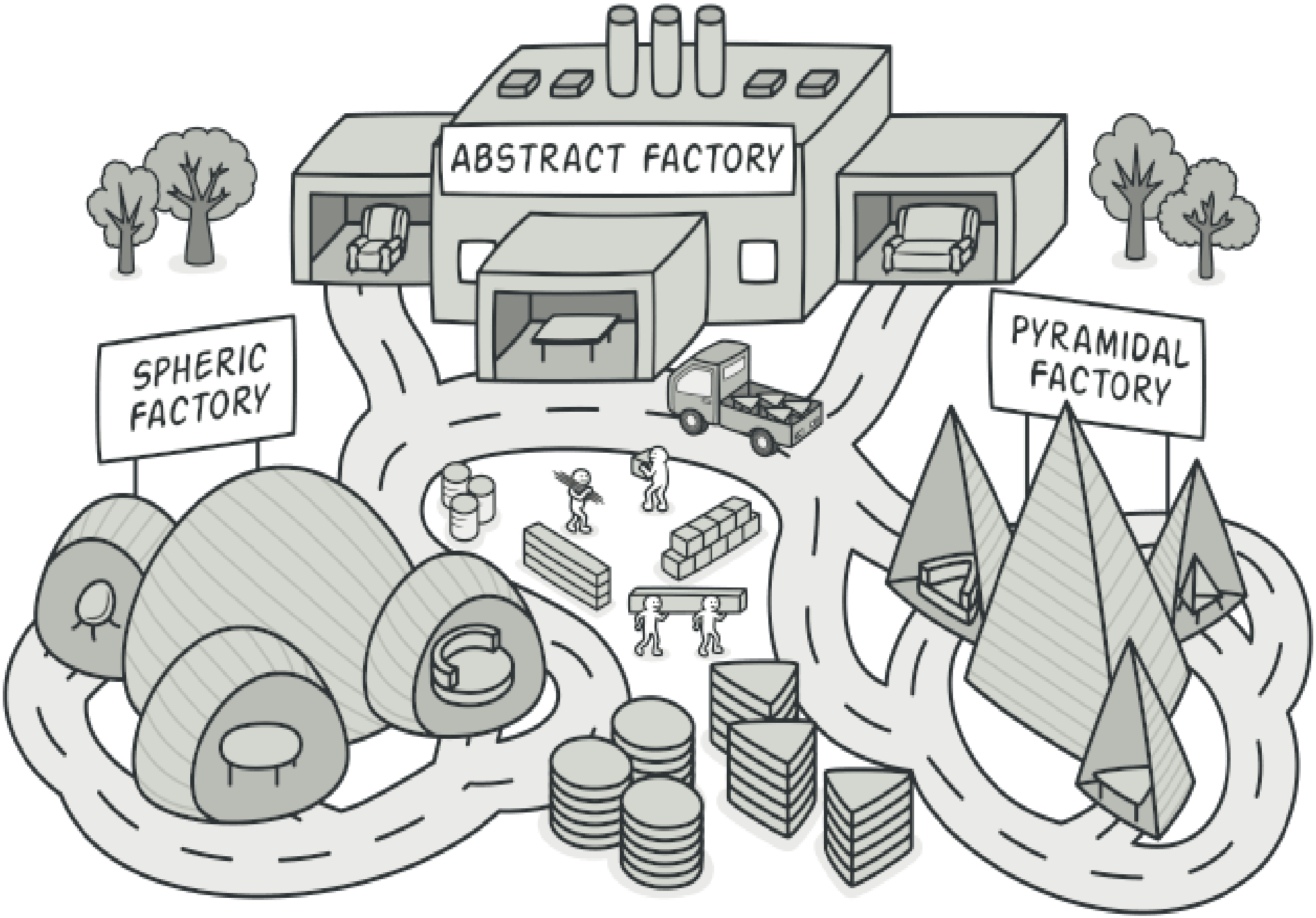
Abstract Factory

Lecture 18

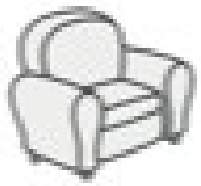
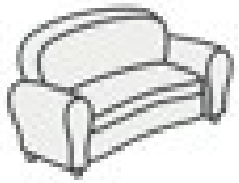
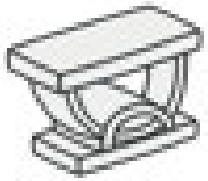
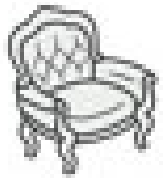
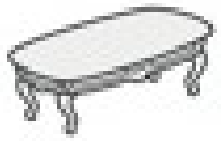
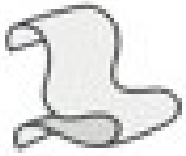
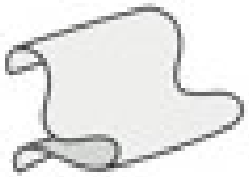
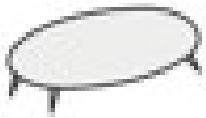
Intent

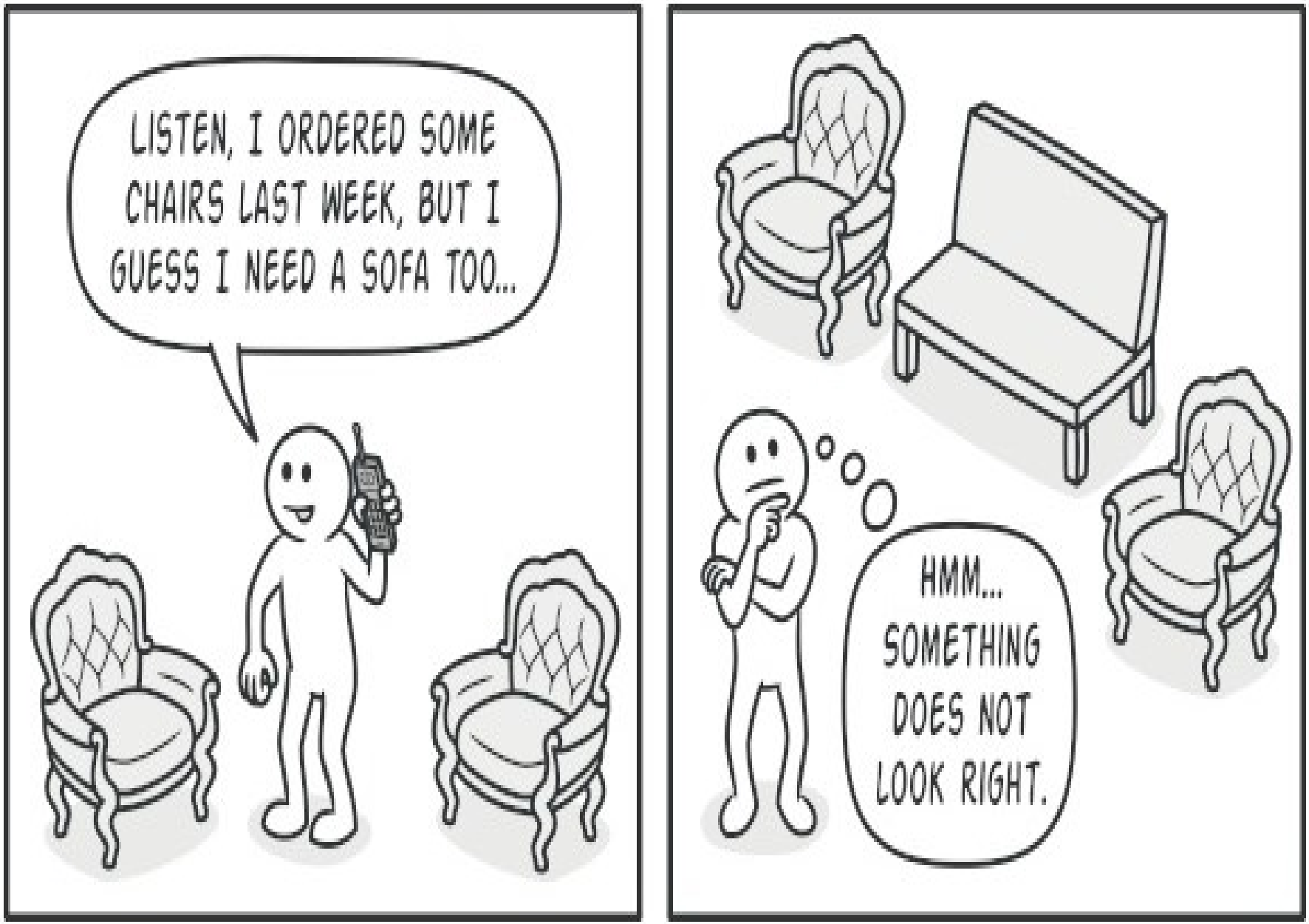
- **Abstract Factory** is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.


ABSTRACT FACTORY
SPHERIC FACTORY
PYRAMIDAL FACTORY

Problem

- Imagine that you're creating a furniture shop simulator. Your code consists of classes that represent:
- A family of related products, say: Chair + Sofa + CoffeeTable.
- Several variants of this family. For example, products Chair + Sofa + CoffeeTable are available in these variants:
  - Modern, Victorian, ArtDeco.

	Chair	Sofa	Coffee Table
Art Deco			
Victorian			
Modern			

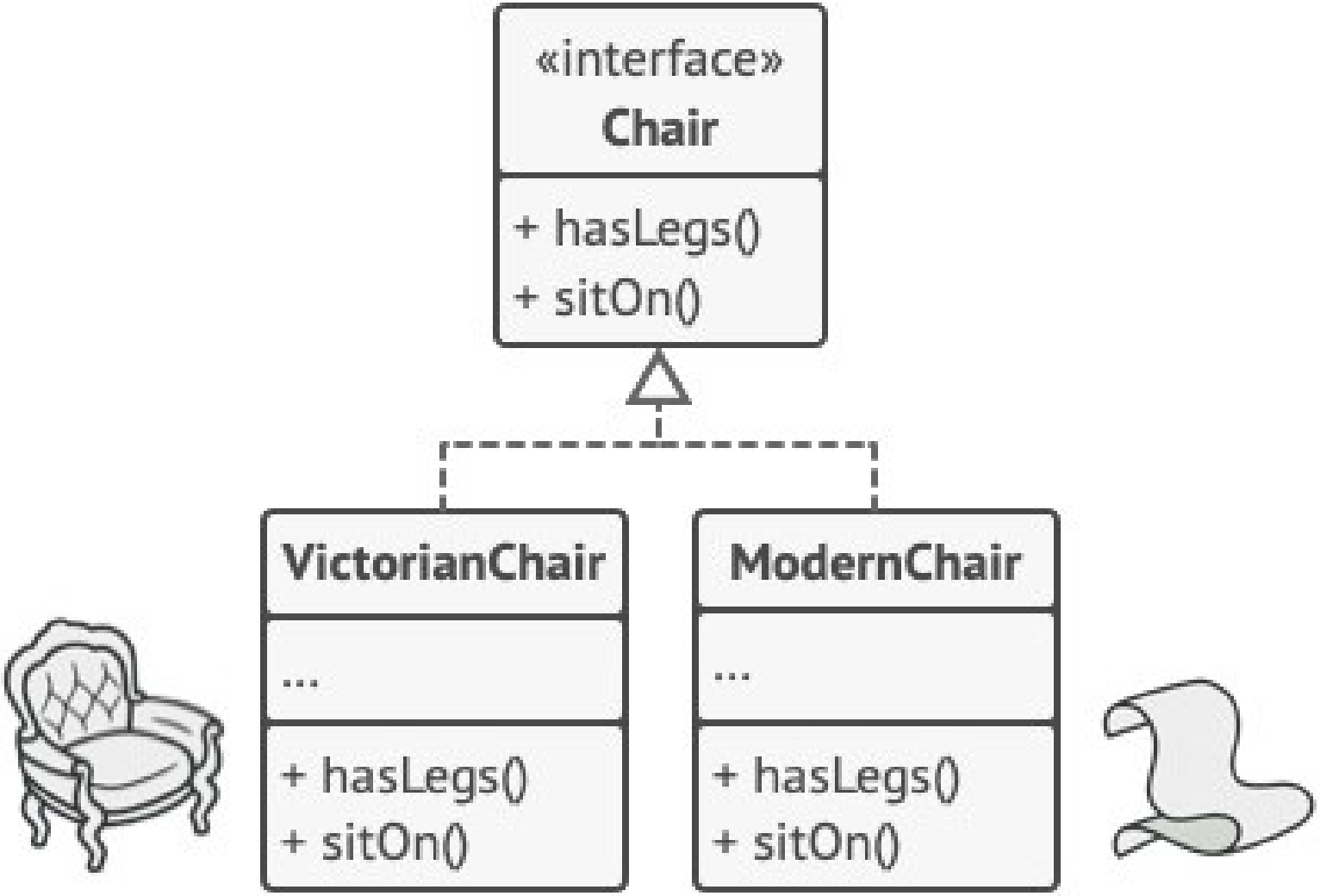

LISTEN, I ORDERED SOME CHAIRS LAST WEEK, BUT I GUESS I NEED A SOFA TOO...
HMM... SOMETHING DOES NOT LOOK RIGHT.

Problem

- You need a way to create individual furniture objects so that they match other objects of the same family. Customers get quite mad when they receive non-matching furniture.
- Also, you don’t want to change existing code when adding new products or families of products to the program. Furniture vendors update their catalogs very often, and you wouldn’t want to change the core code each time it happens.

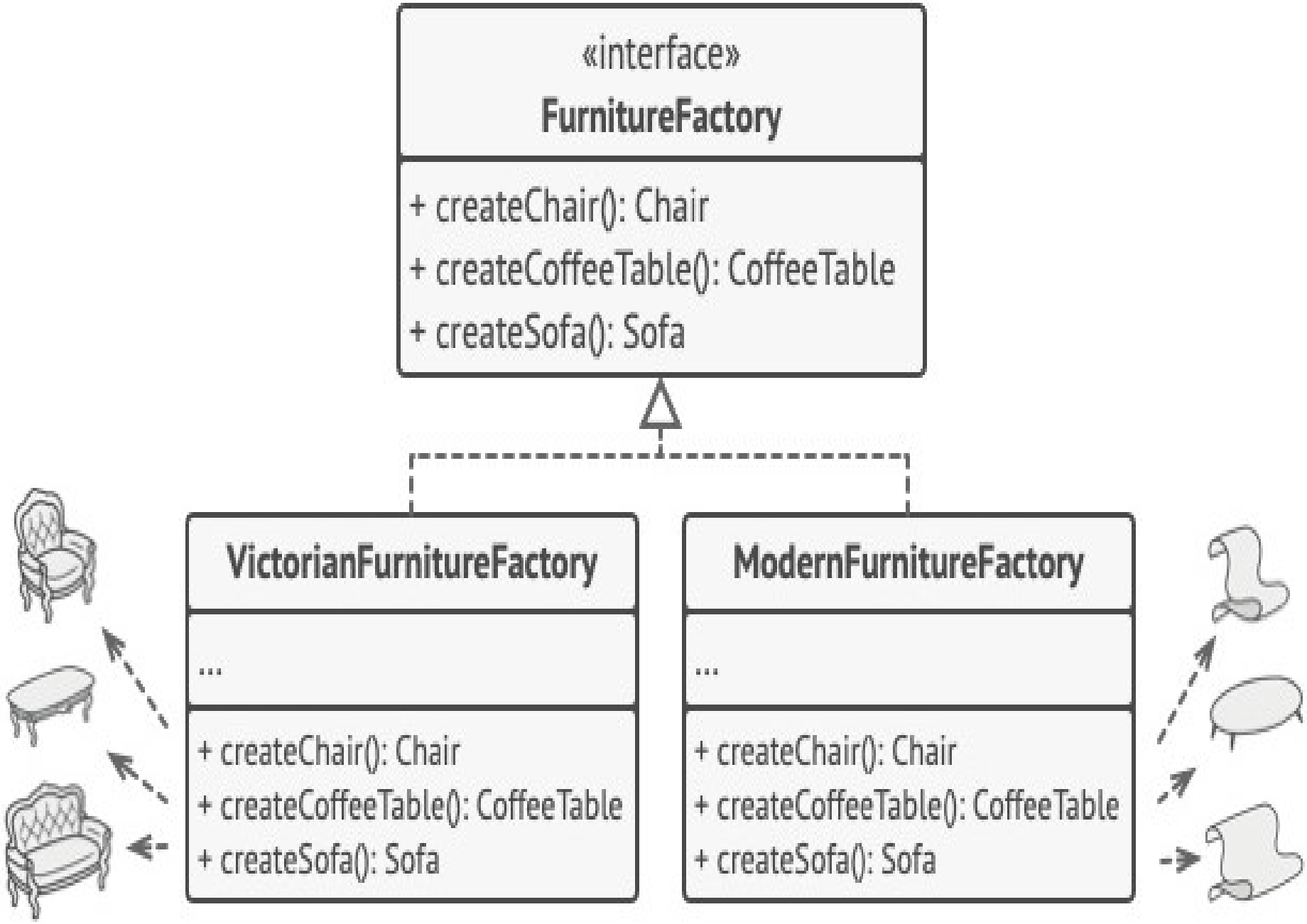
Solution

- The first thing the Abstract Factory pattern suggests is to explicitly declare interfaces for each distinct product of the product family (e.g., chair, sofa or coffee table).
- Then you can make all variants of products follow those interfaces.
- For example, all chair variants can implement the Chair interface; all coffee table variants can implement the CoffeeTable interface, and so on.


«interface»
Chair
+ hasLegs()
+ sitOn()
VictorianChair
...
+ hasLegs()
+ sitOn()
ModernChair
...
+ hasLegs()
+ sitOn()

Solution

- The next move is to declare the Abstract Factory—an interface with a list of creation methods for all products that are part of the product family (for example, createChair, createSofa and createCoffeeTable).
- These methods must return abstract product types represented by the interfaces we extracted previously: Chair, Sofa, CoffeeTable and so on.

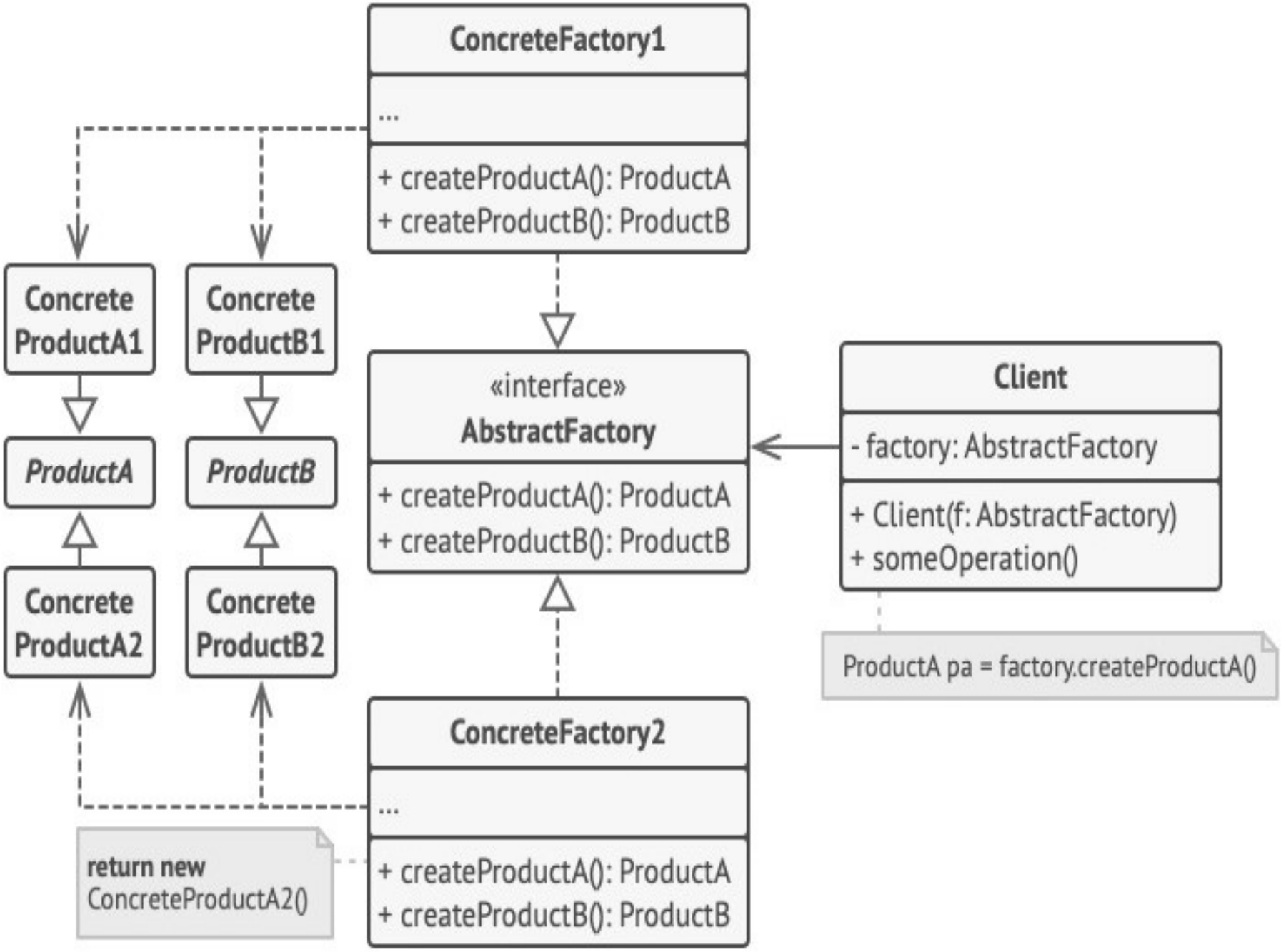

«interface»
FurnitureFactory
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
VictorianFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
ModernFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa

Explanation

- Now, how about the product variants? For each variant of a product family, we create a separate factory class based on
  the AbstractFactory interface.
- A factory is a class that returns products of a particular kind.
- For example, the ModernFurnitureFactory can only
  create ModernChair, ModernSofa and ModernCof feeTable objects.

Explanation

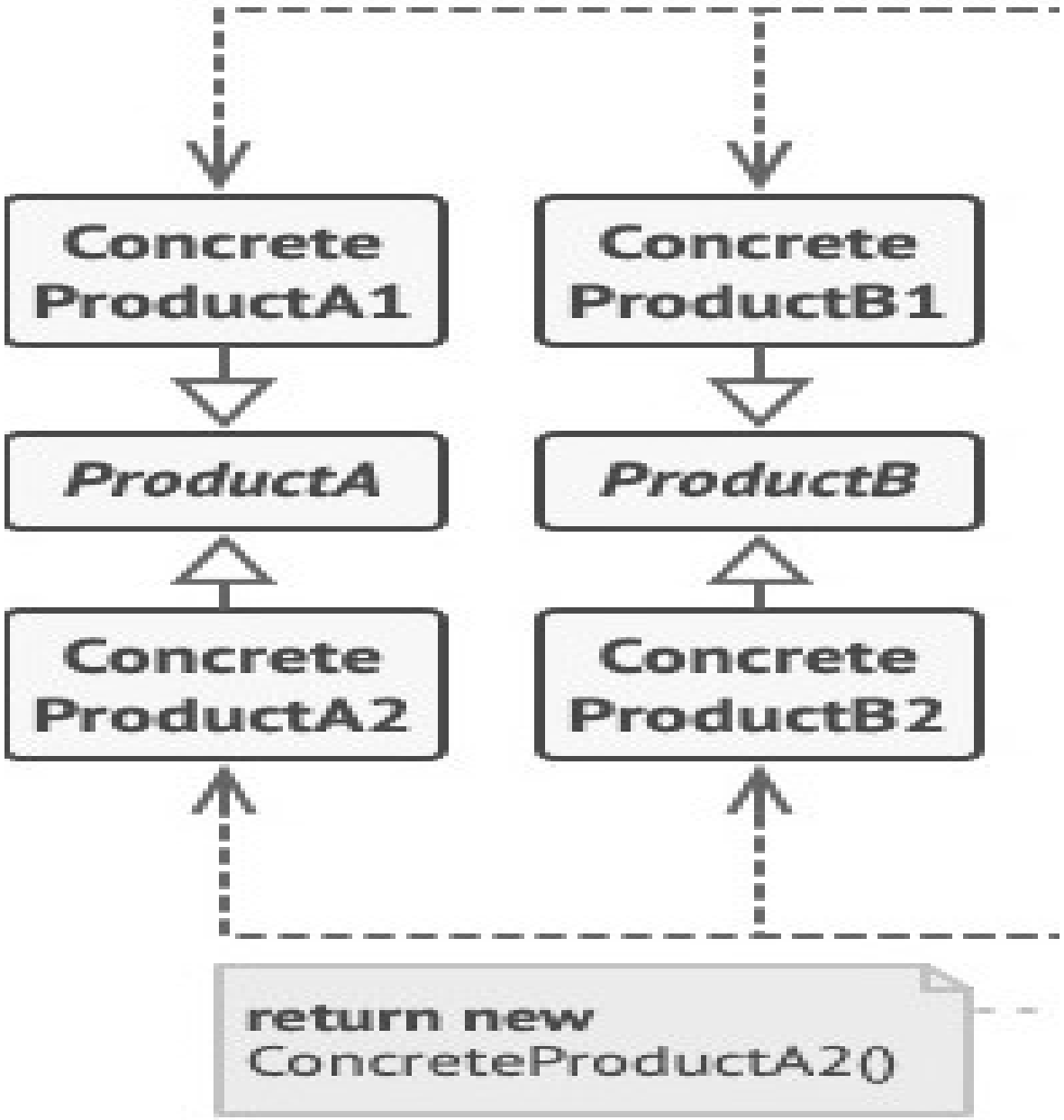
- The client code has to work with both factories and products via their respective abstract interfaces.
- This lets you change the type of a factory that you pass to the client code, as well as the product variant that the client code receives, without breaking the actual client code.


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
ProductA
ProductB
Concrete ProductA2
Concrete ProductB2
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA()
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

Structure

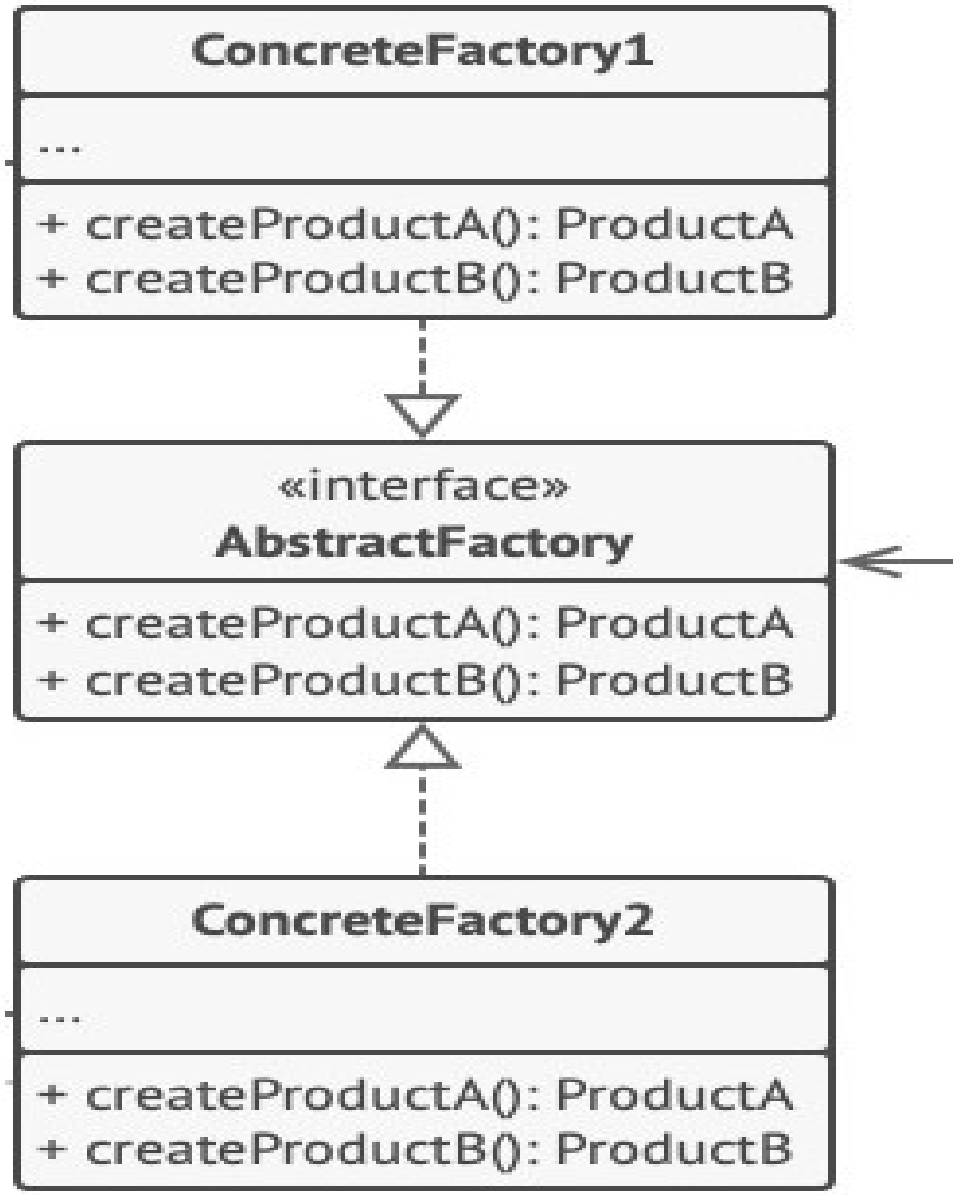
- Step 1
  - **Abstract Products** declare interfaces for a set of distinct but related products which make up a product family.

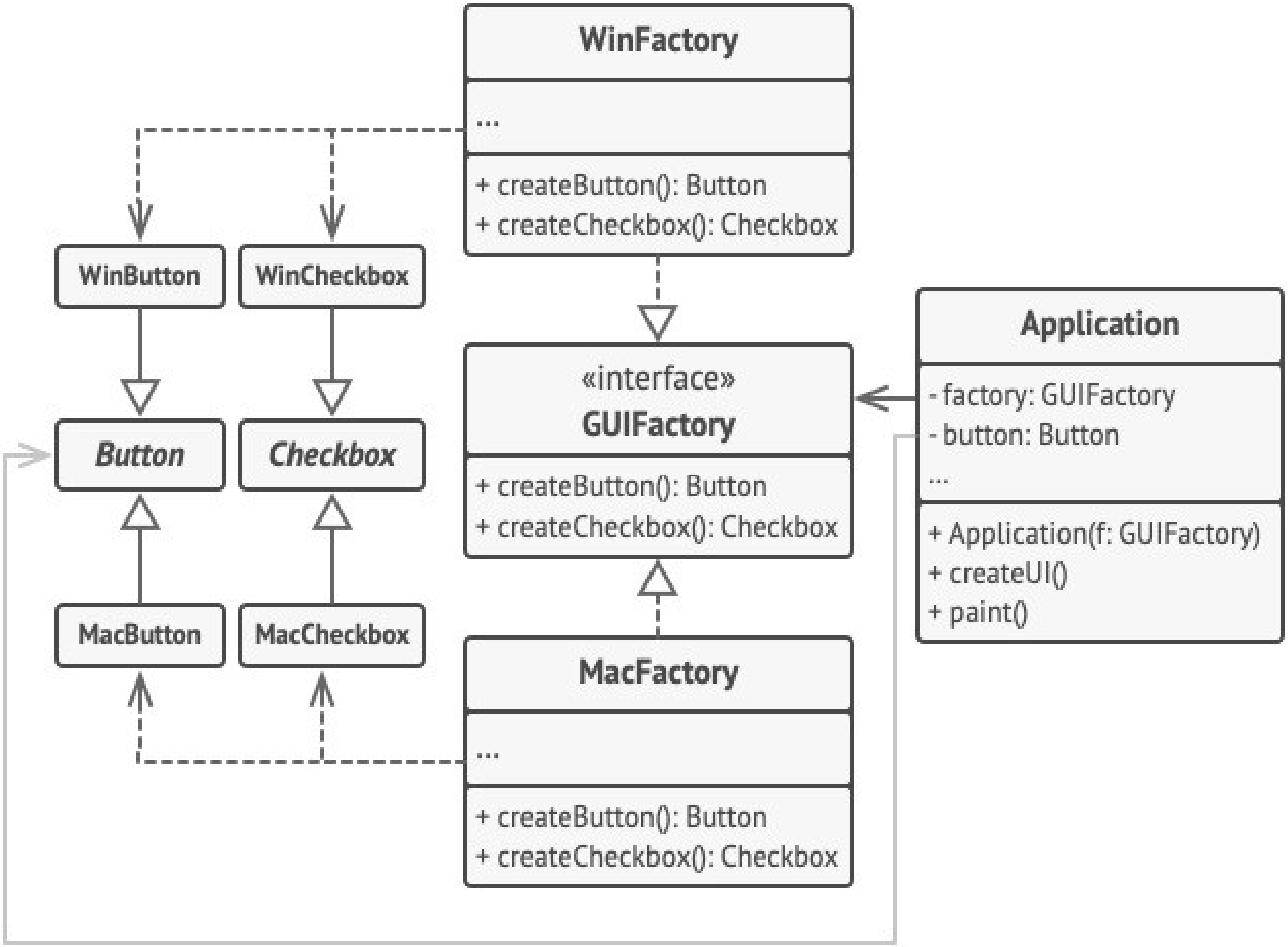
- Step 2
  - **Concrete Products** are various implementations of abstract products, grouped by variants. Each abstract product (chair/sofa) must be implemented in all given variants (Victorian/Modern).


Concrete ProductA1
Concrete ProductB1
ProductA
ProductB
Concrete ProductA2
Concrete ProductB2
return new ConcreteProductA2()

Structure

- Step 3
  - The **Abstract Factory** interface declares a set of methods for creating each of the abstract products.
- Step 4
  - **Concrete Factories** implement creation methods of the abstract factory. Each concrete factory corresponds to a specific variant of products and creates only those product variants.


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB


WinFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox
WinButton
WinCheckbox
Button
Checkbox
MacButton
MacCheckbox
«interface»
GUIFactory
+ createButton(): Button
+ createCheckbox(): Checkbox
Application
- factory: GUIFactory
- button: Button
...
+ Application(f: GUIFactory)
+ createUI()
+ paint()
MacFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox

Pseudocode

```
interface GUIFactory is
    method createButton():Button
    method createCheckbox():Checkbox

class WinFactory implements GUIFactory is
    method createButton():Button is
        return new WinButton()
    method createCheckbox():Checkbox is
        return new WinCheckbox()
class MacFactory implements GUIFactory is
    method createButton():Button is
        return new MacButton()
    method createCheckbox():Checkbox is
        return new MacCheckbox()
```

```
// interface.
interface Button is
    method paint()

// Concrete products are created by corresponding concrete
// factories.
class WinButton implements Button is
    method paint() is
        // Render a button in Windows style.

class MacButton implements Button is
    method paint() is
        // Render a button in macOS style.
```

```
interface Checkbox is
    method paint()

class WinCheckbox implements Checkbox is
    method paint() is
        // Render a checkbox in Windows style.

class MacCheckbox implements Checkbox is
    method paint() is
        // Render a checkbox in macOS style.
```

```
class Application is
    private field factory: GUIFactory
    private field button: Button
    constructor Application(factory: GUIFactory) is
        this.factory = factory
    method createUI() is
        this.button = factory.createButton()
    method paint() is
        button.paint()
```

```
class ApplicationConfigurator is
    method main() is
        config = readApplicationConfigFile()

        if (config.OS == "Windows") then
            factory = new WinFactory()
        else if (config.OS == "Mac") then
            factory = new MacFactory()
        else
            throw new Exception("Error! Unknown operating system.")

        Application app = new Application(factory)
```

References

- https://refactoring.guru/design-patterns/abstract-factory