():long

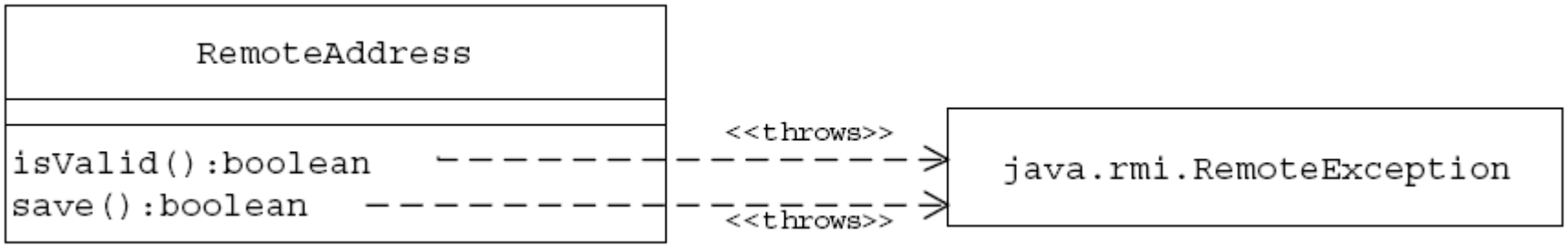
Abstract Class/Method

Một phương thức không có phần thân (trong một class) được gọi là *abstract method*. Một *class* có ít nhất một *abstract method* được xem là *abstract class*. *Client objects* không thể khởi tạo một *abstract class*. Một *subclass* của *abstract class* phải hiện thực tất cả *abstract method* của *abstract class* hoặc được khai báo như *abstract class* của chính nó.



Exception

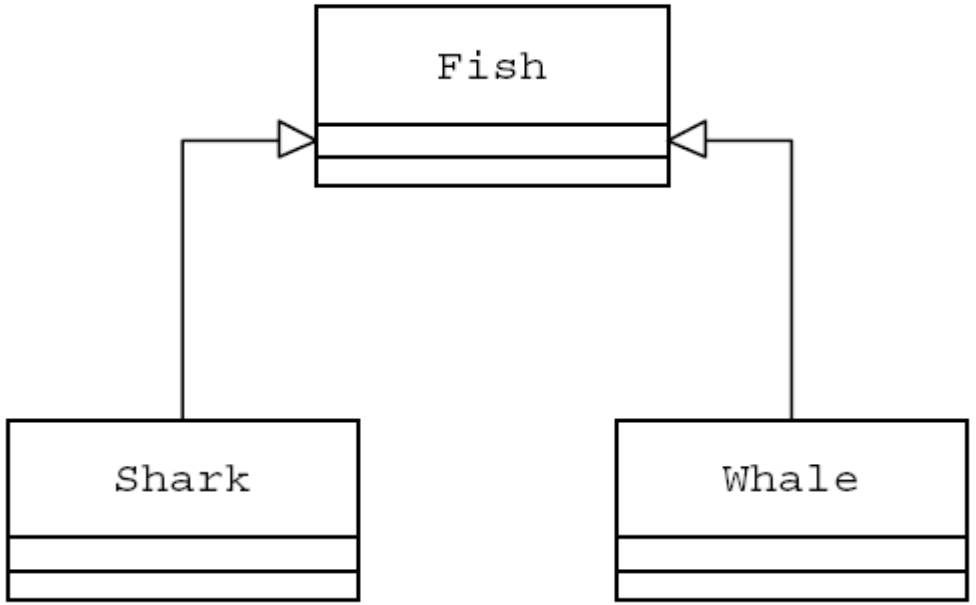
Một mũi tên với nhãn “throws” được sử dụng để chỉ ra một phương thức cụ thể ném một ngoại lệ. Mũi tên chỉ từ phương thức đến lớp ngoại lệ. Cả hai phương thức isValid và save khai báo để ném một ngoại lệ của kiểu java.rmi.RemoteException



Generalization

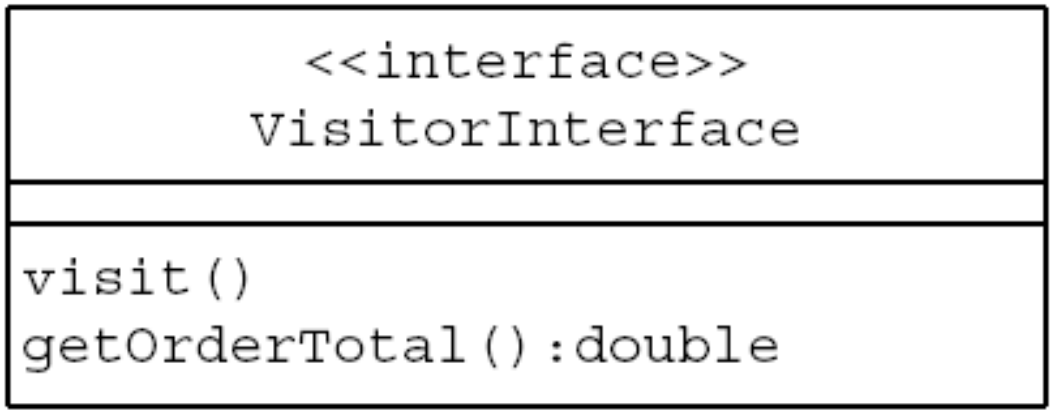
Khái quát hóa (Generalization) được sử dụng để miêu tả khái niệm kế thừa hướng đối tượng khi có một lớp cơ sở với hành vi chung và mỗi lớp được dẫn xuất của nó chứa hành vi/các chi tiết cụ thể.

Một mũi tên rỗng, đóng rỏ từ lớp con Shark/Whale đến lớp con Fish biểu diễn khái quát hóa.



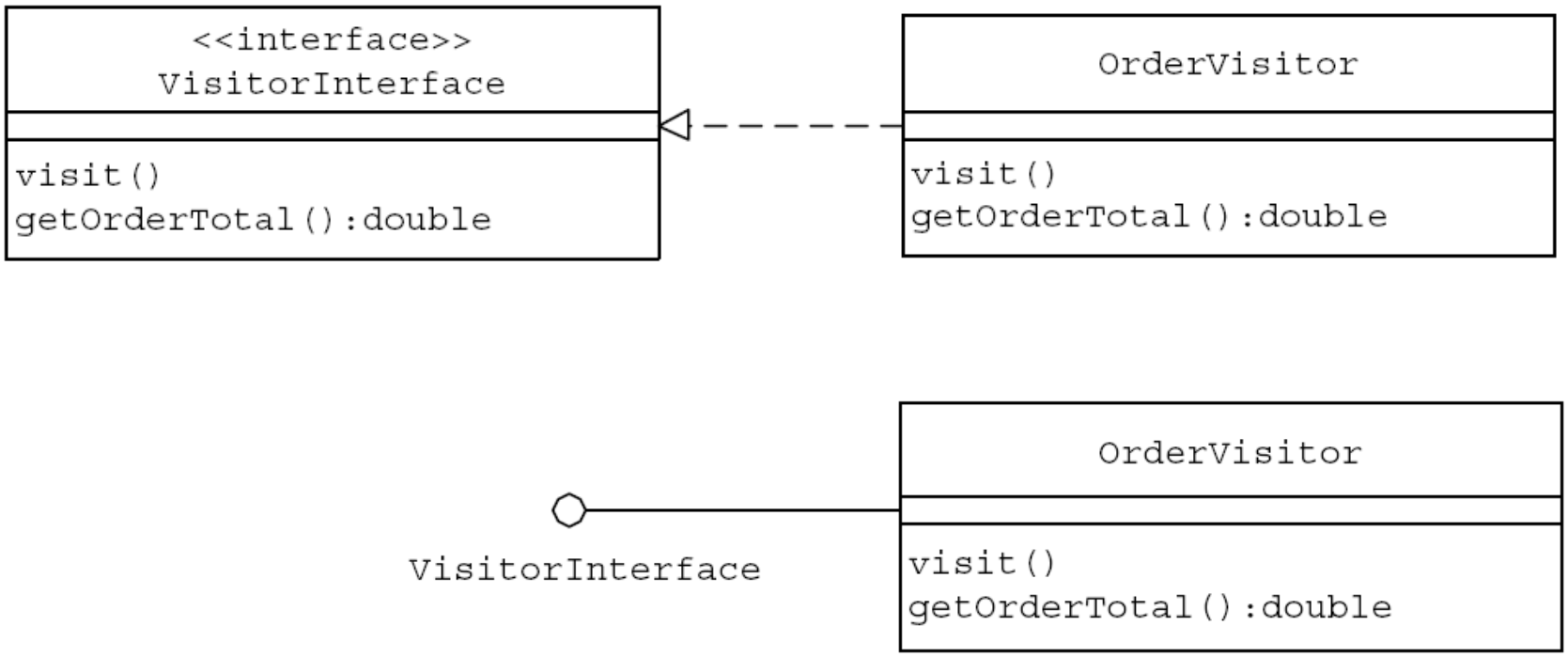
Interface

Một interface xác định các hoạt động có thể nhìn thấy từ bên ngoài của một lớp. Một interface thường chỉ xác định một phần hành vi của một lớp hiện thực thực tế. Một interface có thể được vẽ sử dụng hình chữ nhật giống class, với tên interface bên dưới "interface".



Realization

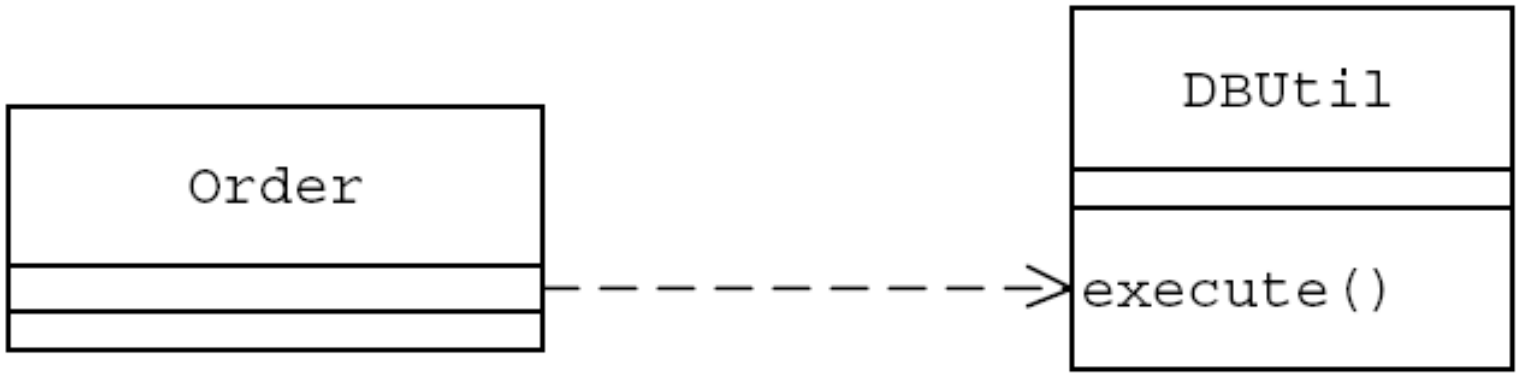
Một hiện thực hóa mô tả mối quan hệ giữa interface và class nó cung cấp một hiện thực thực tế. ”.



Dependency

- Sự phụ thuộc mô tả mối quan hệ giữa thành phần source và target, khi có một mối quan hệ phụ thuộc giữa chúng. Có nghĩa là khi thay đổi trong target, phần tử source phải trải qua việc thay đổi cần thiết nhưng không ngược lại.

- Lớp Order sử dụng phương thức execute của lớp DBUtil để thực thi câu lệnh SQL và do đó Order phụ thuộc vào DBUtil.



Class Association

- **Tính đa dạng (Multiplicity)**

  Tính đa dạng được sử dụng để chỉ định số lượng thể hiện của một class được liên kết đến một thể hiện của class khác.

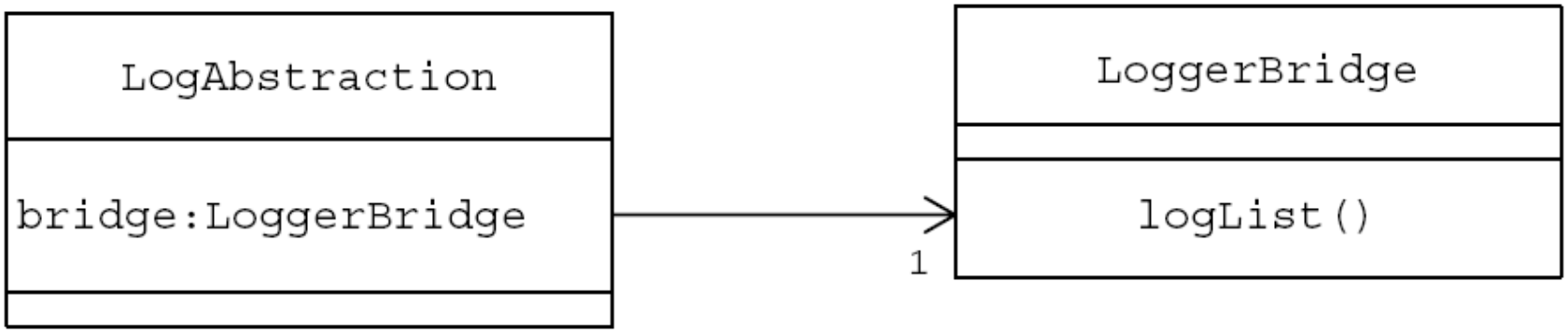
Notation	*Description*
1	No More than One
0..1	Zero or One
*	Many
0..*	Zero or Many
1..*	One or Many

Class Association

- **Khả năng điều hướng (Navigability)**

  Khi Class A chứa thông tin cần thiết để tiếp cận Class B, thì có thể điều hướng là từ Class A đến Class B. Nói cách khác, Class A biết về Class B, nhưng không ngược lại.

  Một thể hiện của lớp LogAbstraction duy trì nội bộ một đối tượng LoggerBridge và sẽ có thể tiếp cận trực tiếp. Do đó một đối tượng LoggerBridge có thể điều hướng từ một thể hiện LogAbstraction.



Class Association

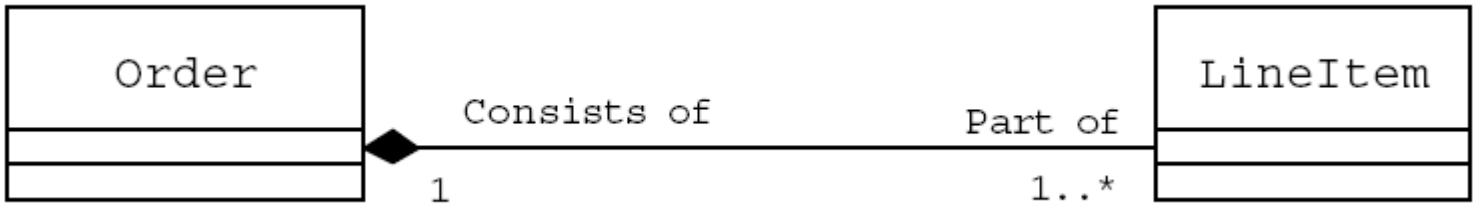
- **Composition**

  Class A chứa Class B.

  Điều này cho thấy mối quan hệ mạnh mẽ giữa Class A và Class B,

  Một dòng item là một phần của Order

  Một dòng item không thể tồn tại mà không có Order.

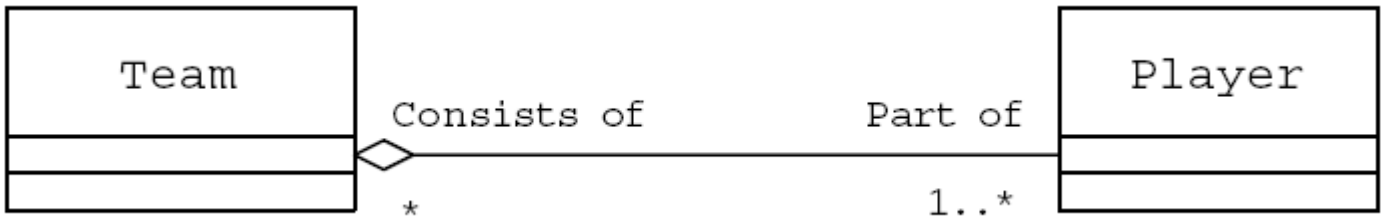


Class Association

- **Aggregation**

  Đây là một dạng composition nhẹ hơn. Cả class đóng vai trò quan trọng hơn thành phần class, nhưng không giống trường hợp composition, class thành phần có thể tồn tại một mình mà không cần cả hai.

  Một Player là một phần của Team.
  Một Player có thể theo gia nhiều Team, do đó khi Team giải thể, Player vẫn còn



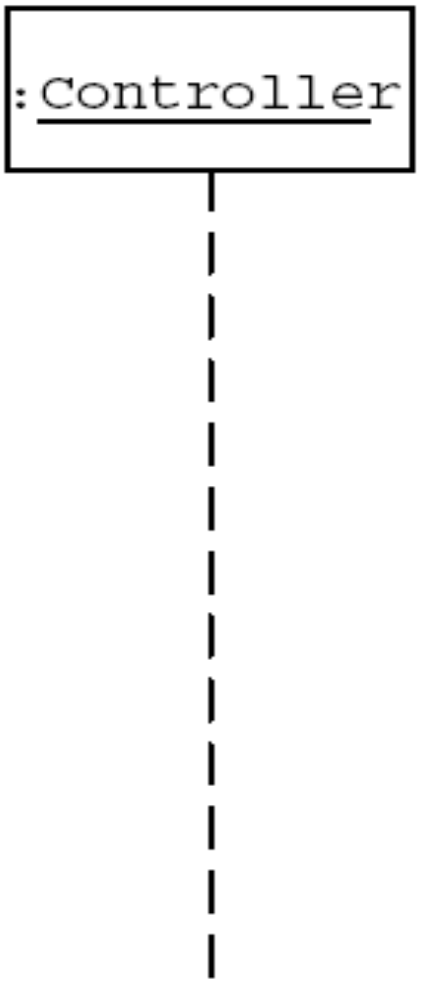
Sequence Diagrams

Sequence diagrams được sử dụng để miêu tả những tương tác giữa các đối tượng cộng tác về mặt message được chuyển đổi theo thời gian cho kết quả cụ thể.

Ngoài ra, một sequence diagrams cũng có thể được sử dụng để mô hình hóa các luồng nghiệp vụ.

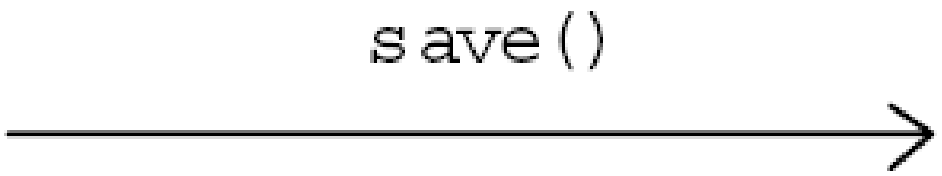
Object

Một object được biểu diễn với tên của lớp trong hình chữ nhật đứng trước bởi dấu **:**



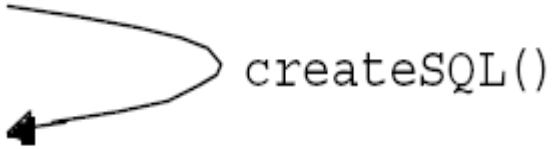
Message

Một message là 1 giao tiếp giữa objects. Một đường nằm ngang liền mạch cho thấy 1 message có thể được gán nhãn với tên message/operation cùng với các giá trị đối số của nó

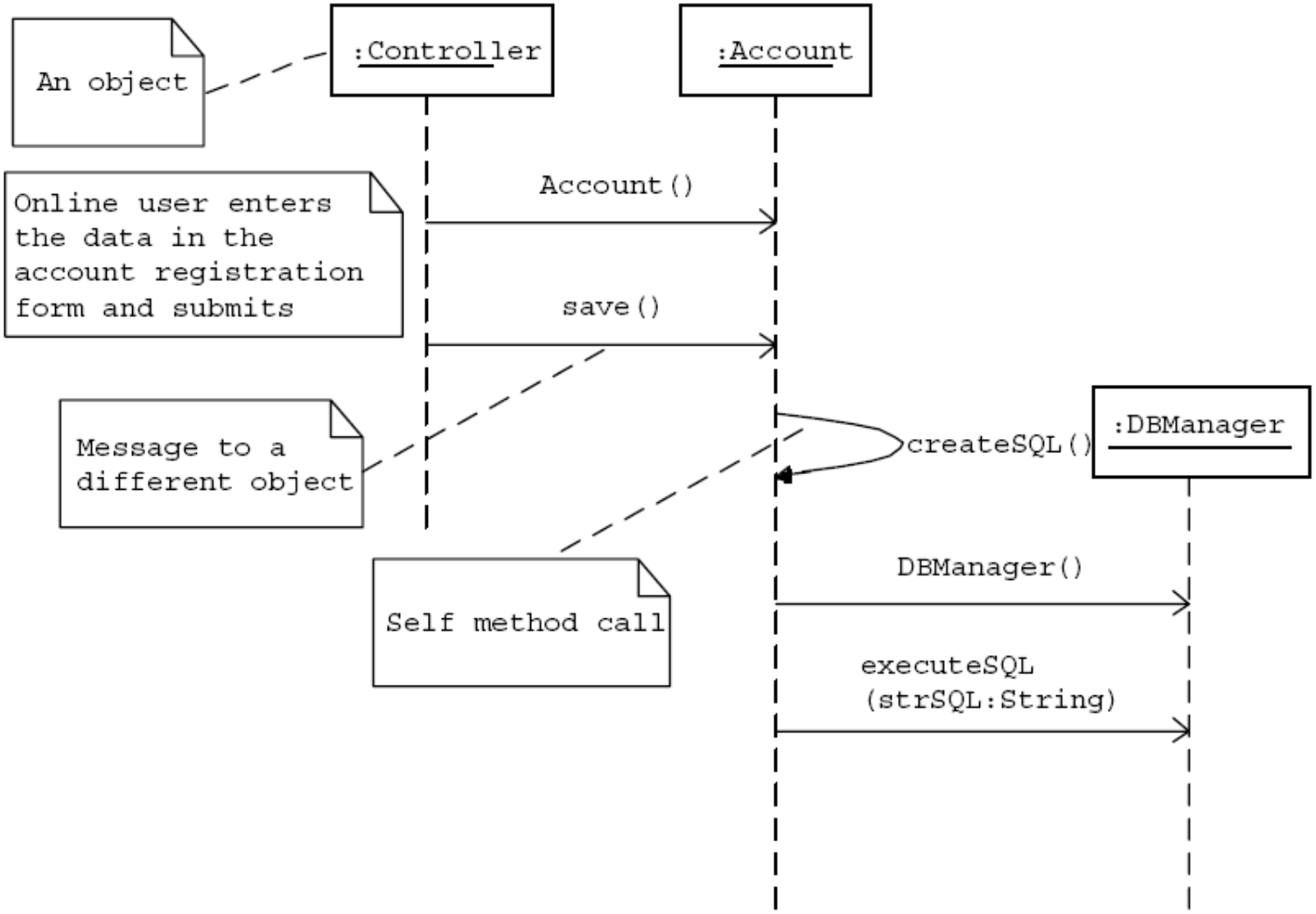


Self Call

Đây là cuộc gọi 1 message từ một đối tượng vào chính nó.



Ví dụ: sequence diagram



Creational Patterns

1. Abstract Factory
2. Builder
3. Factory Method
4. Prototype
5. Singleton

Singleton Pattern

- Singleton là một trong những mẫu thiết kế đơn giản nhất trong Java, nó cung cấp một trong các hướng để tạo một object.

- Mẫu này liên quan đến một mẫu duy nhất, có trách nhiệm tạo object trong khi đảm bảo rằng chỉ một object duy nhất được tạo ra. Class này cung cấp một hướng để truy cập object duy nhất của nó, nó có thể được truy cập trực tiếp mà không cần thuyết minh object của class.

Implementtation

Factory Pattern

- Factory pattern là một trong những mẫu thiết kế được sử dụng nhiều nhất trong Java, nó cung cấp các hướng tốt nhất để tạo một object.

- Trong mẫu Factory, chúng ta tạo object mà không cho biết logic tạo đến client và xem object được tạo mới sử dụng interface chung.

Implementtaion

Abstract Factory Pattern

- Các mẫu Abstract Factory làm việc xung quanh một super-factory, nó tạo ra các factory khác. Factory này cũng được gọi là factory của những factory.

- Trong mẫu abstract factory, một interface chịu trách nhiệm tạo ra một factory của những đối tượng có liên quan mà không chỉ rõ ràng các class của chúng. Mỗi factory được sinh ra có thể cung cấp các đối tượng theo mẫu Factory.

Implementation

Builder Pattern

- Builder pattern xây dựng một object phức tạp sử dụng objects đơn giản và sử dụng hướng tiếp cận từng bước.

Implementation

Prototype Pattern

- Mẫu prototype đề cập đến tạo object trùng nhau trong khi vẫn giữ hiệu suất.
- Mẫu này hiện thực một prototype interface, để tạo một bản sao của đối tượng hiện tại. Mẫu này được sử dụng khi việc tạo trực tiếp đối tượng là tốn kém.
- Ví dụ, một object được tạo ra sau một hoạt động CSDL tốn kém. Chúng ta có thể cache object, trả về bản sao của nó cho yêu cầu tiếp theo và cập nhật CSDL khi nào cần giảm những cuộc gọi CSDL.

Implementation

Structure Patterns

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Facade
6. Flyweight
7. Proxy

Adapter Pattern

- Mẫu Adapter làm việc như một cầu nối giữa hai interface không tương thích, mẫu này tích hợp khả năng của hai interface độc lập.
- Mẫu này liên quan đến một lớp duy nhất chịu trách nhiệm kết nối các chức năng của các interface độc lập và không tương thích.
- Ví dụ, đầu đọc thẻ hoạt động như một adapter giữa thẻ nhớ và laptop.

Implementation

Bridge Pattern

- Bridge được sử dụng khi cần tách riêng ra một sự trừu tượng khỏi hiện thực của nó để cả hai có thể thay đổi độc lập, mẫu này tách lớp hiện thực và lớp trừu tượng bằng cách cung cấp một cấu trúc cầu nối giữa chúng.

- Mẫu này liên quan đến một interface hoạt động như một cầu nối, nó làm cho tính năng của các lớp độc lập từ các lớp hiện thực interface. Cả hai kiểu lớp có thể thay đổi cấu trúc mà không ảnh hưởng nhau.

Implementation

Composite Pattern

- Mẫu composite được sử dụng khi cần xử lý một nhóm đối tượng theo cách tương tự như một đối tượng duy nhất. Mẫu composite tạo ra các đối tượng theo cấu trúc cây để biểu diễn một phần cũng như toàn bộ hệ thống phân cấp.

Implementation

Decorator Pattern

- Mẫu decorator cho phép người dùng thêm chức năng mới đến một đối tượng có sẵn mà không thay đổi cấu trúc của nó, hoạt như một lớp bao bọc cho lớp hiện có.
- Mẫu này tạo ra một lớp decorator bọc lớp bản gốc và cung cấp thêm chức năng giữa cho các phương thức lớp nguyên vẹn.

Implementation

Facade Pattern

- Mẫu Facade che giấu sự phức tạp của hệ thống và cung cấp một interface để client sử dụng có thể truy cập hệ thống, mẫu này thêm một interface vào hệ thống hiện có để che dấu sự phức tập của nó.

- Mẫu này liên quan đến một lớp duy nhất cung cấp các phương thức đơn giản hóa theo yêu cầu bởi client và các cuộc gọi ủy quyền đến các phương thức của các lớp hệ thống hiện có.

Implementation

Flyweight Pattern

- Mẫu Flyweight chủ yếu được sử dụng để giảm số lượng các đối tượng đã tạo và để giảm khối bộ nhớ và tăng hiệu quả.
- Mẫu này có gắng sử dụng lại các đối tượng loại tương tự hiện có sẵn sàng bằng cách lưu trữ chúng và tạo đối tượng mới khi không tìm thấy đối tượng phù hợp.

Implementation

Proxy Pattern

- Trong mẫu Proxy, một lớp biểu diễn chức năng của một lớp khác. .

Implementation

Behavioral Patterns

1. **Chain of Responsibility.**
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Template Method
11. Visitor

Chain of Responsibility Pattern

- Mẫu này tạo ra một chuỗi các đối tượng bên nhận (receiver) cho một yêu cầu, nó tách riêng bên gửi và bên nhận yêu cầu dựa trên kiểu yêu cầu.

- Trong mẫu này, thường mỗi bên nhận chứa tham chiếu đến bên nhận khác. Nếu một đối tượng không thể xử lý yêu cầu thì nó được gửi đến bên nhận tiếp theo và cứ tiếp tục như thế.

Implementation

Command Pattern

- Command là mẫu thiết kế hướng dữ liệu, một yêu cầu được gói dưới một đối tượng như dòng lệnh và chuyển qua đối tượng bên khởi xướng. Đối tượng bên khởi xướng tìm kiếm đối tượng phù hợp mà có thể xử lý lệnh này và chuyển lệnh đến đối tượng tương ứng để thực thi.

Implementation

Iterator Pattern

- Iterator là mẫu thiết kế được sử dụng rất phổ biến trong Java và môi trường lập trình .Net. Mẫu này được sử dụng để nhận được cách truy cập các thành phần của một đối tượng kết hợp theo cách tuần tự mà không cần biết việc biểu diễn của nó.

Implementation

Mediator Pattern

- Mediator được sử dụng để giảm sự giao tiếp phức tạp giữa nhiều đối tượng và lớp, nó cung cấp một lớp hòa giải (mediator) thường xử lý tất cả các giao tiếp giữa các lớp khác nhau và hỗ trợ bảo trình dễ dàng code bằng khớp nối lỏng lẻo.

Implementation

Memento Pattern

- Memento được sử dụng để khôi phục trạng thái của một đối tượng về trạng thái trước đó.

Implementation

Observer Pattern

- Observer được sử dụng khi có mối quan hệ 1 – nhiều giữa các đối tượng như nếu một đối tượng được sử đổi, các đối tượng phụ thuốc nó sẽ được thông báo tự động..

Implementation

State Pattern

- Trong mẫu State việc thay đổi hành vi lớp được trên trạng thái của nó.
- Chúng ta tạo ra các đối tượng biểu diễn các trạng thái khác nhau và một đối tượng ngữ cảnh có hành vi thay đổi khi đối tượng tạng thái thay đổi.

Implementation

Strategy Pattern

- Trong mẫu Strategy, một hành vi lớp hoặc thuật toán của nó có thể được thay đổi ở thời gian chạy.
- Chúng ta tạo ra các đối tượng biểu diễn các chiến lược khác nhau và một đối tượng ngữ cảnh có hành vi thay đổi theo đối tượng chiến lược của nó. Đối tượng chiến lược thay đổi thuật toán thực thi của đối tượng ngữ cảnh.

Implementation

Template Pattern

- Trong mẫu Template, một lớp trừu tượng cho thất way/template được xác định để thực thi các phương thức của nó
- Các lớp con có thể ghi đè việc hiện thực phương thức theo nhu cầu nhưng việc gọi phải giống cách được định nghĩa bởi lớp trừu tượng.

Implementation

Visitor Pattern

- Trong mẫu Visitor, chúng ta sử dụng lớp visitor thay đổi thực thi thuật toán của một lớp thành phần. Theo cách này, việc thực thi thuật toán của thành phần có thể khác nhau và khi visitor thay đổi.
- Theo mẫu, đối tượng thành phần phải chấp nhận đối tượng visitor có thể đối tượng visitor xử lý hoạt động trên đối tượng thành phần .

Implementation

Design Pattern Practice

http://www.cse.hcmut.edu.vn/~hiep/DesignPatterns/


Q&A