# Factory Method Pattern

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

# Design Aspect of Factory Method

Subclass of object that is instantiated

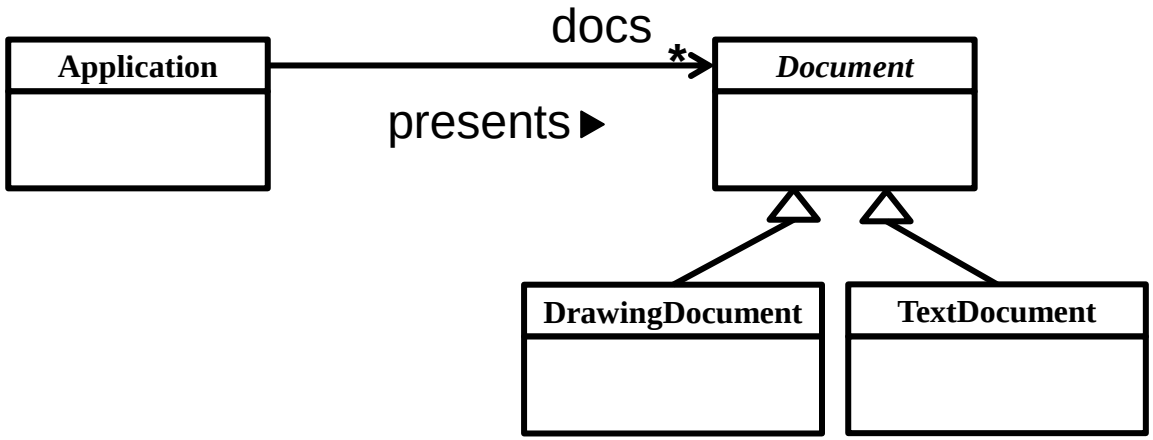
# Outline

- Requirements Statement
- Initial Design and Its Problems
- Design Process
- Refactored Design after Design Process
- Another Example
- Recurrent Problems
- Intent
- Factory Method Pattern Structure
- Static Factory vs. Non-Static Factory
- Homework

# Powerful Document Viewer (Factory Method)

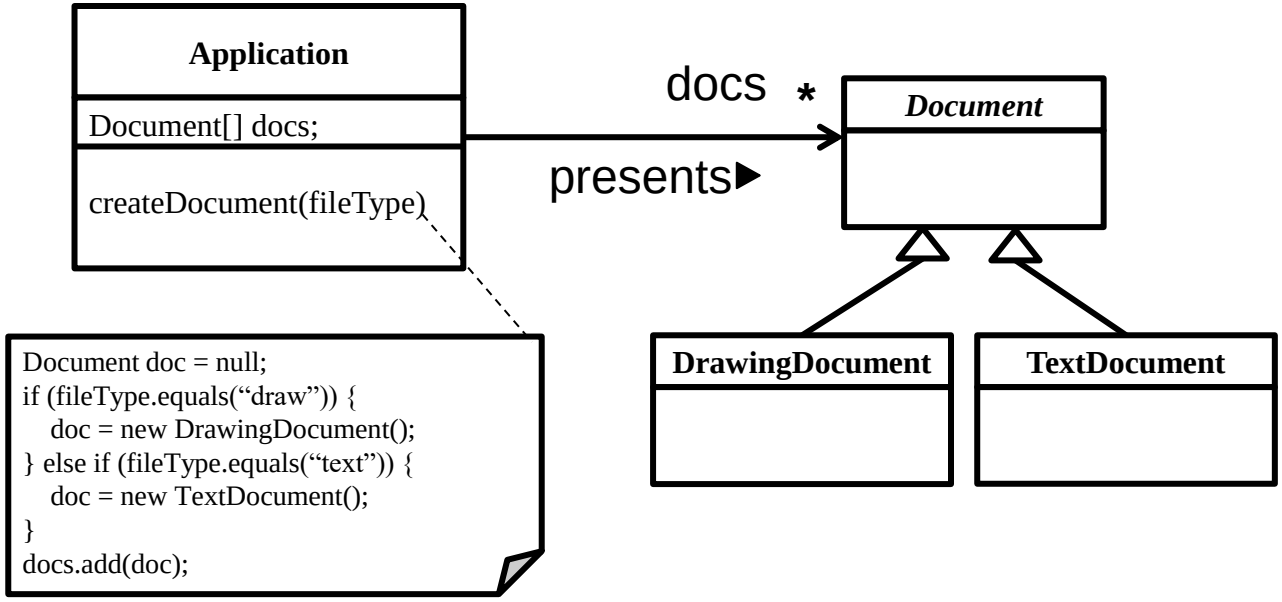
# Requirements Statement$_1$

- A powerful application can present multiple documents at the same time.
- These documents include DrawingDocument, TextDocument, and so on.

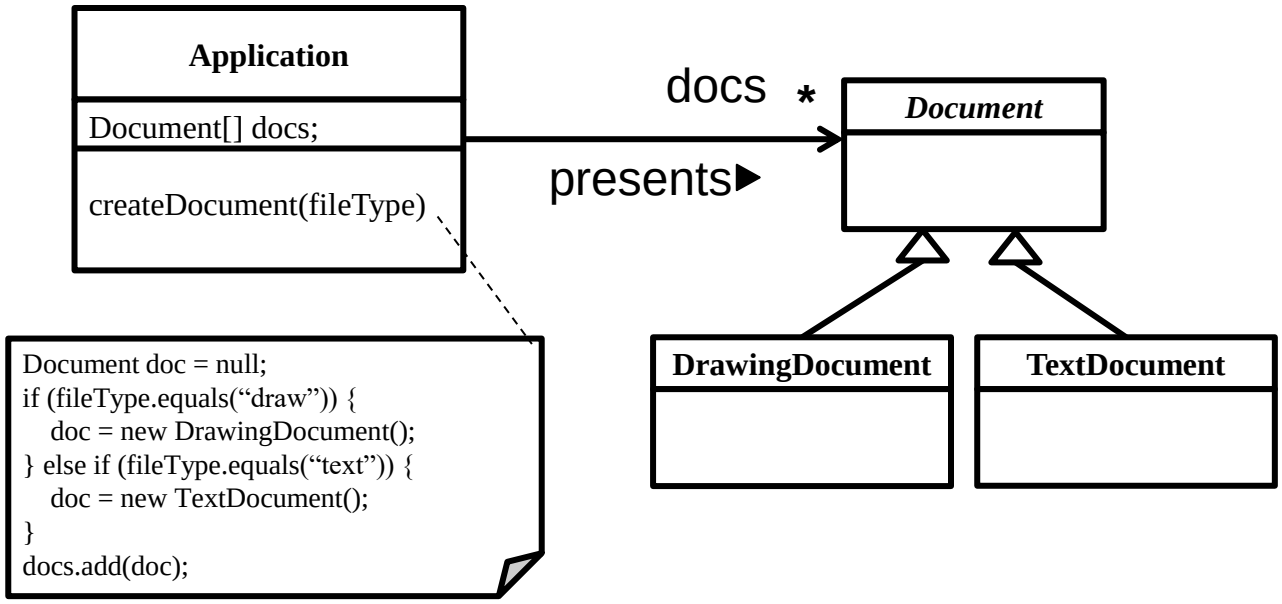


# Requirements Statement$_2$

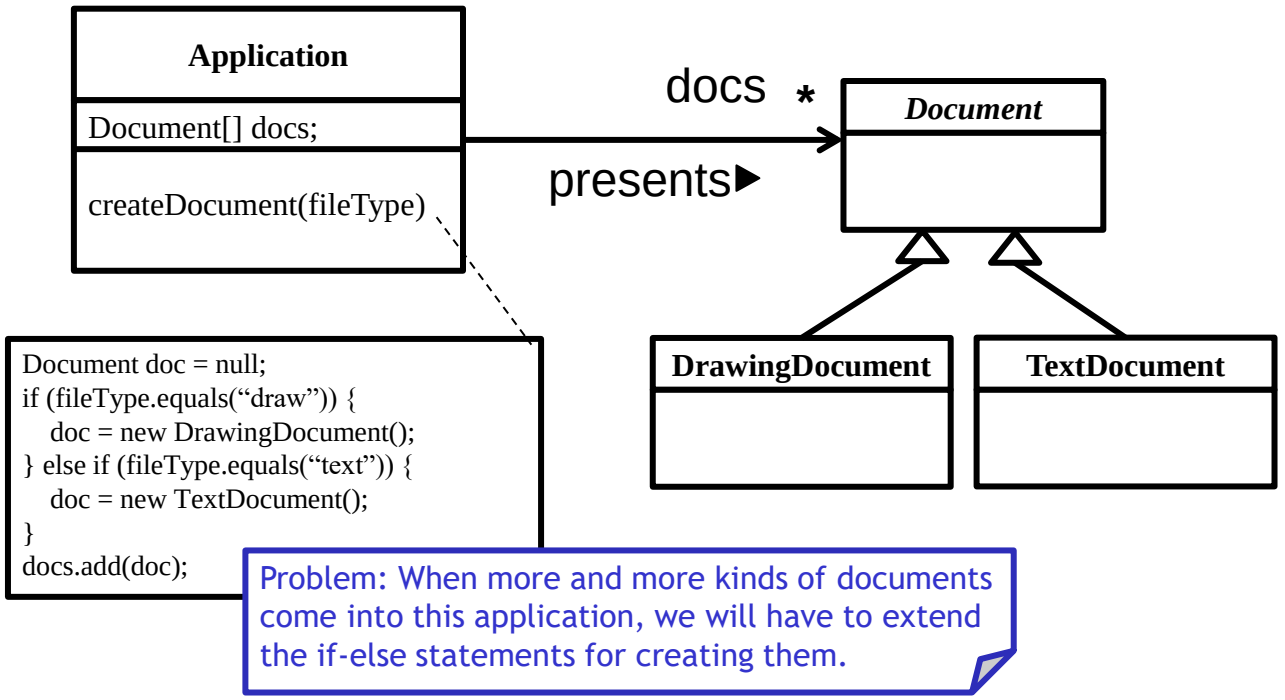
- The application is responsible for managing documents and will create them as required.



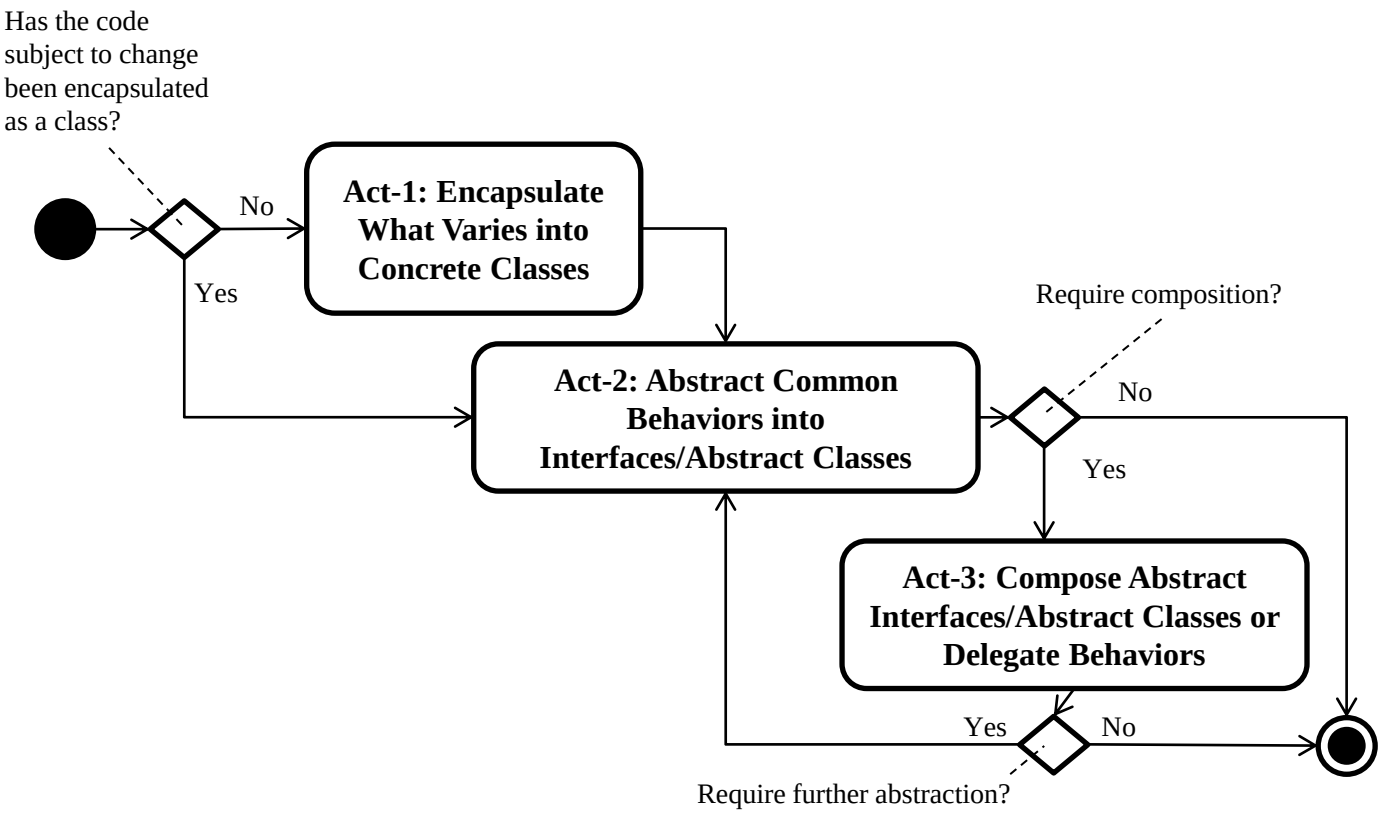
# Initial Design



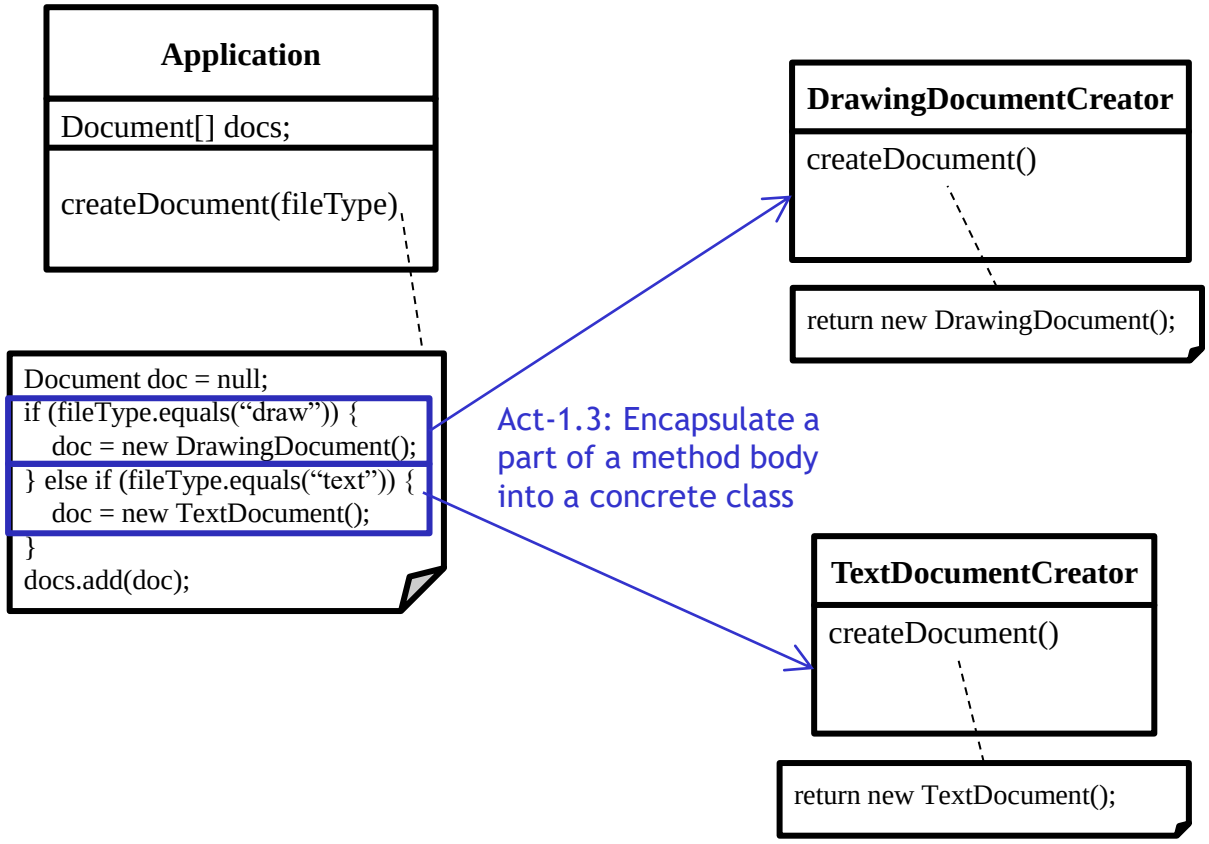
# Problems with Initial Design



# Design Process for Change

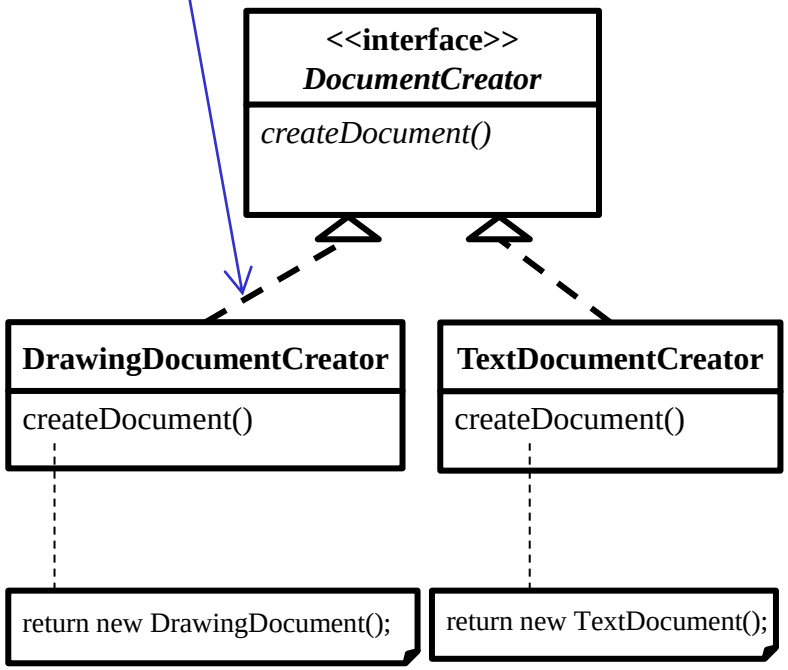


# Act-1: Encapsulate What Varies

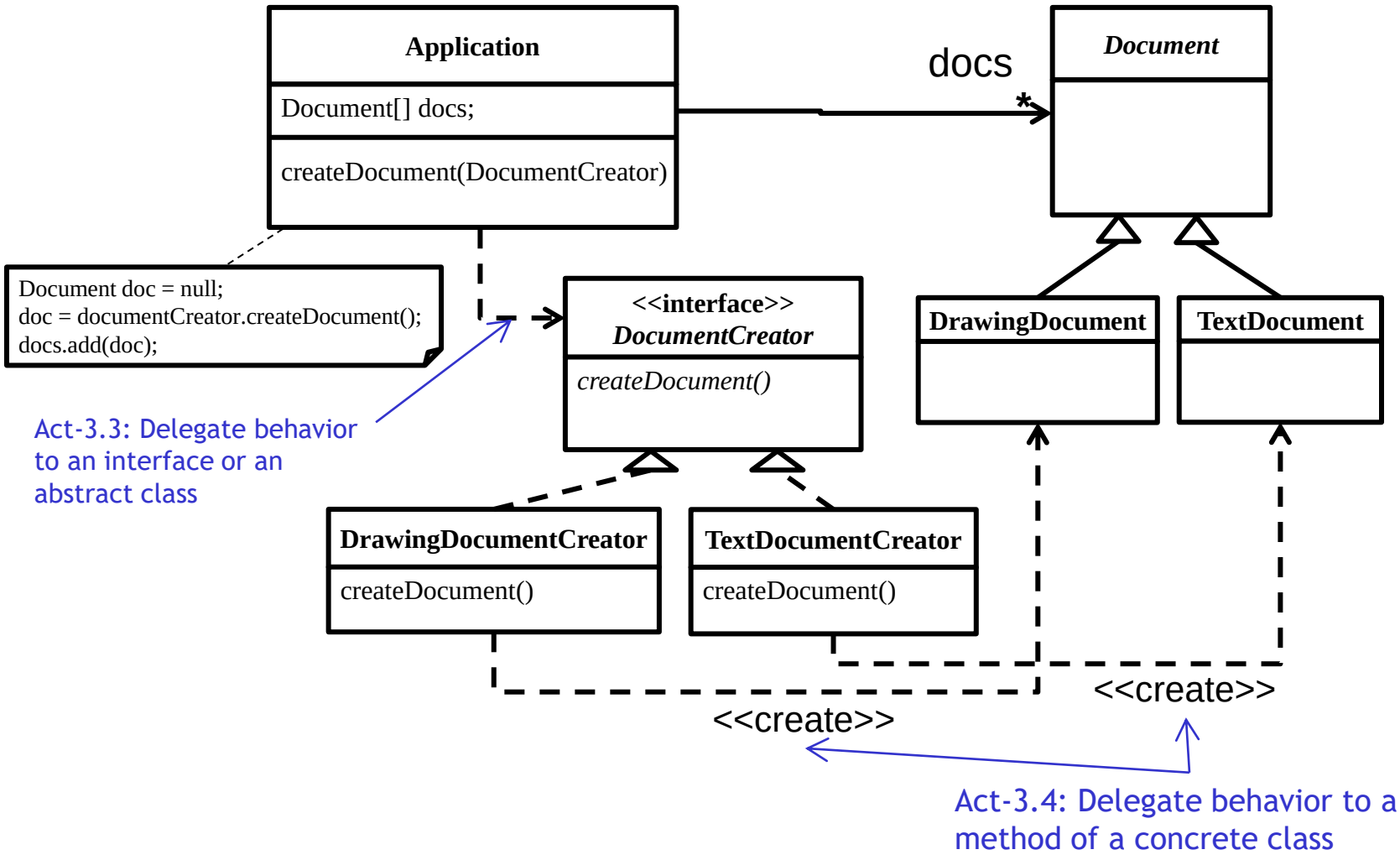


# Act-2: Abstract Common Behaviors

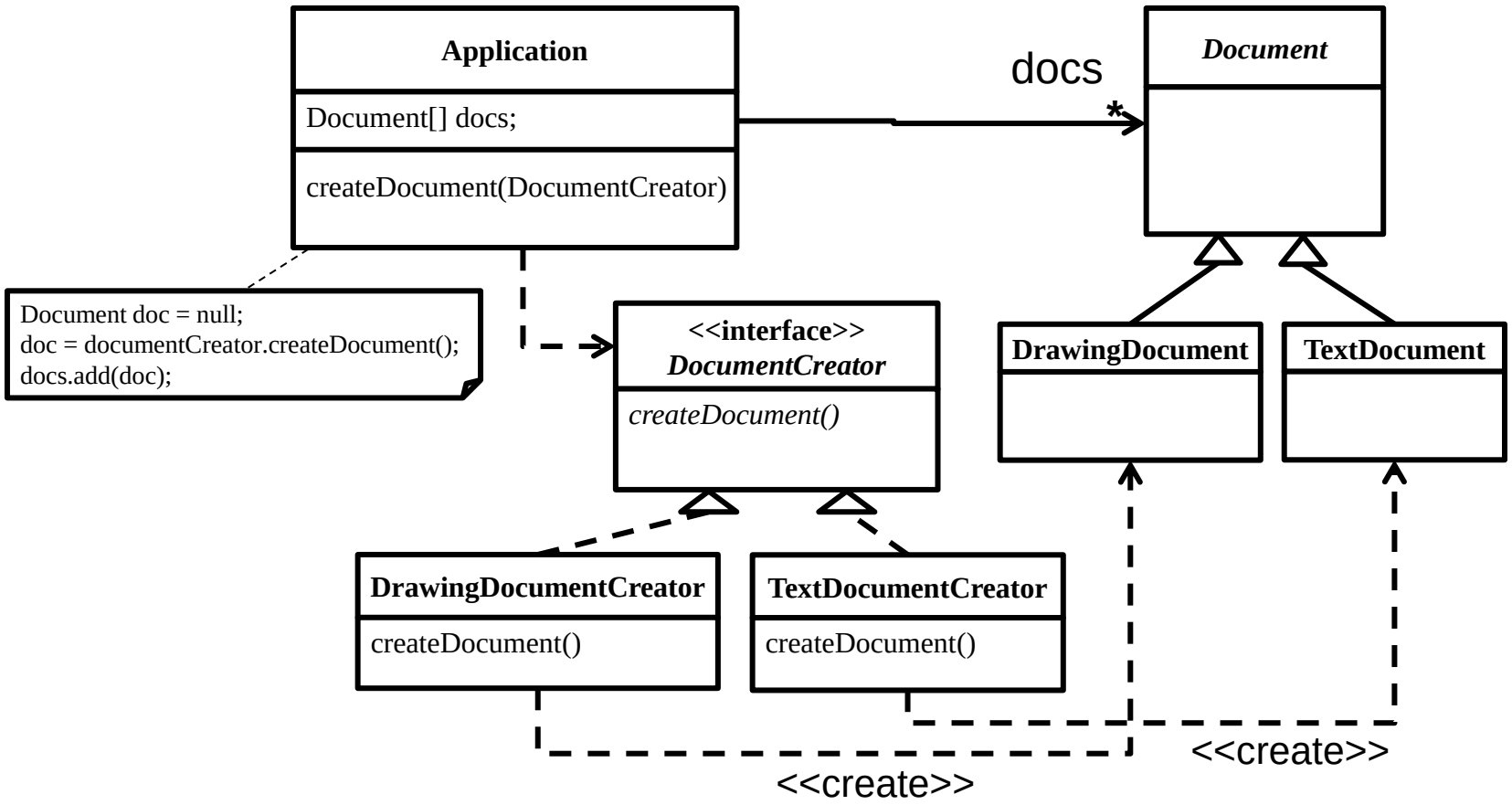
Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



# Act-3: Compose Abstract Behaviors



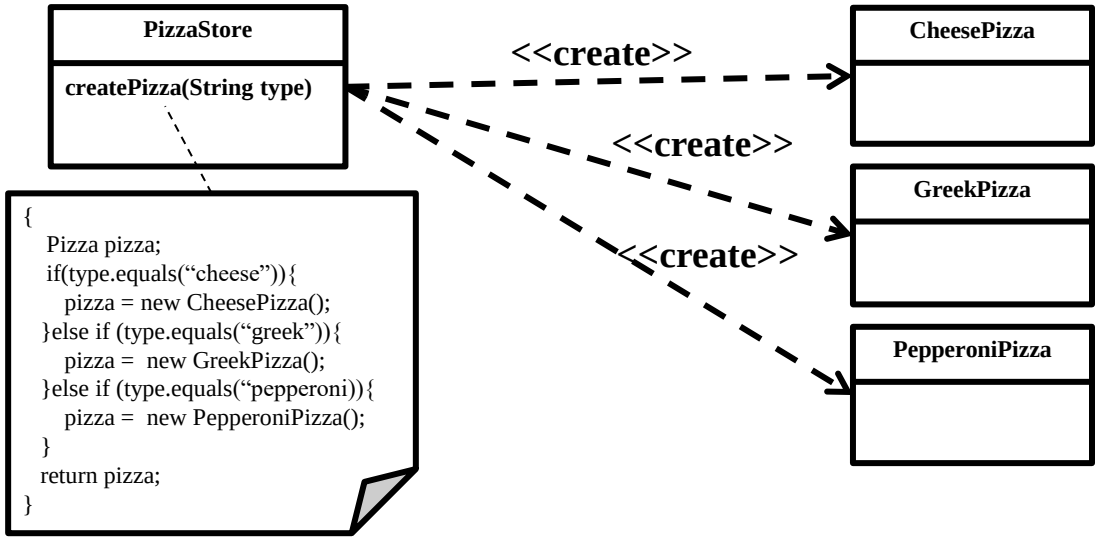
# Refactored Design after Design Process



# Pizza Store (Factory Method)

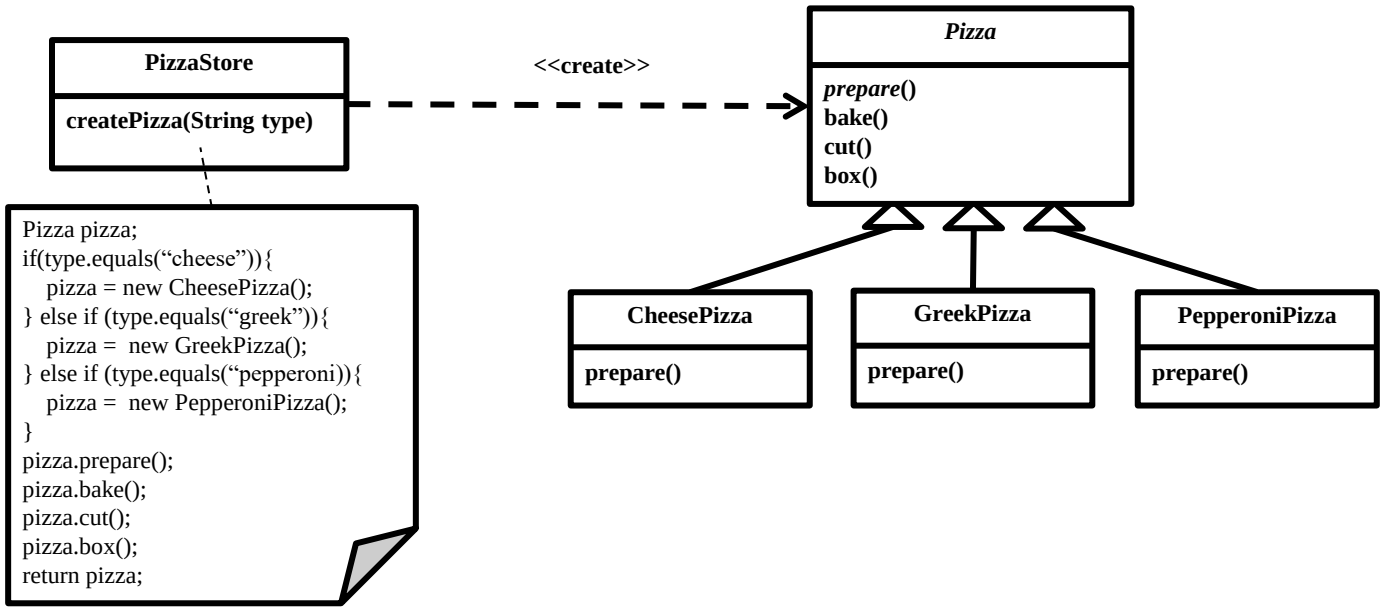
# Requirements Statement$_1$

- Pizza Store
  - The store makes more than one type of pizza: Cheese Pizza, Greek Pizza, and Pepperoni Pizza



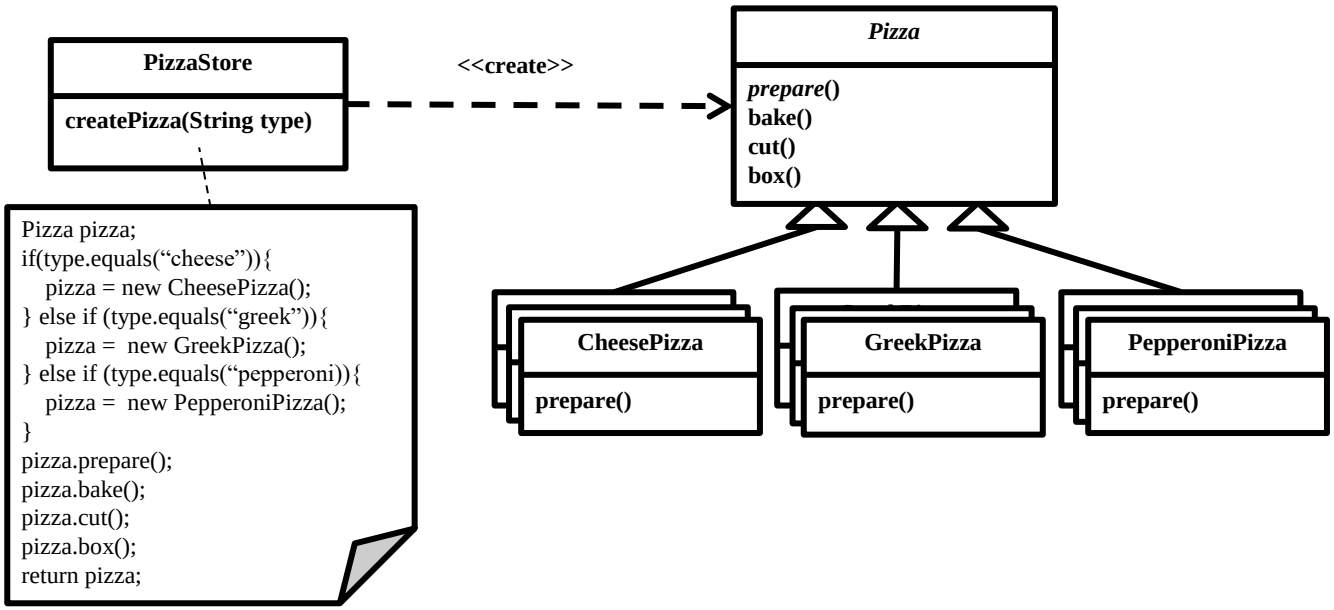
# Requirements Statement$_2$

- Pizza Store
  - Each pizza has different way to prepare, and has the same way to bake, to cut, and to box.

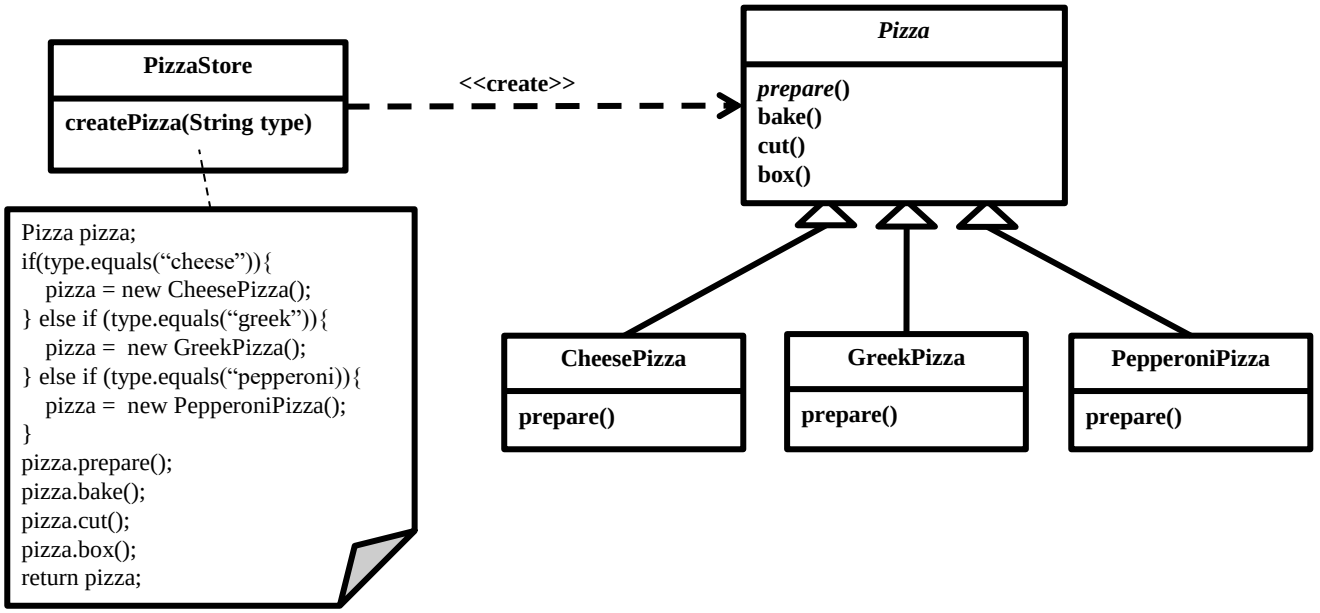


# Requirements Statement$_3$

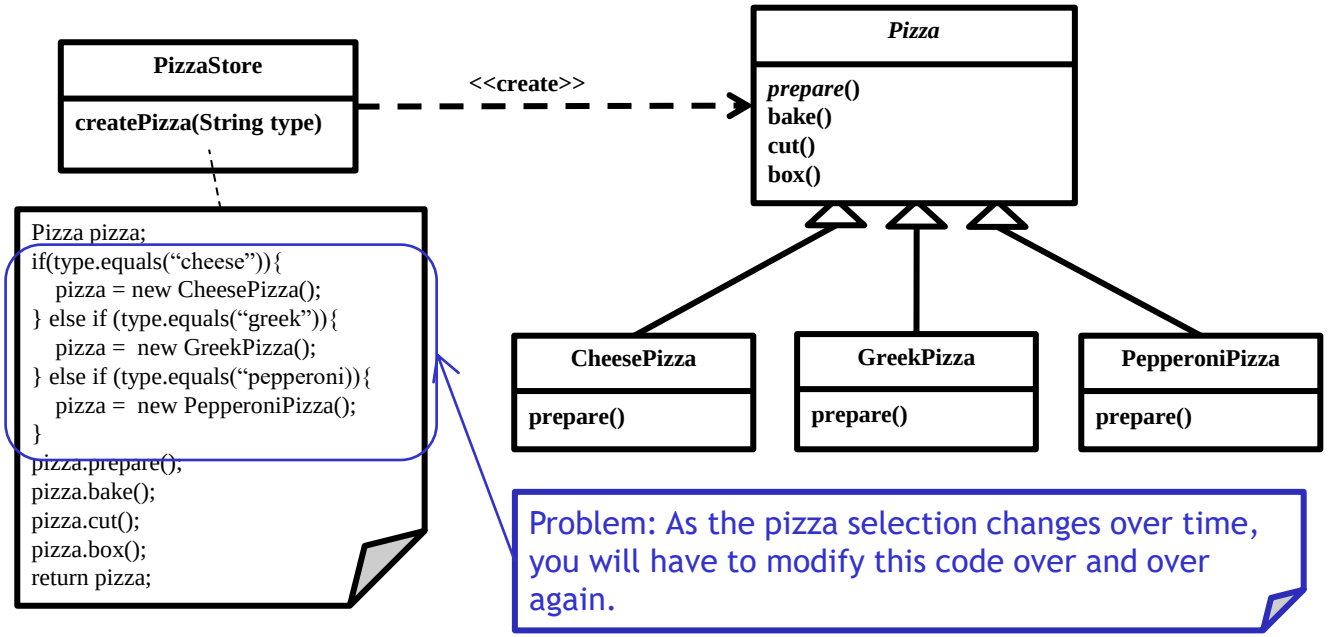
- Pizza Store
  - To make this store more competitive, you may add a new flavor of pizza or remove unpopular ones.



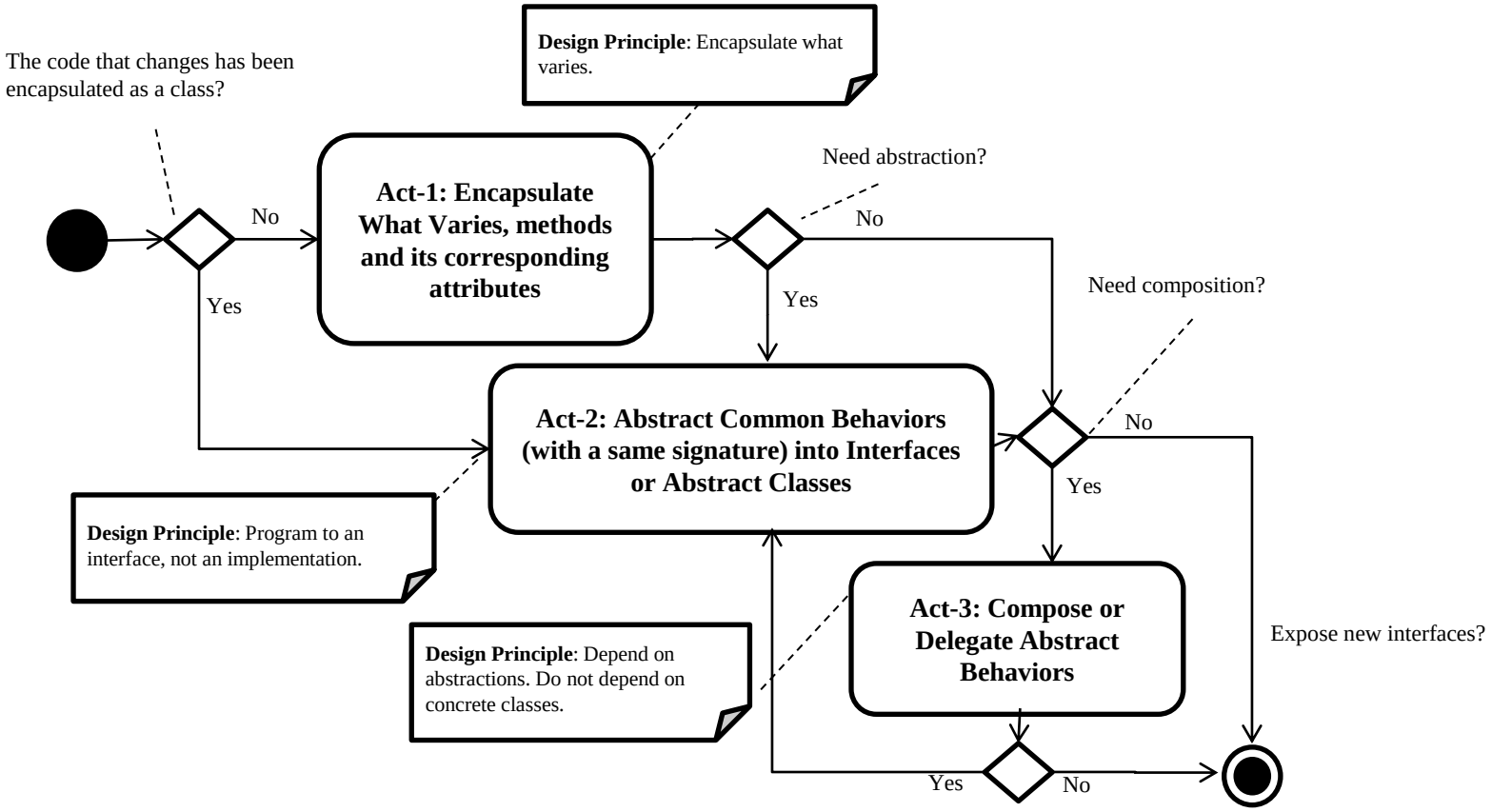
# Initial Design - Class Diagram



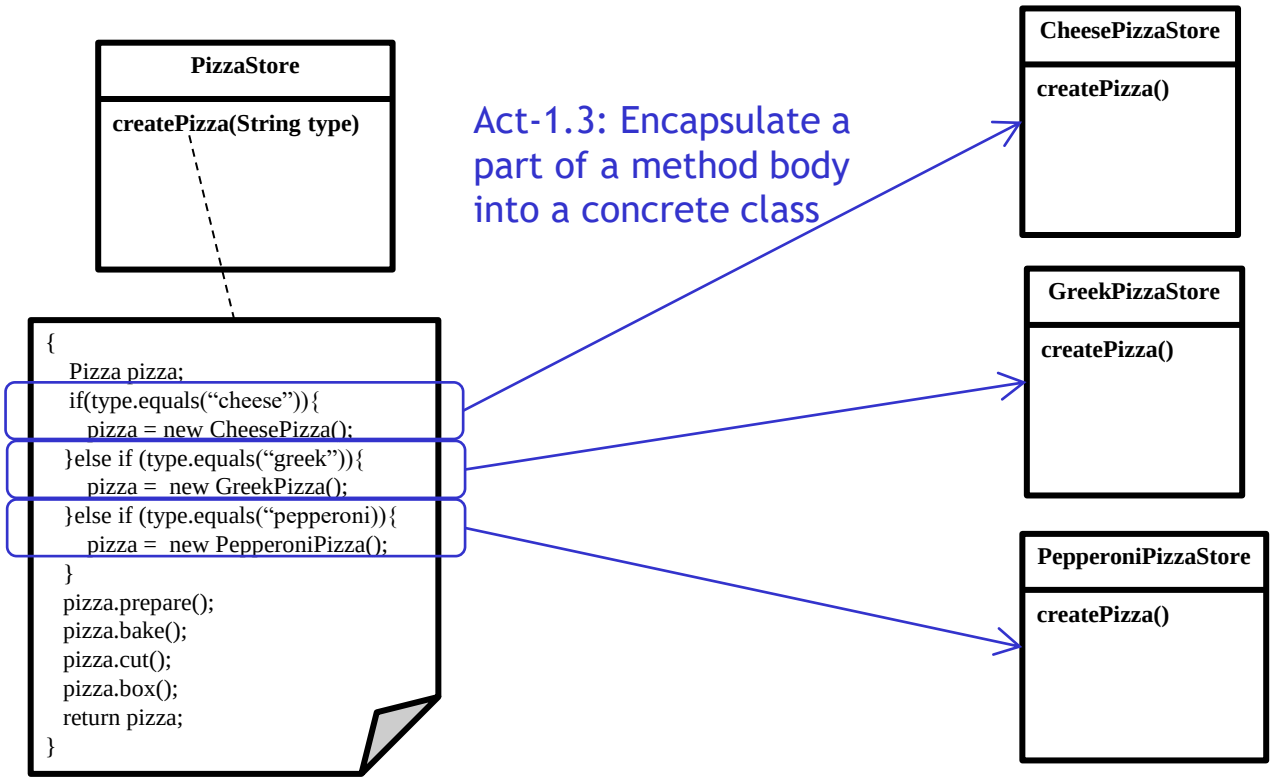
# Problems with Initial Design



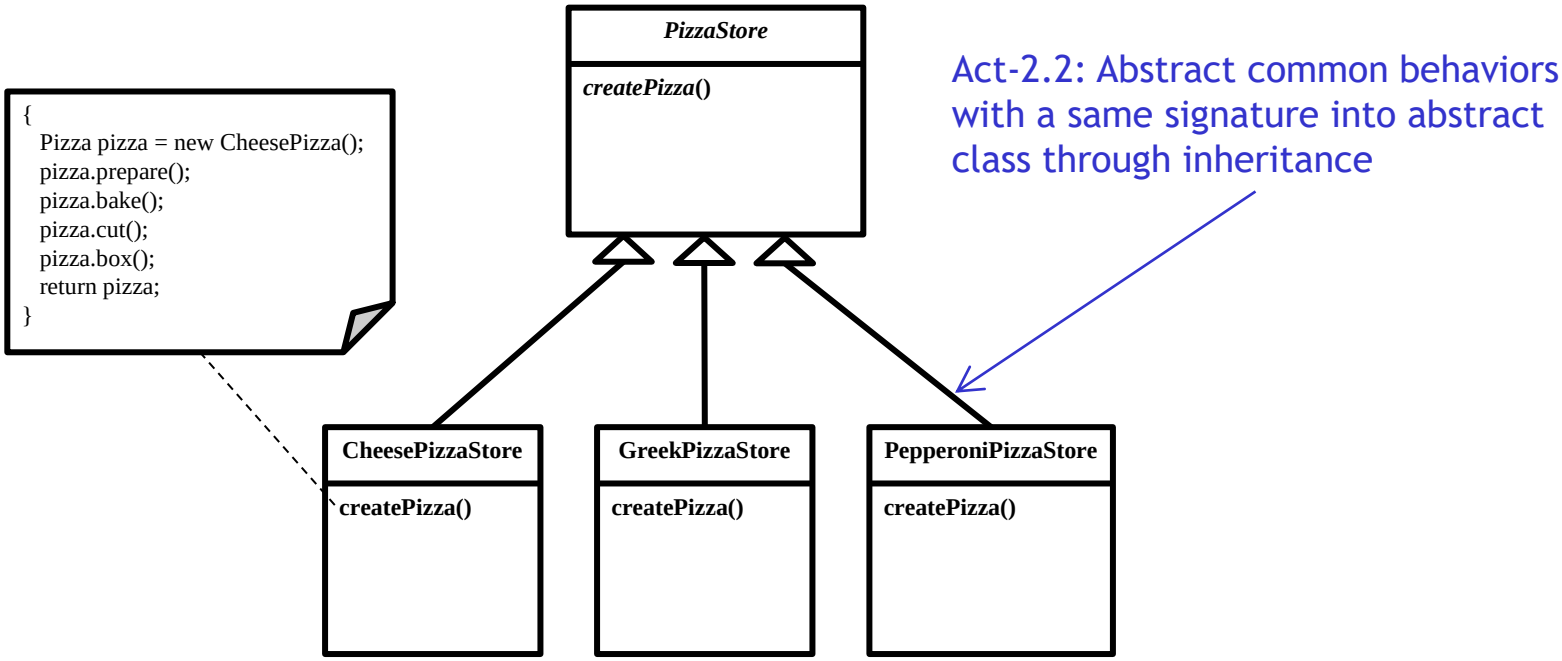
# Design Process for Change



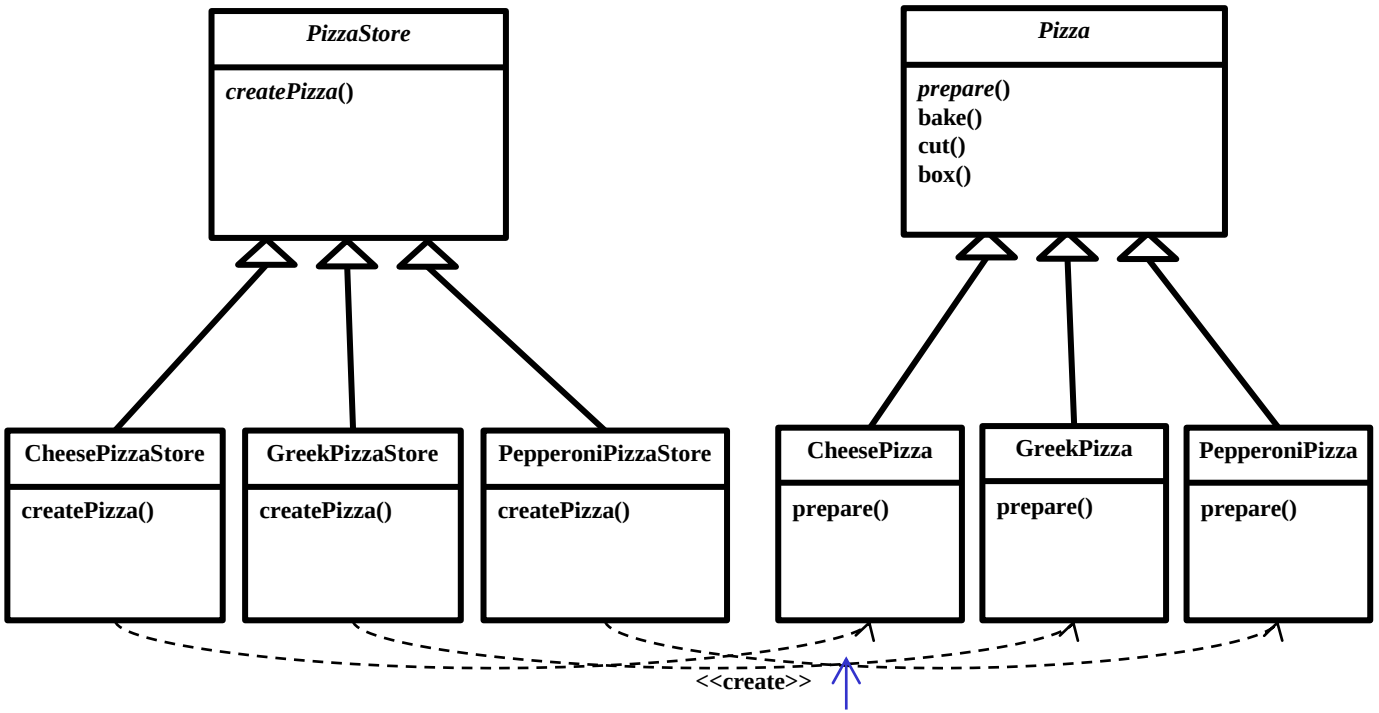
# Act-1: Encapsulate What Varies



# Act-2: Abstract Common Behaviors

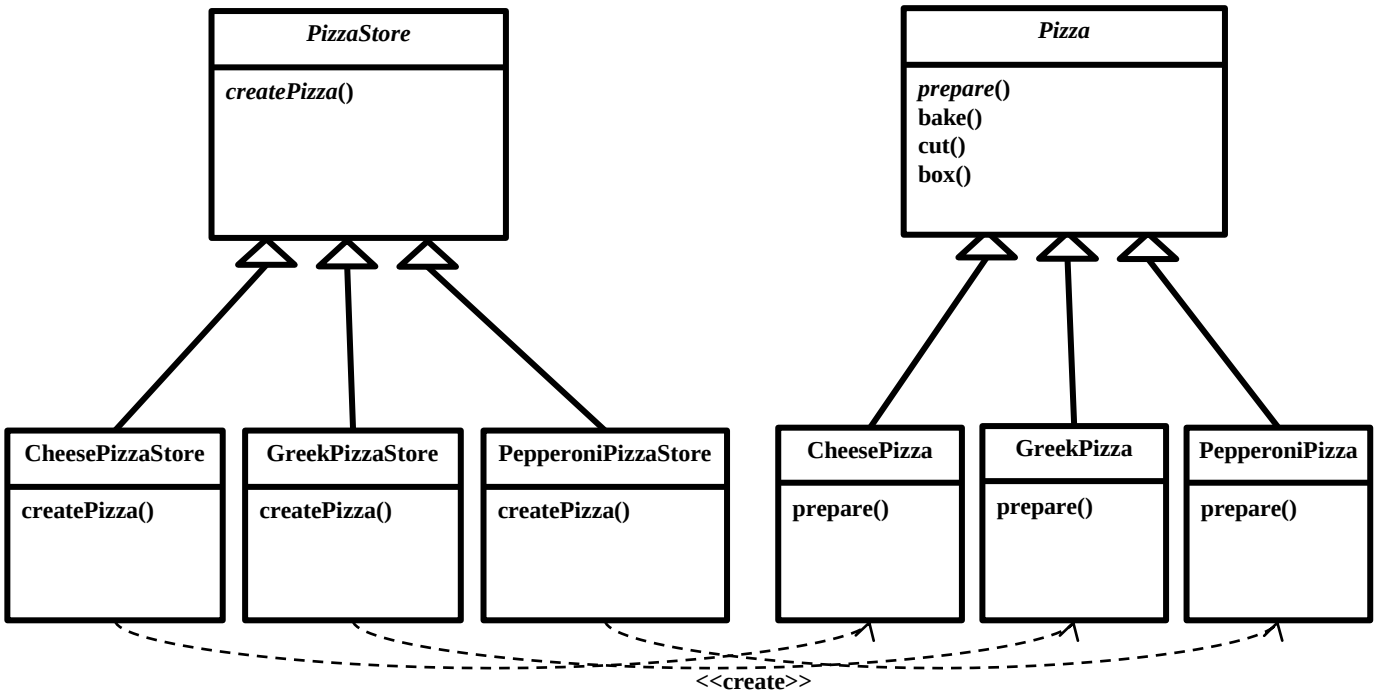


# Act-3: Delegate Abstract Behaviors




Act-3.4: Delegate behavior to a method of a concrete class

# Refactored Design after Design Process



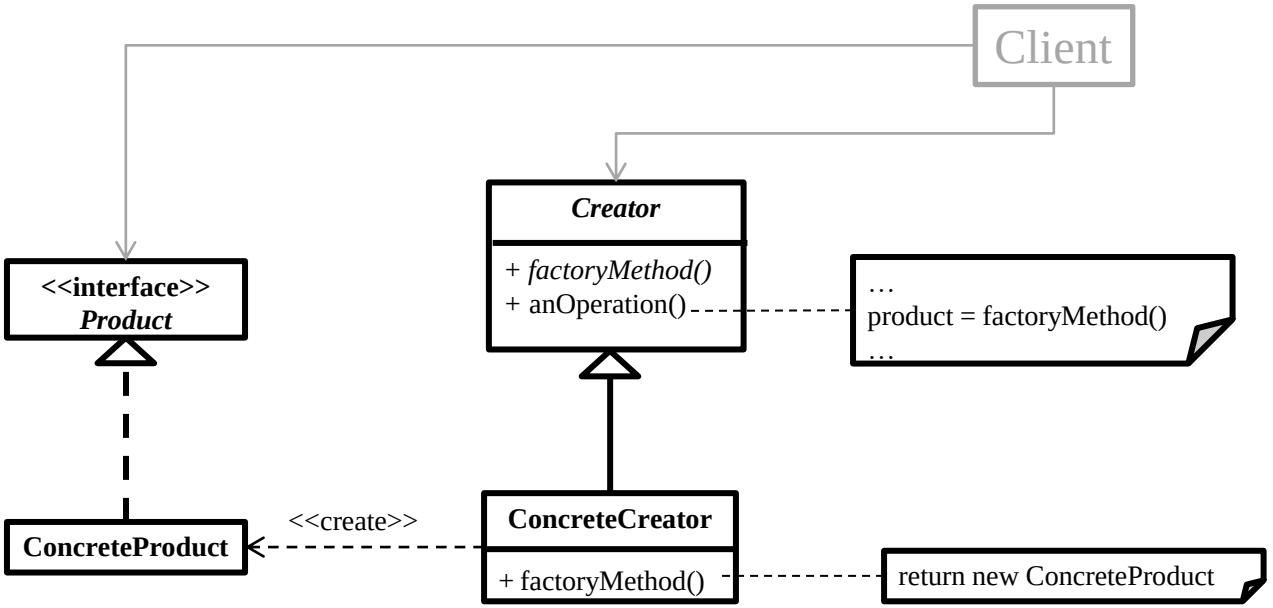
# Recurrent Problem

- As the objects being created changes over time, we need to modify the code of the creator object for the creations over and over again.
  - We need to encapsulate the knowledge of which objects to create and moves this knowledge out of the creator object.

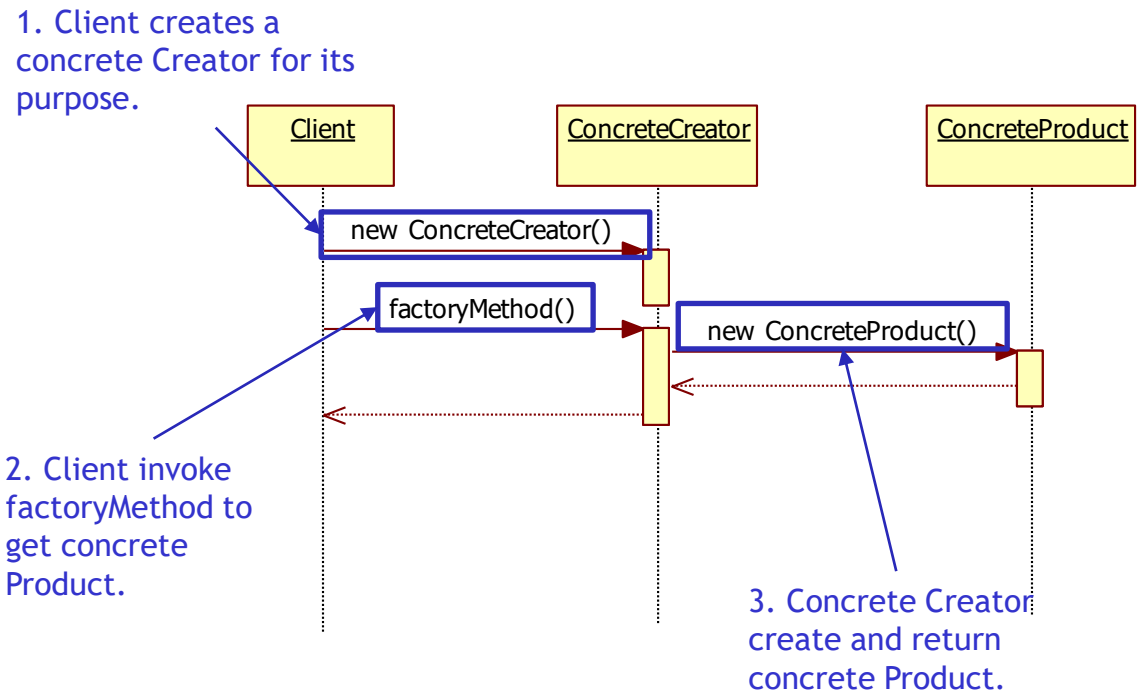
# Factory Method Pattern

- Intent
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

# Factory Method Pattern Structure$_1$



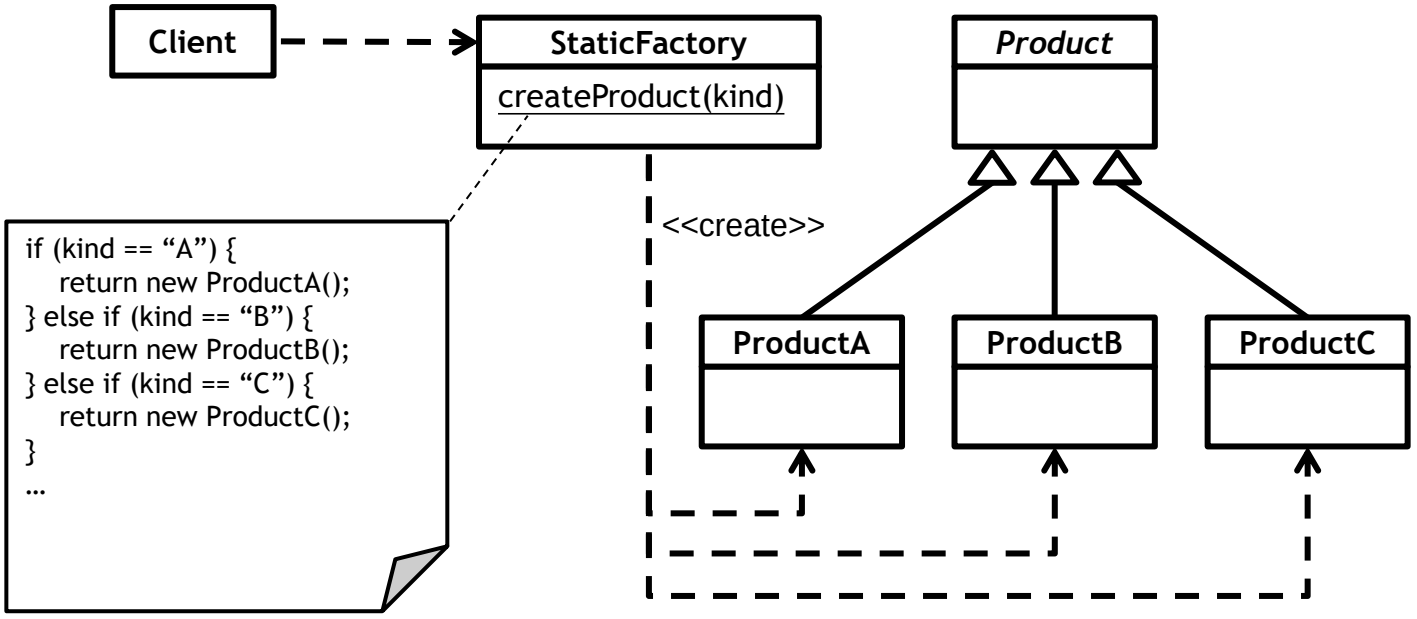
# Factory Method Pattern Structure$_2$



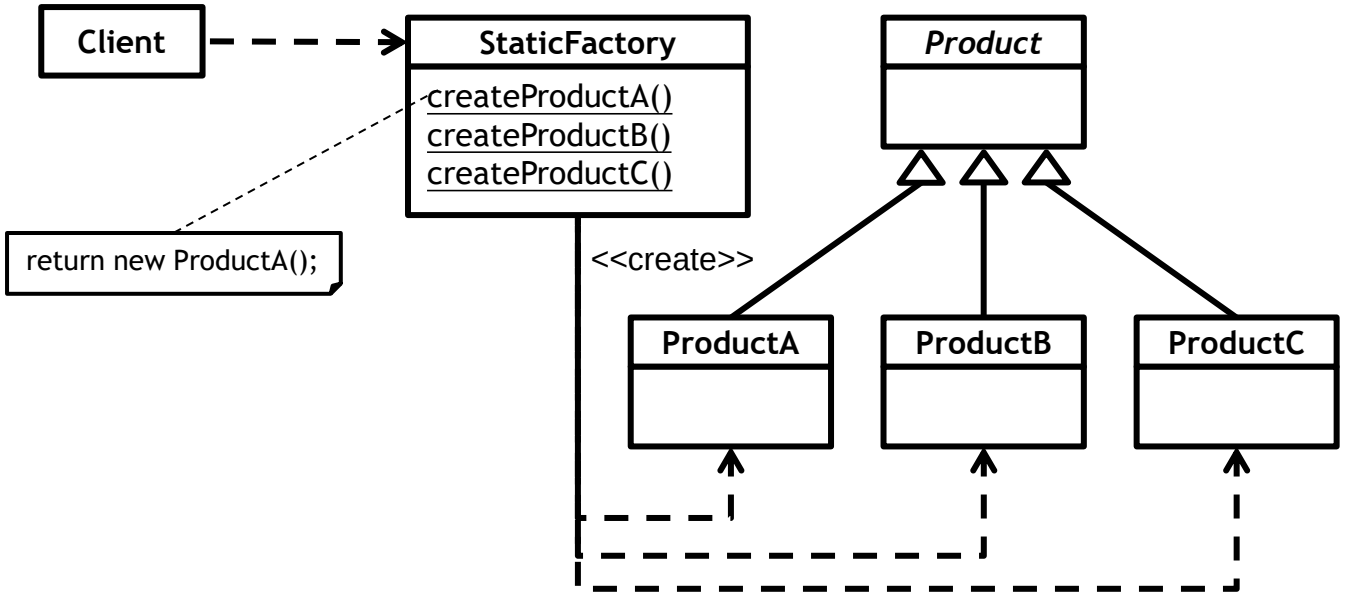
# Factory Method Pattern Structure$_3$

| | Instantiation | Use | Termination |
|---|---|---|---|
| **Client** | Other class except classes in the factory method | Other class except classes in the factory method | Other class except classes in the factory method |
| **Product** | X | Client class use Concrete Product through this interface | X |
| **Concrete Product** | **Concrete Creator** | Client class | Other class or the client class |
| **Creator** | X | Client class use this interface to get product that produced by Concrete Creator | X |
| **Concrete Creator** | Other class or the client class | Client class | Other class or the client class |

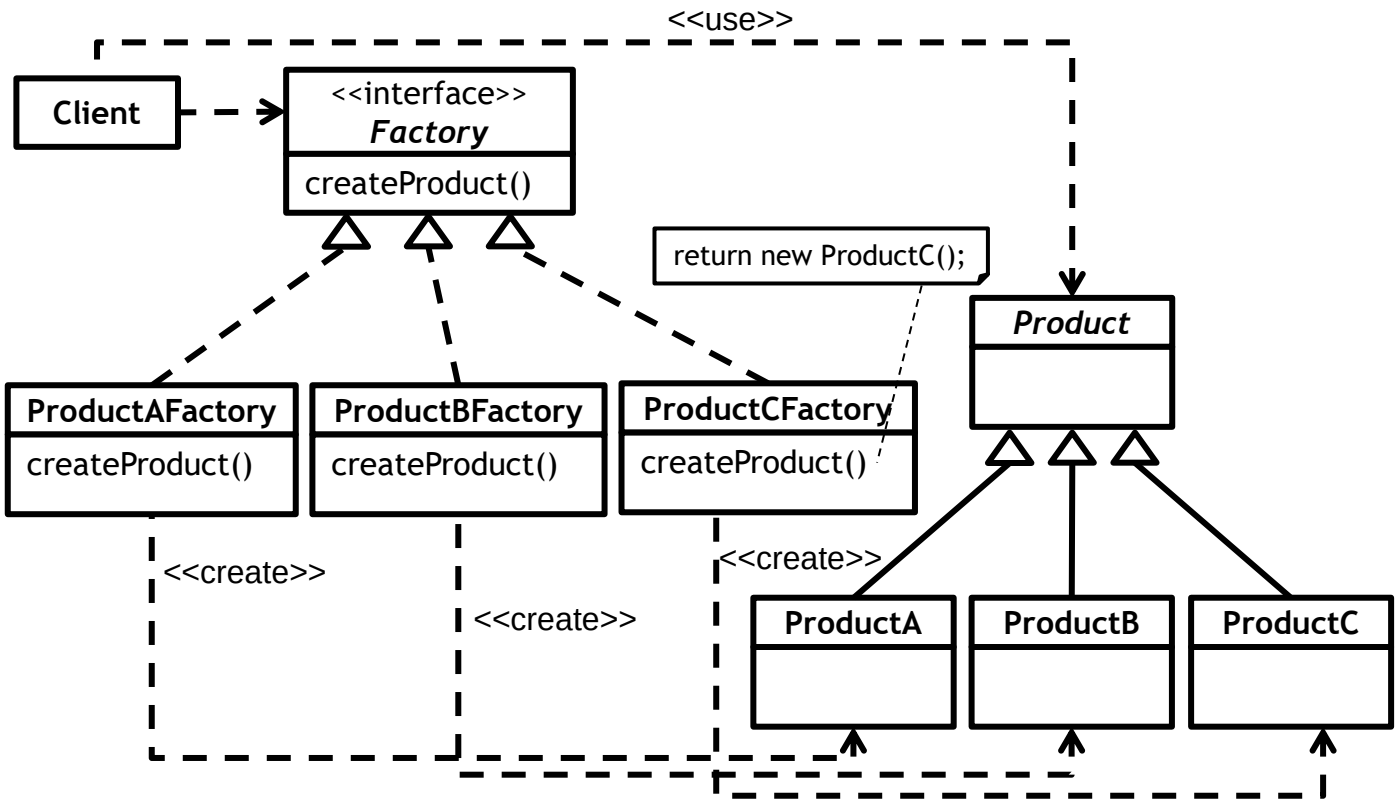
# Static Factory (I)



# Static Factory (II)



# Non-Static Factory



# Static Factory vs. Non-Static Factory

| | Static Factory | Non-Static Factory |
|---|---|---|
| **Instantiation** | You don't need to instantiate a factory object for creation. | You have to instantiate a specific factory object to create specific products. |
| **Overriding** | You can't override the factory method because it is impossible to override a static method. | You can override the factory method through subclassing. |
| **Relationship** | The factory class must know all classes that it is in charge of creating. | The factory abstraction just need to know product abstraction instead of every product implementation. |
| **Add New Products** | Open factory class and add new if-else statements or new factory methods. (It violates the Open-Closed Principle) | Add new concrete classes and implement the factory interface or inherit the factory abstract class. |