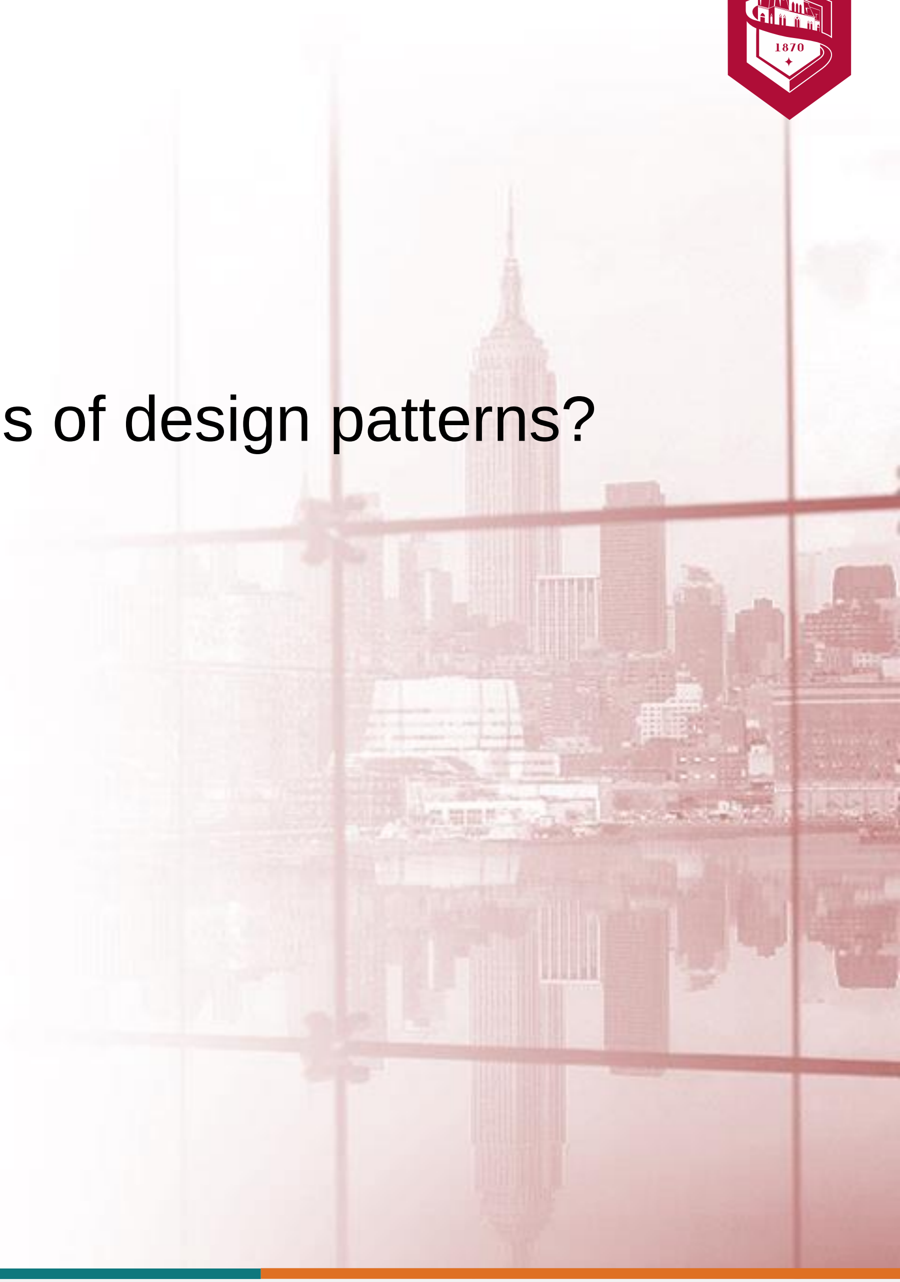
SSW 322: Software Engineering Design VI

Design Patterns (1)
2020 Spring

Prof. Lu Xiao
lxiao6@stevens.edu
Babbio 513
Office Hour: Monday/Wednesday 2 to 4 pm
Software Engineering
School of Systems and Enterprises

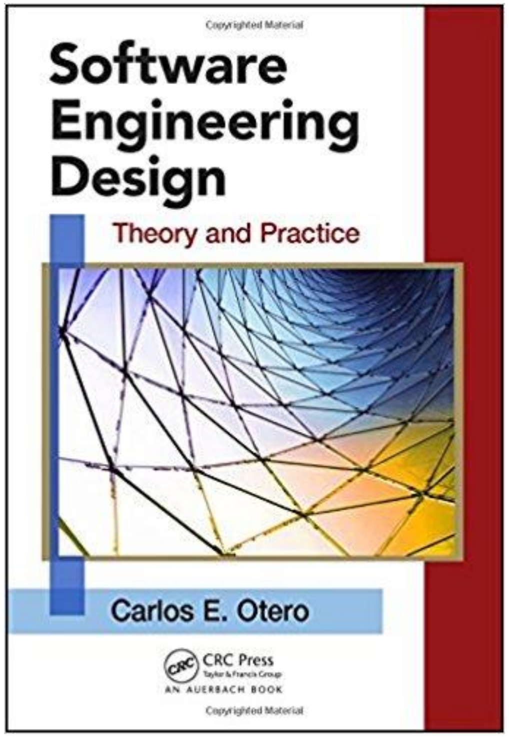
Today’s Topics

- What is a design pattern?
- Why design pattern is needed?
- What are the different categories of design patterns?
- Creational design patterns:
  - Abstract Factory Pattern
  - Factory Method Pattern
  - Singleton Pattern



Acknowledgement

- https://sourcemaking.com/design_patterns
- **Software Engineering Design: Theory and Practice**



What is design pattern?

- In software engineering, a **design pattern** is a general *repeatable solution* to a *commonly occurring problem* in software design.
  - A design pattern isn't a finished design that can be transformed directly into code.
  - It is a description or template for how to solve a problem that can be used in many different situations.

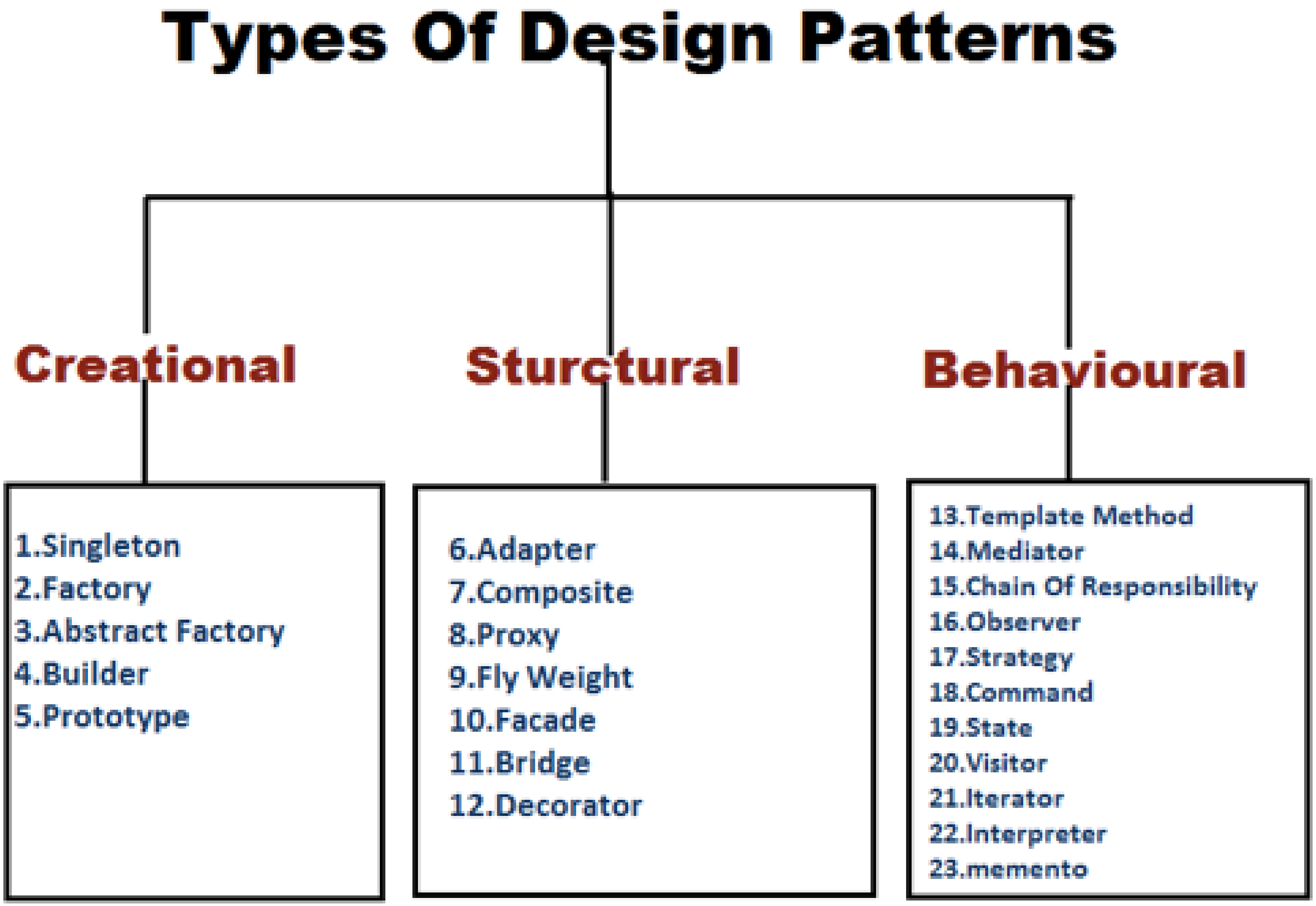
Why design patterns are useful?

- Design patterns can speed up the development process by providing tested, proven development paradigms.
- Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.
- Patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

What are the different types of design patterns?

- Creational Design Patterns
  - These design patterns are all about class instantiation.
- Structural Design Patterns
  - These design patterns are all about Class/Object composition.
- Behavioral Design Patterns
  - These design patterns are all about Class's objects communication.

Types Of Design Patterns



Creational

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

Sturctural

6.Adapter
7.Composite
8.Proxy
9.Fly Weight
10.Facade
11.Bridge
12.Decorator

Behavioural

13.Template Method
14.Mediator
15.Chain Of Responsibility
16.Observer
17.Strategy
18.Command
19.State
20.Visitor
21.Iterator
22.Interpreter
23.memento

Creational Design Patterns

- Creational design patterns are for abstracting and controlling the way objects are created in software applications.
- The key point: parts of a system responsible for creating (or instantiating) objects do so through a *common creational interface without knowledge of how the creation details.*
- Examples of creational patterns: ***abstract factory, factory method,*** builder, prototype, and ***singleton***

Abstract Factory Pattern
---creating families of products

Two Distinct Computer Families

- Consider two distinct families of computers:
  - Standard computer made up of standard keyboard, standard monitor, and standard CPU.
  - Advanced computer made up of advanced keyboard, advanced monitor, and advanced CPU.

Potential Challenges

- Advanced computer can be created using standard parts and vice versa.
- The code inside of “computer” class would be required to know which parts to use.
- Not easy to add new parts or new families of products.

Potential Challenges

- Advanced computer can be created using standard parts and vice versa.
- The code inside of “computer” class would be required to know which parts to use.
- Not easy to add new parts or new families of products.

What happens when there is addition of new part types and new models of computers?

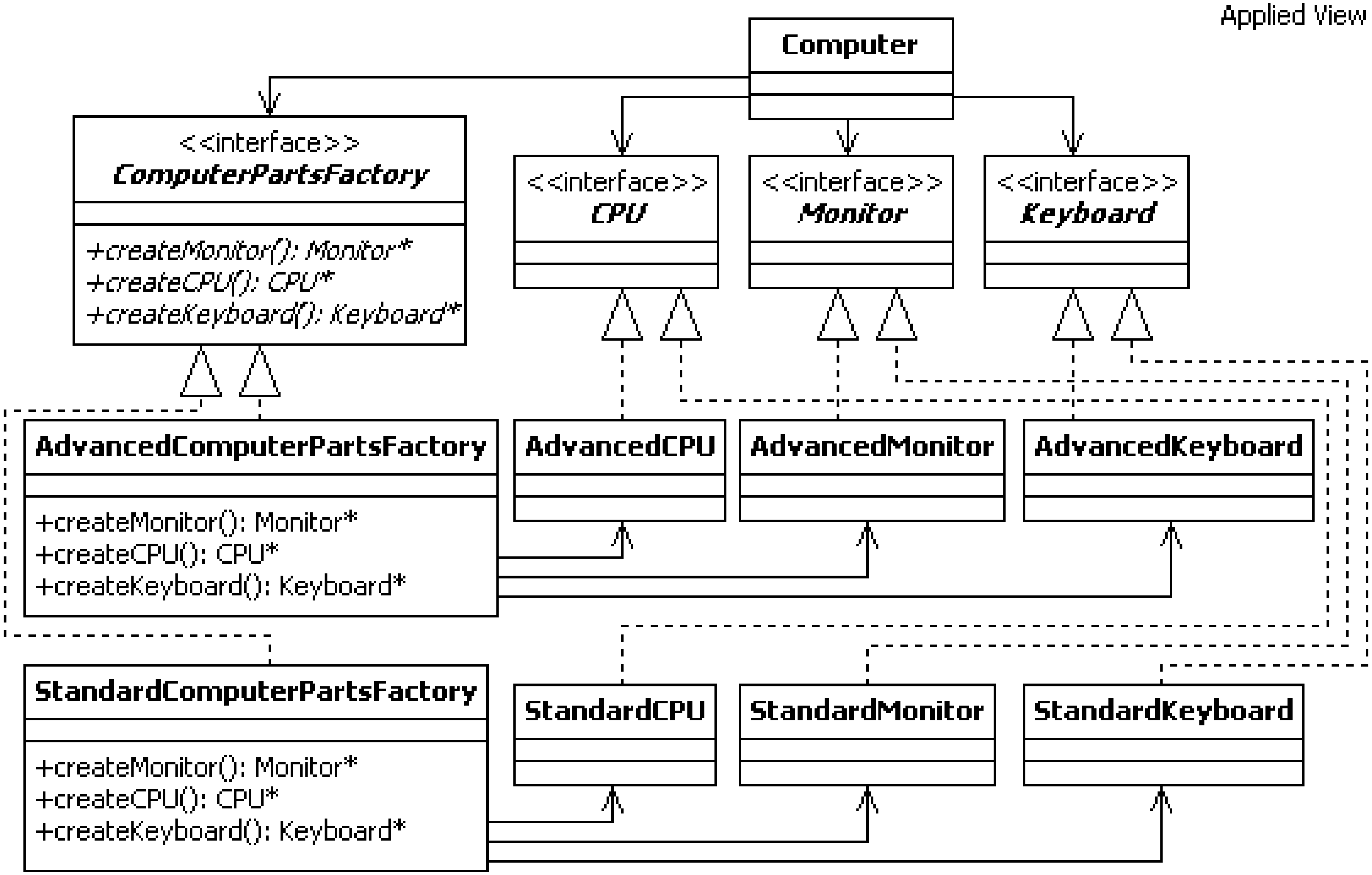
Potential Challenges

- Advanced computer can be created using standard parts and vice versa.
- The code inside of “computer” class would be required to know which parts to use.
- Not easy to add new parts or new families of products.

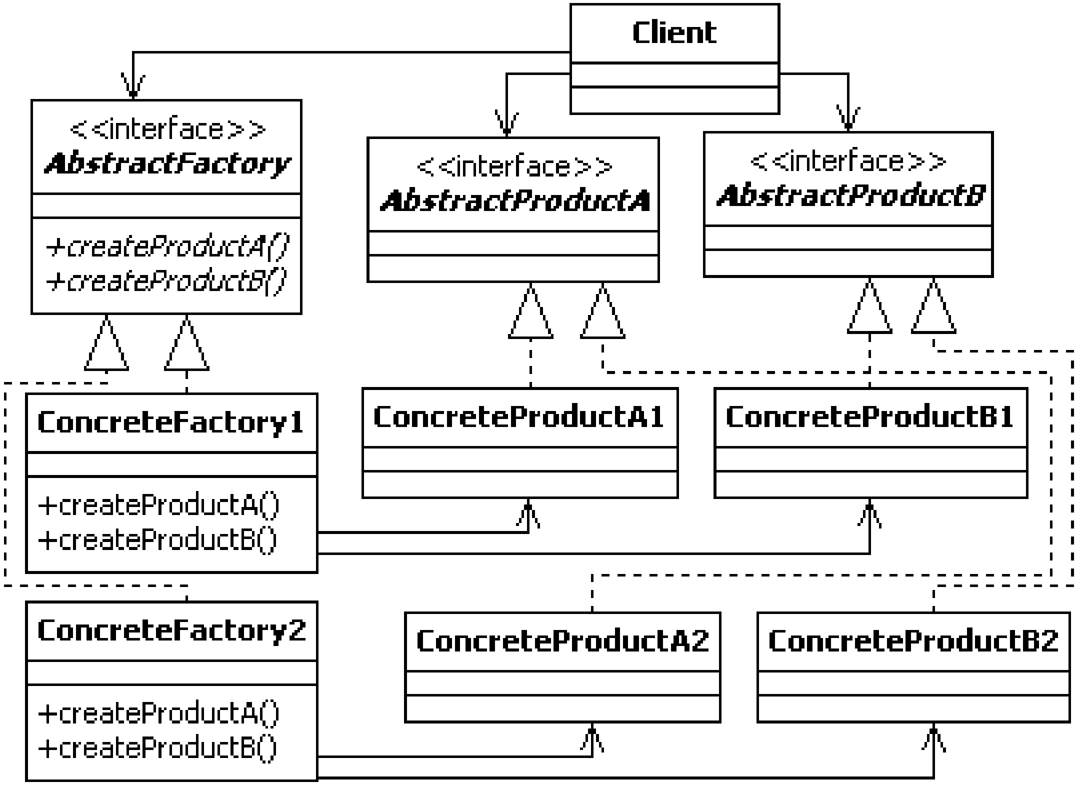
What happens when there is addition of new part types and new models of computers?

Cause a clear violation of open-close-principle: Open to Extension and Close to Modification

Meet the Abstract Factory Pattern



Abstract Factory Pattern-Photo ID



- The abstract factory is an object creational design pattern intended to manage and encapsulate the creation of a set of ***objects that conceptually belong together*** and that represent ***a specific family of products***.

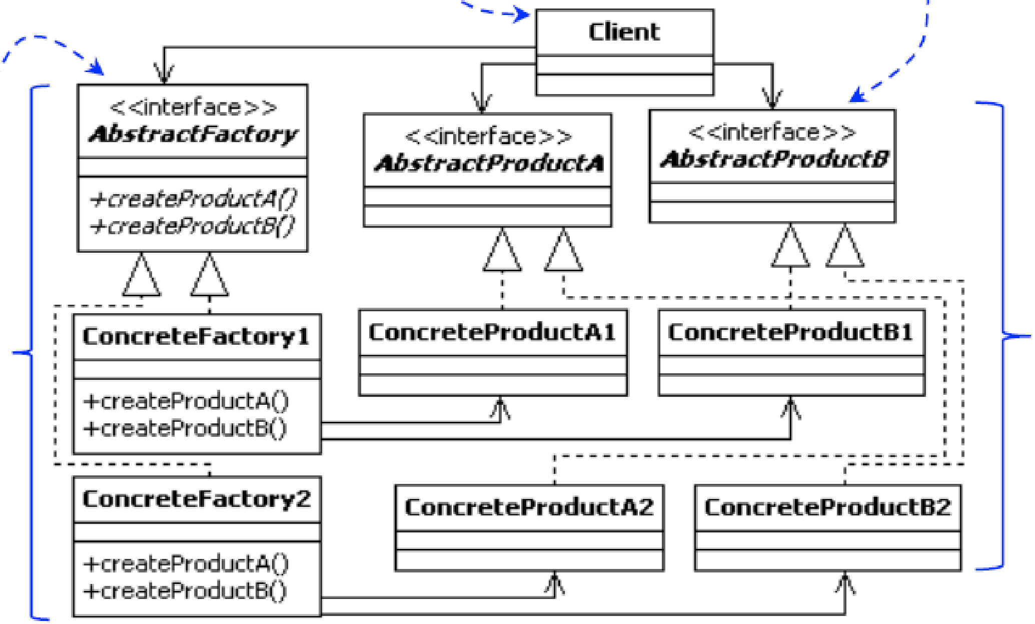
- How: provide an ***interface*** for creating ***families*** of related or dependent objects ***without specifying their concrete classes***.

Abstract Factory Pattern-Photo ID

Client only knows about creator and product interfaces! This allows us to vary behavior without changing client code!

Products need to obey the Product interface!

Factories need to obey the Factory interface!



Client

<<interface>>
AbstractFactory
+createProductA()
+createProductB()

<<interface>>
AbstractProductA

<<interface>>
AbstractProductB

ConcreteFactory1
+createProductA()
+createProductB()

ConcreteFactory2
+createProductA()
+createProductB()

ConcreteProductA1

ConcreteProductB1

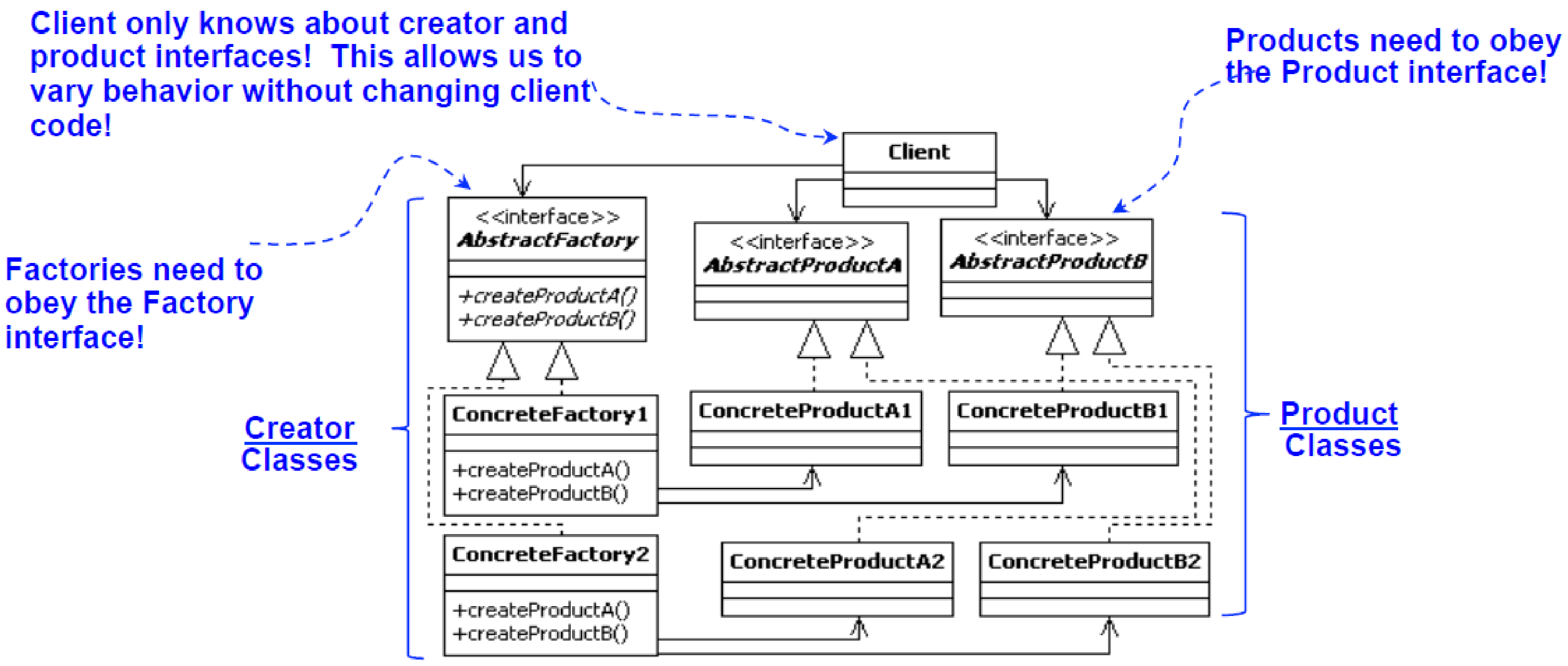
ConcreteProductA2

ConcreteProductB2

Creator Classes

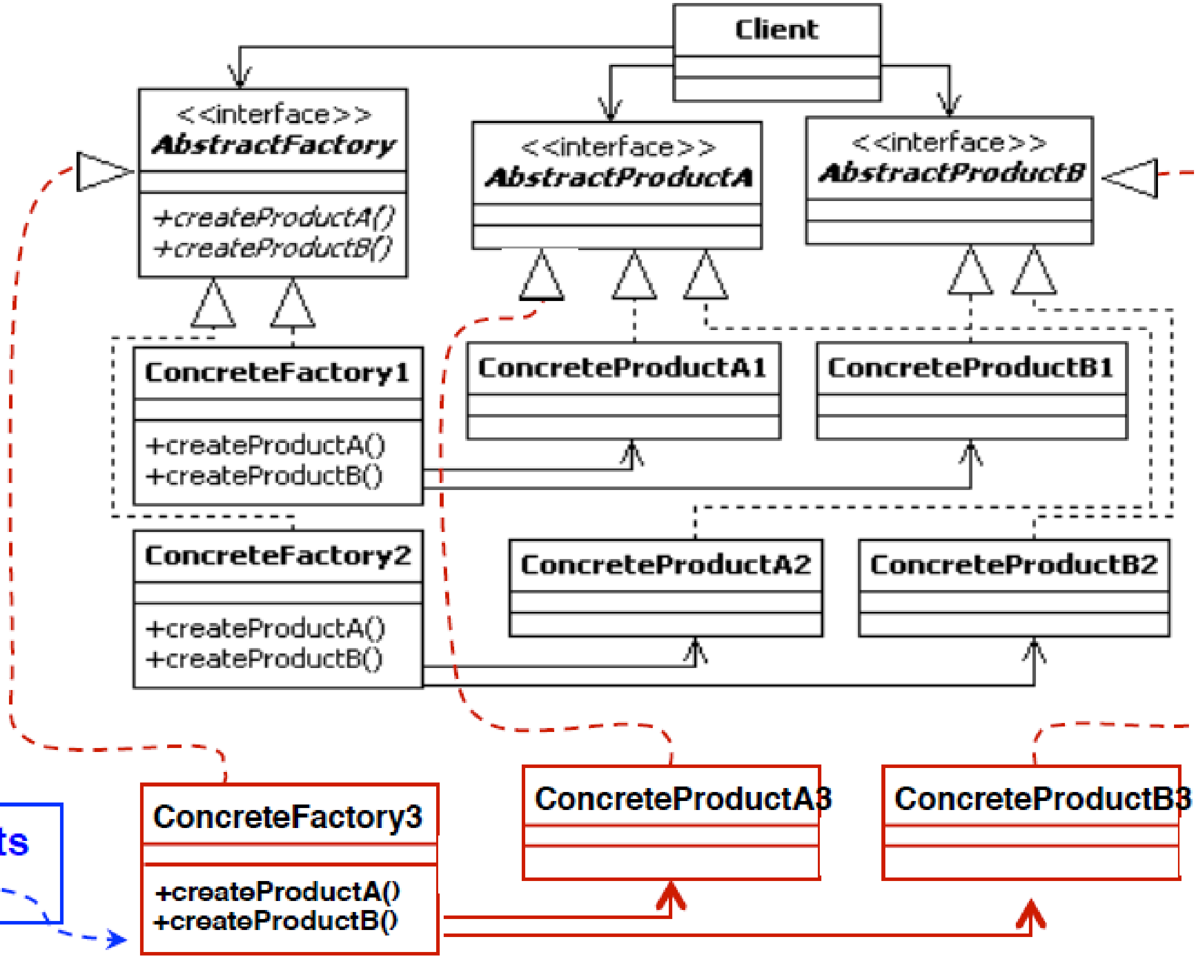
Product Classes

Abstract Factory Pattern-Photo ID



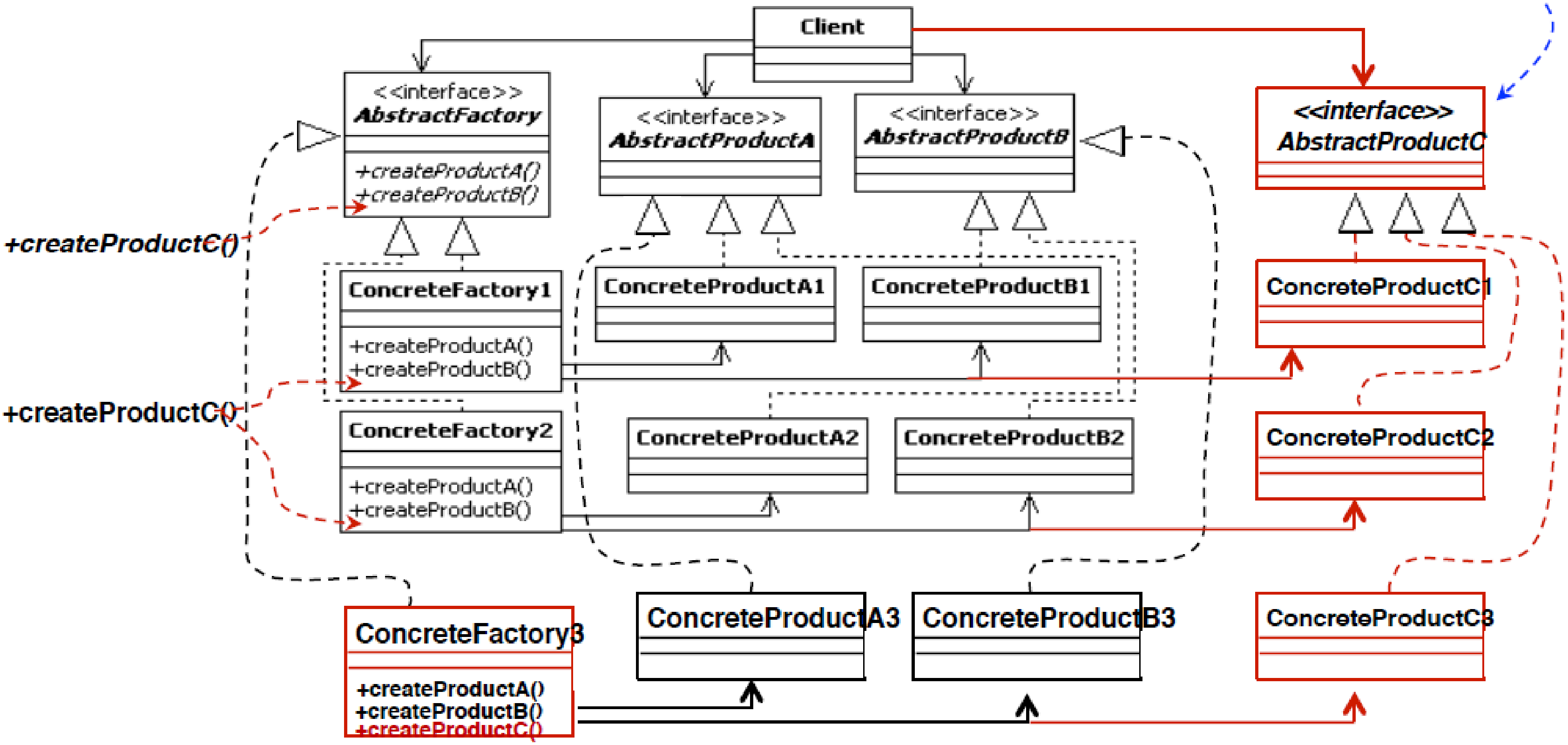
1. Different families of products can be interchangeable at run time.
2. Products that belong to a specific group/family are instantiated together.
3. For new product ***(How?)*** or new families of product ***(How?)***, just add concrete factory or parts, without changing existing code

Add a New Family of Products



A New Family of Products has been added!

Add New Products to Families



A New Product has been added to existing families!

7 Steps to Abstract Factory Pattern

1. Design the product interfaces (e.g., Cpu, Monitor, and Keyboard)
2. Identify the different families (or groups) required for the problem (e.g., standard vs. advanced computers)
3. For each family (or group) identified in step 2, design concrete products that realize the respective product interfaces identified in step 1.
4. Create the factory interface (e.g., ComputerPartsFactory). The factory interface contains *n* interface methods, one for each product interface identified in step 1.
5. For each family (or group) identified in step 2, create concrete factories that realize the factory interface created in step 4.
6. Associate each concrete factory from step 5 with their respective products from step 3.
7. Create the Client (e.g., Computer) which is associated with both product and factory interfaces created in steps 1 and 4, respectively.

Pros and Cons

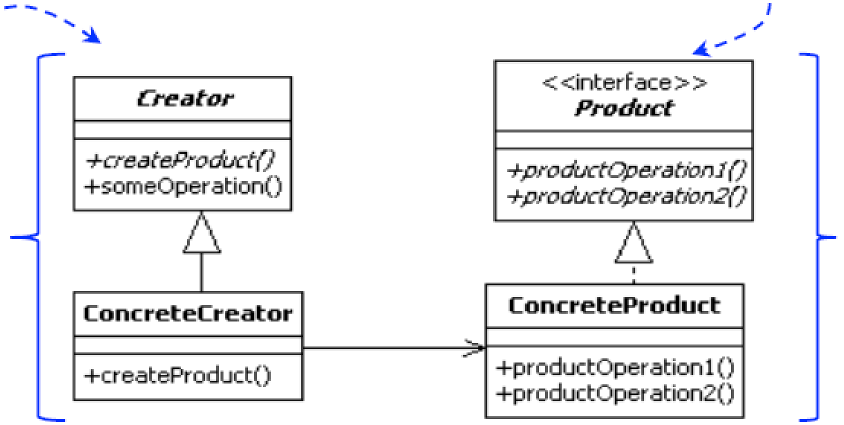
- Cons
  - Large number of classes are required
- Pros
  - Isolates concrete product classes so that ***reusing*** them becomes easier.
  - Promotes ***consistency*** within specific product families.
  - Adding new families of products require ***no modification*** to existing code.
  - Helps minimize the degree of complexity when changing the system to meet ***future needs.***

Factory Method Pattern
---A simplified version of the AF pattern

Factory Method Pattern

- The factory method design pattern is a class creational pattern used to encapsulate and ***defer object instantiation to derived classes*** (a simplified version of the AF pattern)
- How: 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-Photo ID



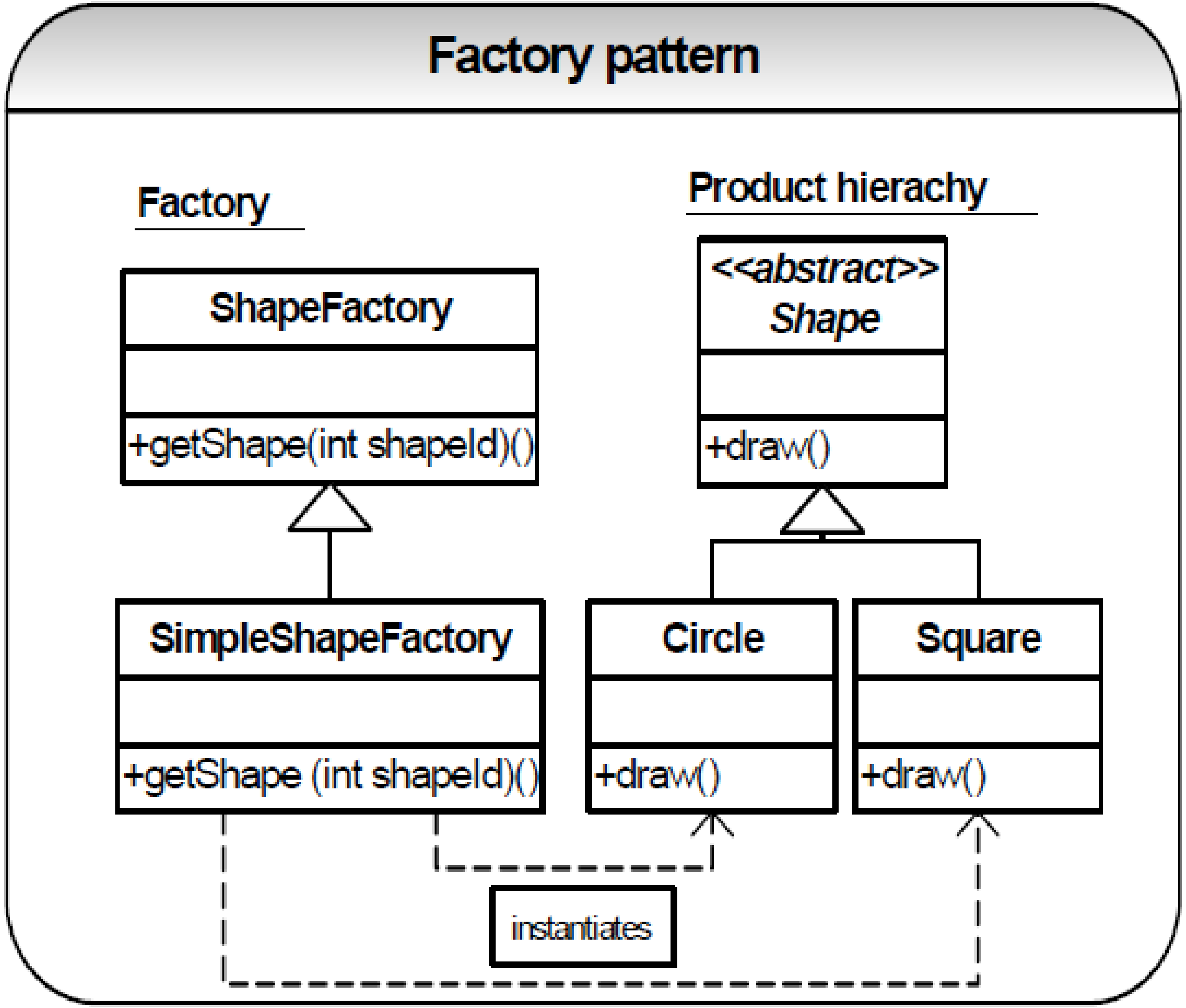
Factories need to obey the Factory interface!

Creator Classes

Products need to obey the Product interface!

Product Classes

Factory Method Pattern-Example



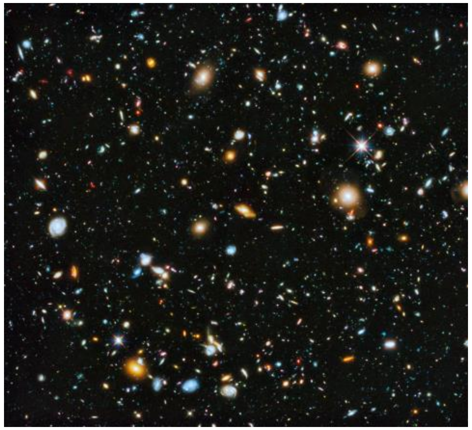
AF vs. FM Pattern Summary

- All factories encapsulate object creation. All factory patterns promote loose coupling by reducing the dependency of your application on concrete classes.
- **Abstract Factory:**
  - The Creator of an Abstract Factory Method Pattern is responsible for creating ***a family of different type of products***.
  - The creation of different families of products are ***implemented by concrete factories.***
- **Factory Method:**
  - The Creator of a Factory Method Pattern is only responsible for creating ***a specific type of product.***
  - The instantiation of the concrete products are ***deferred to the subclasses of creator.***

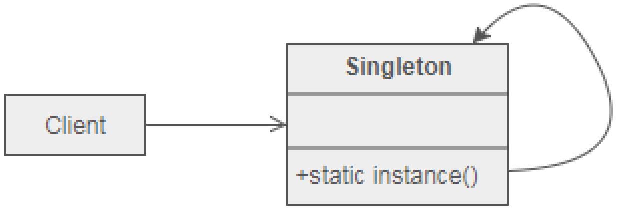
Singleton Pattern
---Ensure a class has only one instance

Singleton Design Pattern

- Problem: Application needs **one, and only one, instance of an object**. Additionally, lazy initialization and global access are necessary.
  - Ensure a class has only one instance and provides a global point of access to it.
  - Encapsulated "just-in-time initialization" or "initialization on first use".

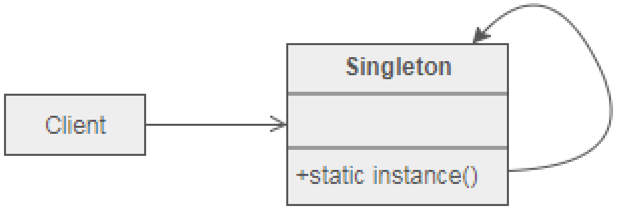


Singleton Pattern – Photo ID



- How do we ensure that a class has only one instance and that the instance is easily accessible?

Singleton Pattern – Photo ID



- By using a special method to instantiate the object.
- The method can check to see if it is already instantiated.
  - If it is, it returns a reference to the previously instantiated object.
  - If it is not, then the method instantiates the object and returns a reference to it.
- This requires that the only way to instantiate the object is through this method. Therefore, the constructor to the class must be *private or protected*.

Sample Code-Java

```java
public class SingletonExample {

        private static SingletonExample instance = null;



        protected SingletonExample() {}

        public static SingletonExample getInstance() {
            if(instance == null) {
                  instance = new SingletonExample();
            }
            return instance;
        }
}
```

Singleton Pattern Key Features

Solution: Guarantees one instance.

Implementation:

- Add a private static member of the class that refers to the desired object. (Initially, it is null.)
- Add a public static method that instantiates this class if this member is null (and sets this member's value) and then returns the value of this member.
- Set the constructor's status to protected or private so that no one can directly instantiate this class and bypass the static constructor mechanism.

thank you