Lập trình hướng đối tượng

Mẫu thiết kế

Nội dung

- Tổng quan về mẫu thiết kế
- Một số mẫu thiết kế thông dụng
  - Singleton
  - Factory Method
  - Abstract Factory
  - Prototype
  - Adapter

Tổng quan

- Câu trả lời cho câu hỏi "làm thế nào để tôi có thể…" khi xây dựng phần mềm
- MTK là những giải pháp đã được sử dụng trong các phần mềm khác nhau và xem là tốt (best practice)
- Mốc quan trọng: 1995, Gang of Four (GoF) Gamma, Helm, Johnson, và Vlissides; *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

Tổng quan

MTK giải quyết các vấn đề phi chức năng

- Khả năng thay đổi
- Khả năng tương tác
- Khả năng tái sử dụng
- Độ tin cậy

Tổng quan

Có 3 loại MTK

- **Khởi tạo (creational):** liên quan đến khởi tạo đối tượng
- **Cấu trúc (structural):** liên quan đến tổ chức lớp và đối tượng
- **Hành vi (behavioral):** liên quan đến việc gán các chức năng cho lớp

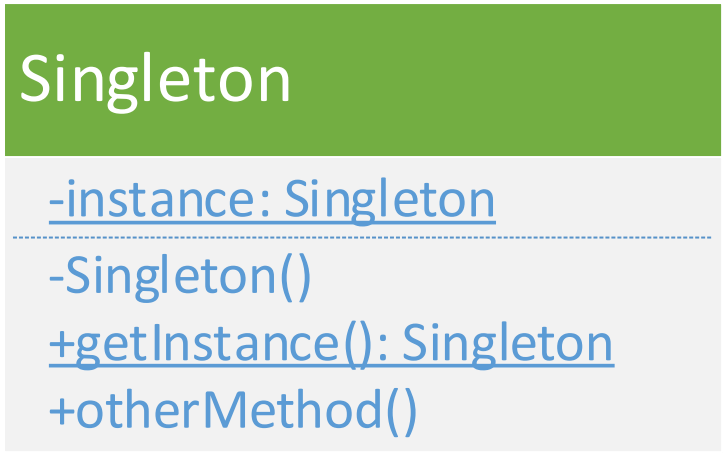
Nội dung chính của một MTK

- Tên: tên của MTL, thường có nghĩa để người dùng dễ hình dung; Ví dụ: Bridge, Mediator, Flyweight
- Ngữ cảnh: ngữ cảnh để áp dụng MTK, ví dụ
- Vấn đề giải quyết: dự định của MTK, mục tiêu (trong điều kiện ràng buộc)
- Giải pháp: các lớp, đối tượng và mối quan hệ giữa các phần tử được đề xuất
- Kết quả: thảo luận về kết quả mang lại của MTK

Singleton

- Ngữ cảnh: Trong một số ứng dụng, việc chỉ có duy nhất một đối tượng (của một lớp đặc biệt nào đó) được tạo ra là rất quan trọng. Ví dụ: kết nối DB, Window manager, file system,…
- Vấn đề: Làm thế nào để chúng ta có thể đảm bảo chỉ duy nhất 1 đối tượng thuộc một lớp nào đó được tạo ra?
- Giải pháp:
  - Lớp có phương thức khởi tạo là private
  - Phương thức getInstance() được sử dụng để tạo đối tượng
  - Trong lần gọi đầu tiên, phương thức getInstance() này sẽ tạo ra một đối tượng
  - Trong những lần gọi tiếp theo, getInstance() không tạo thêm đối tượng mà chỉ trả về đối tượng đã tạo ra

Singleton




```
class Singleton {
        private static Singleton instance = null;
        private Singleton( ) {}
        public static Singleton getInstance( ) {
                if (instance == null)
                        instance = new Singleton();
                return instance;
        }
}
```

Singleton

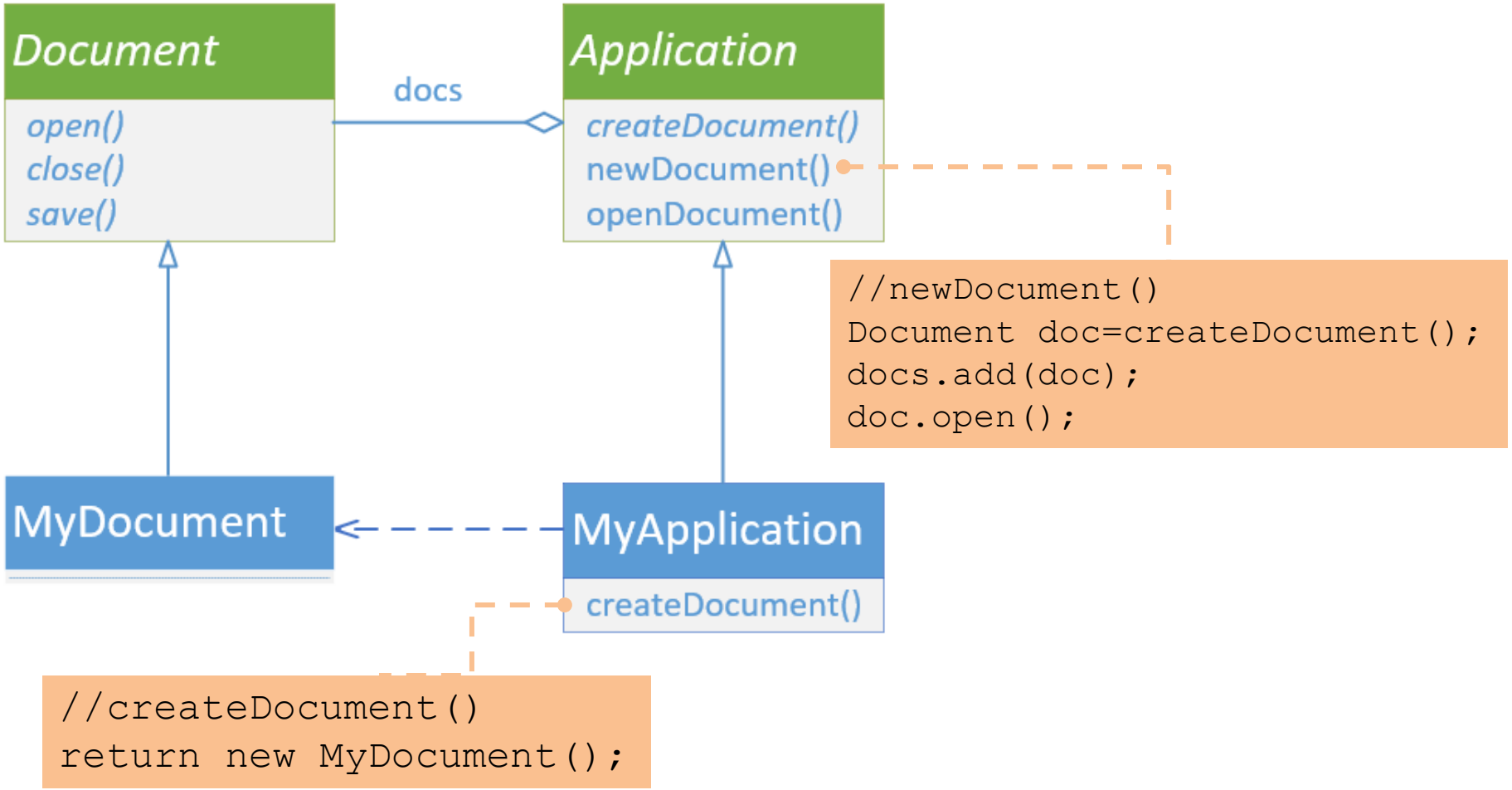
- Thảo luận
  - Mở rộng thành n đối tượng (thay vì một)
  - Nếu một lớp A kế thừa lớp Singleton
  - Dùng thuộc tính và phương thức lớp (static) thay vì dùng Singleton?
  - Vấn đề đa luồng có thể được giải quyết như thế nào?

Factory Method

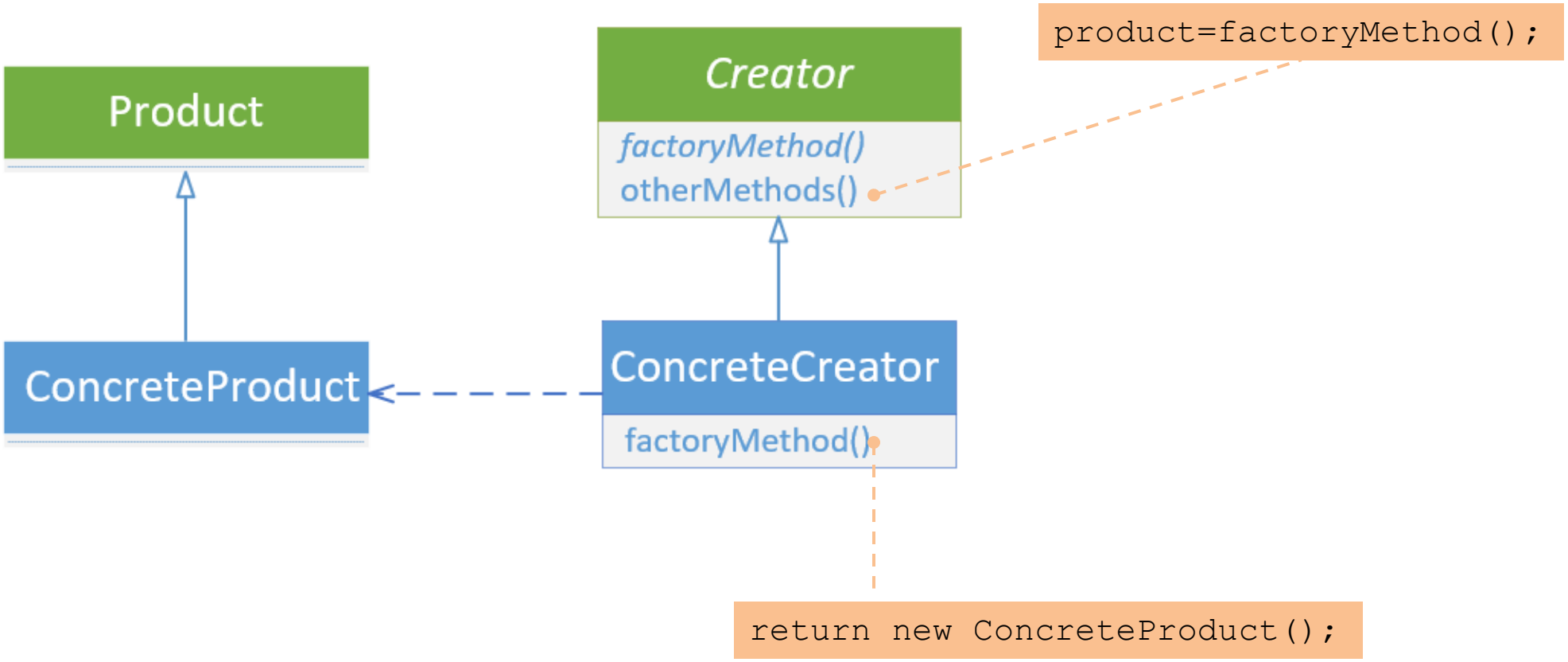
- Ngữ cảnh: Một framework cho ứng dụng đa cửa sổ. Hai phần tử chính trong ứng dụng này là Application và Document (đều là trừu tượng). Người dùng framework phải kế thừa hai lớp này để viết ứng dụng. Application tạo và quản lý Document. Vì Document được kế thừa về sau nên Application không biết được lớp con của Document. Thách thức: Application tạo và quản lý lớp con (chưa biết) của Document

- Phương án: định nghĩa interface để tạo đối tượng nhưng để cho lớp con xác định lớp nào sẽ được sử dụng (để tạo đối tượng)

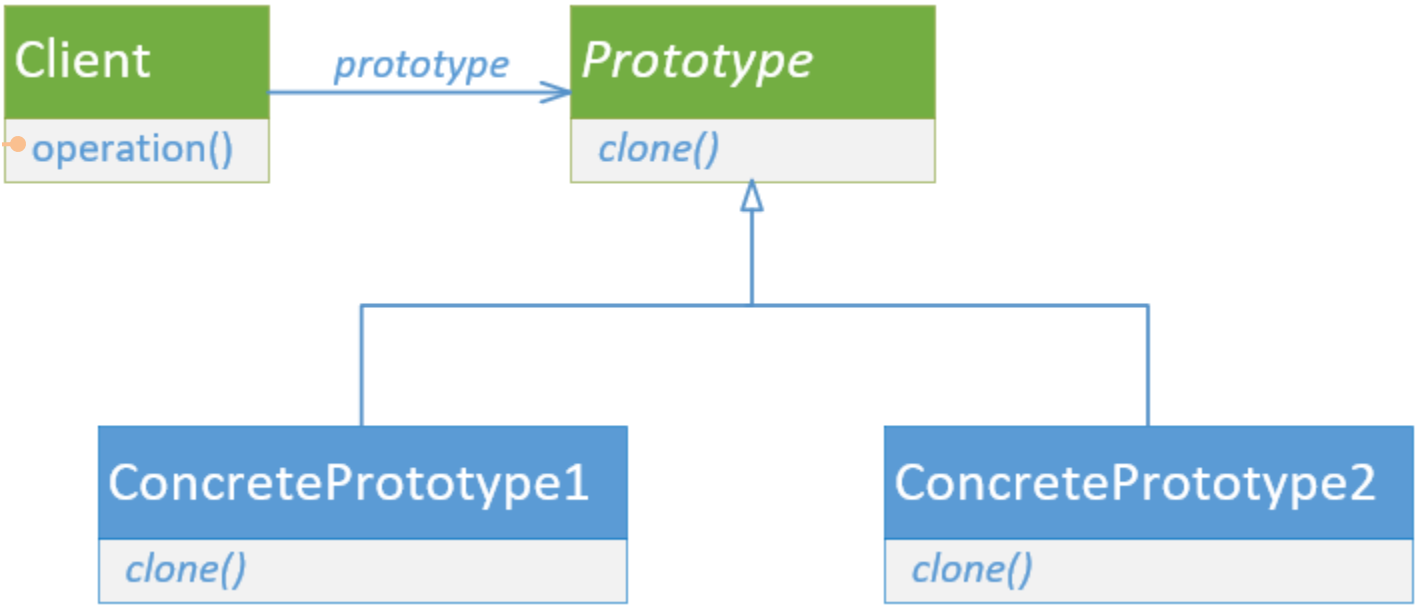
Factory Method



Factory Method



Prototype




```
//operation()
p=prototype.clone();
```

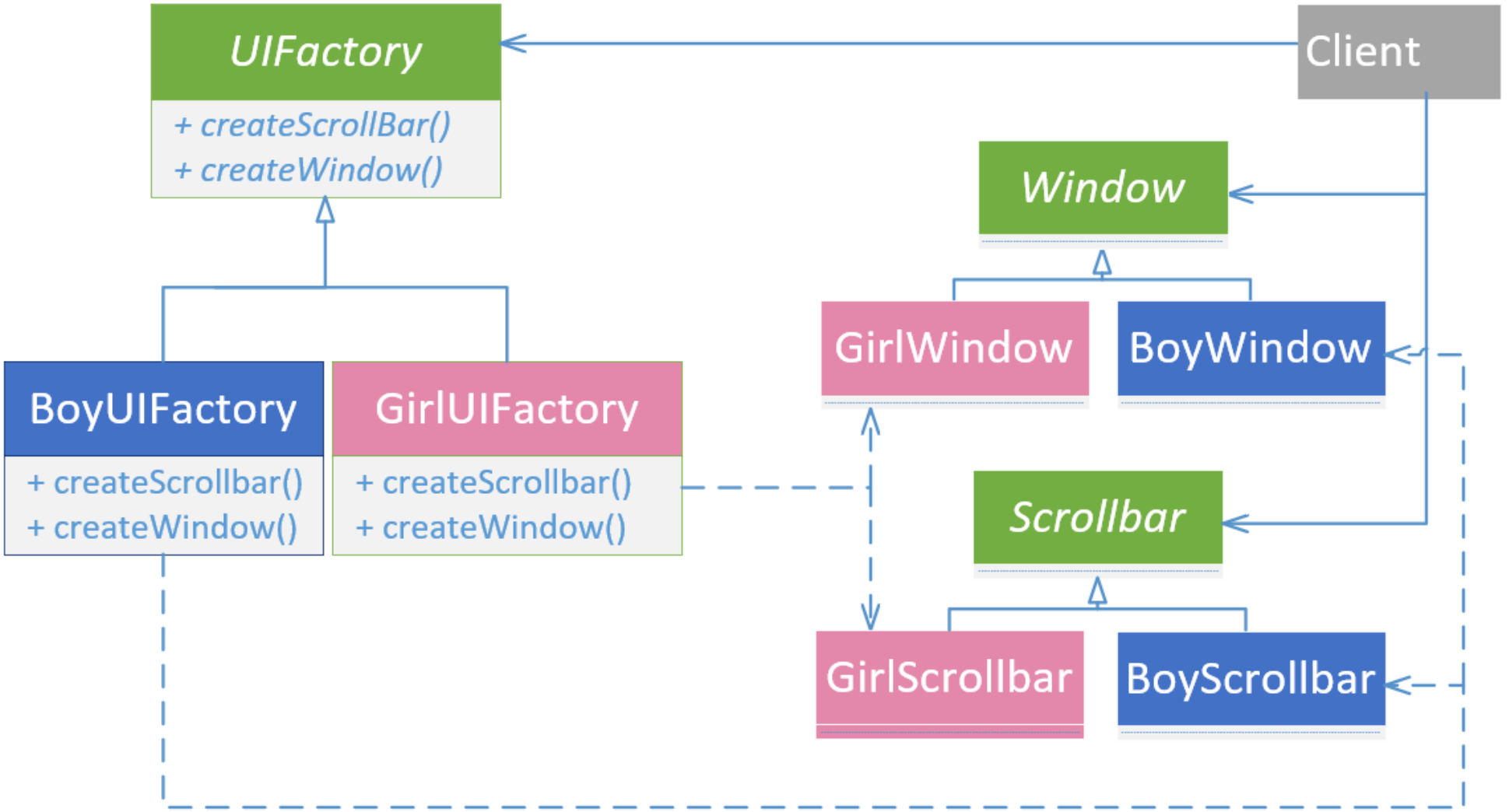
Abstract Factory

- Vấn đề: có nhiều "dòng" đối tượng (product) trong ứng dụng; tại mỗi thời điểm, chỉ một trong các dòng đối tượng đấy được sử dụng

Abstract Factory

- Ngữ cảnh
  - Client độc lập với cách các đối tượng (product) được tạo ra
  - Client được cấu hình với một trong nhiều "dòng" đối tượng
  - Các đối tượng thuộc cùng dòng sẽ được sử dụng cùng nhau

Abstract Factory



Abstract Factory

- Thảo luận
  - Thêm một loại đối tượng (product)
  - Sử dụng Singleton

Adapter

- Để sử dụng framework F, người phát triển ứng dụng phải cung cấp một lớp cài đặt giao diện IMath

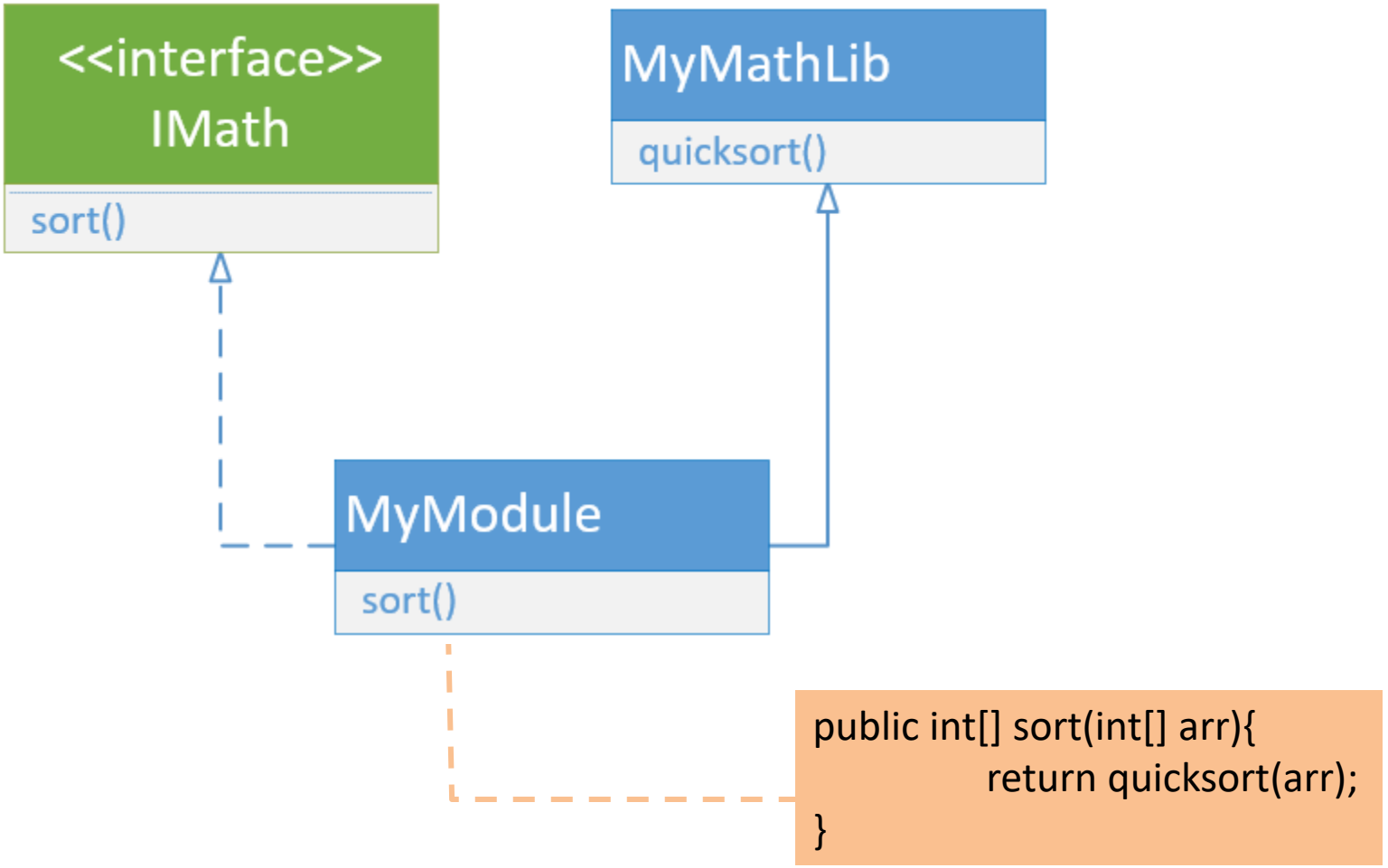
```
IMath{
        int[]   sort(int[] arr)
}
```

- Người phát triển ứng dụng đã download được một thư viện (.class, không chỉnh sửa được) có lớp MyMathLib với phương thức quicksort()

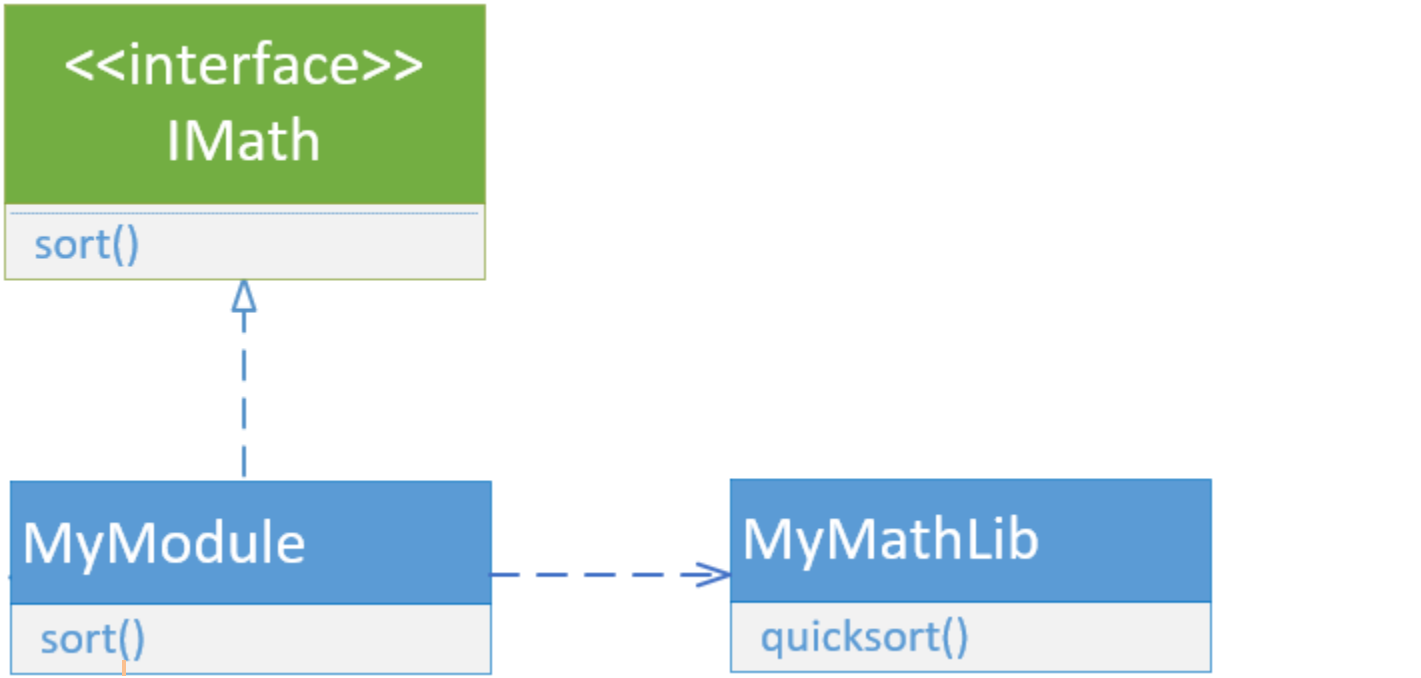
```
public class MyMathLib{
        int[]   quickSort(int[] arr){
        …
        }
}
```

Làm thể nào để sử dụng F với MyMathLib?

Adapter

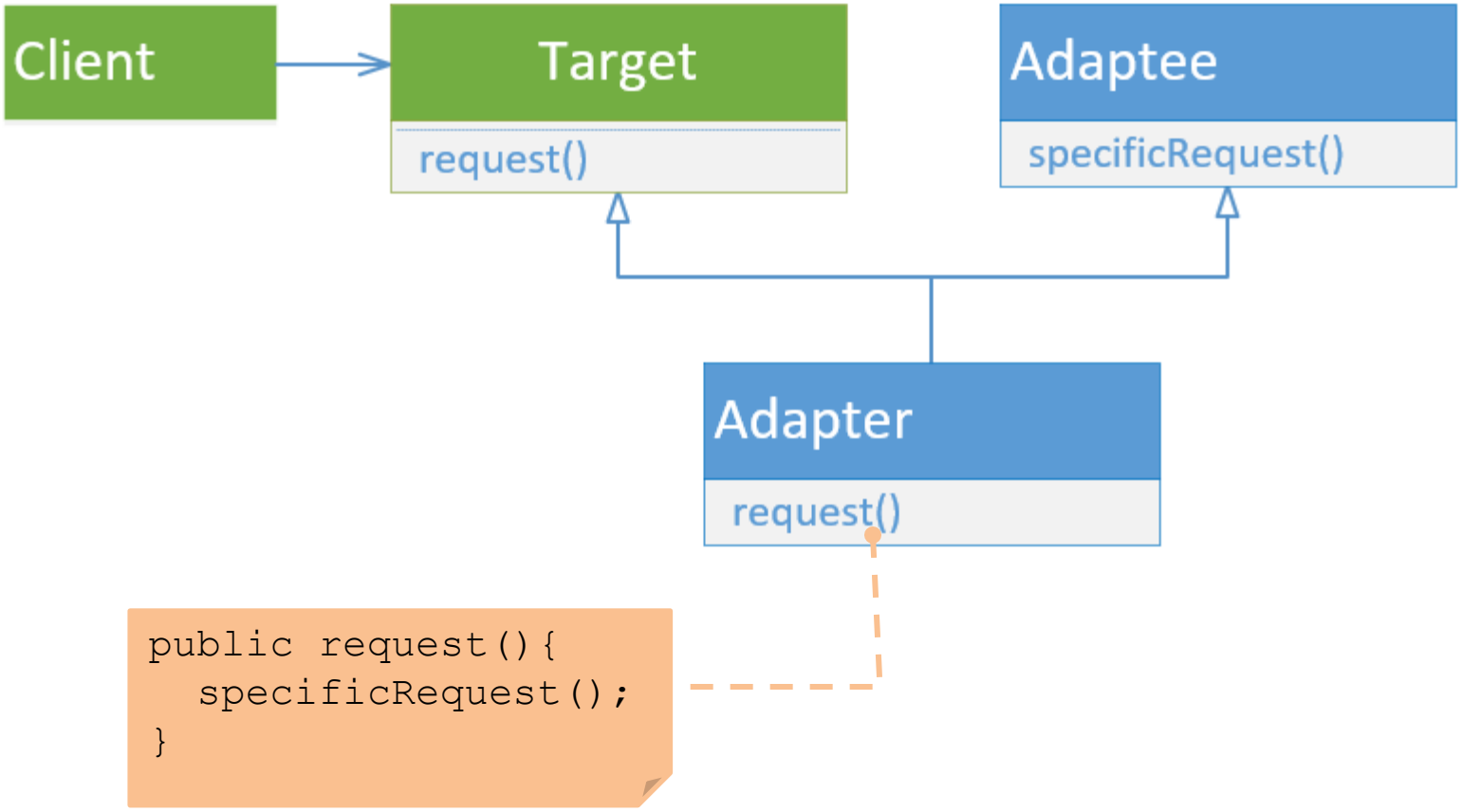

<<interface>>
IMath
sort()
MyMathLib
quicksort()
MyModule
sort()
public int[] sort(int[] arr){
return quicksort(arr);
}

Adapter

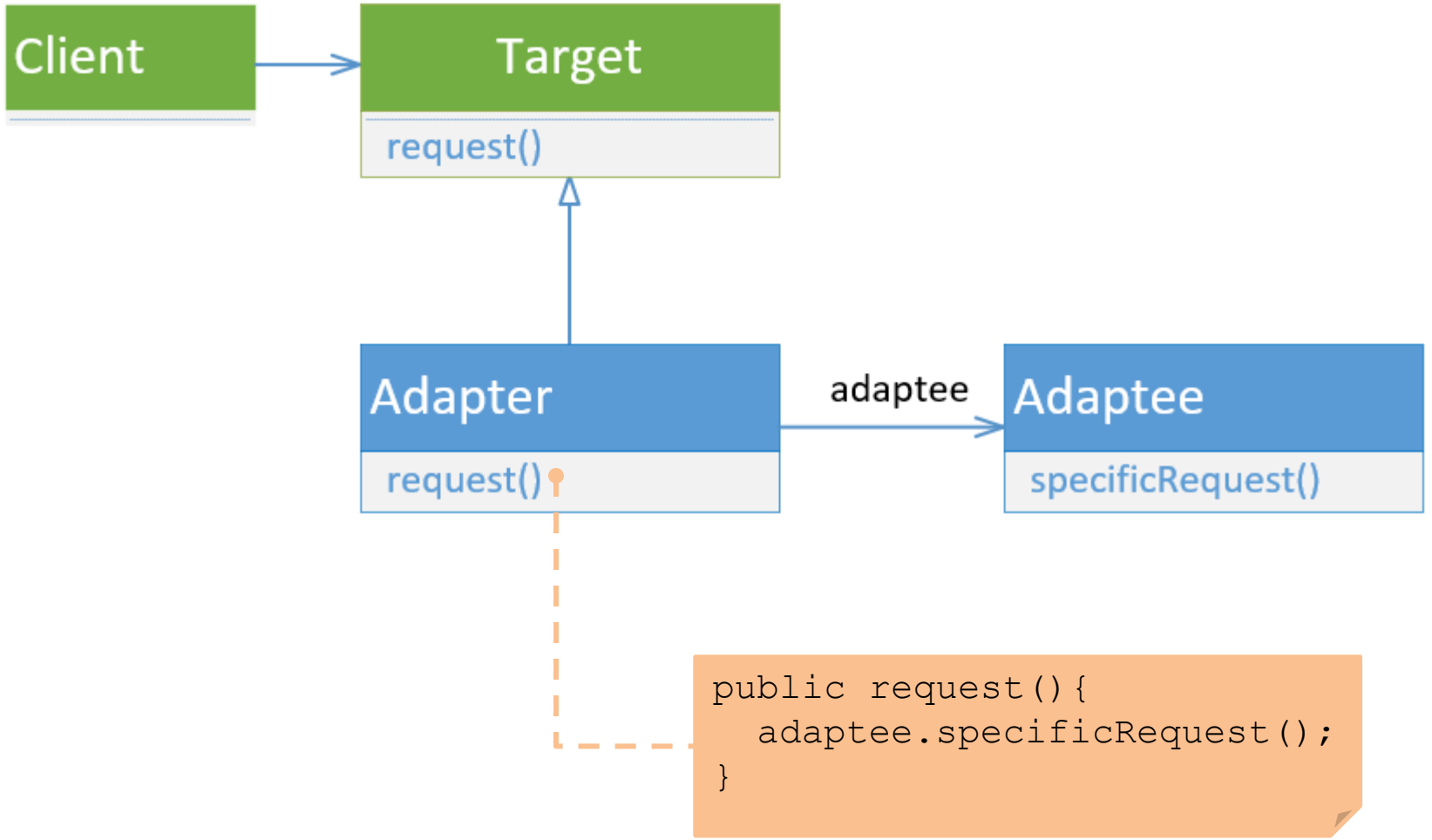



```
public void sort(int[] arr){
        MyMathLib lib=new MyMathLib();
        return lib.quicksort(arr)
}
```

Adapter


Client
Target
request()
Adaptee
specificRequest()
Adapter
request()
public request(){
specificRequest();
}

Adapter


Client
Target
request()
Adapter
request()
adaptee
Adaptee
specificRequest()
public request(){
adaptee.specificRequest();
}

Tổng kết

- Tổng quan về mẫu thiết kế
- Một số mẫu thiết kế thông dụng
  - Singleton
  - Factory Method
  - Abstract Factory
  - Prototype
  - Adapter