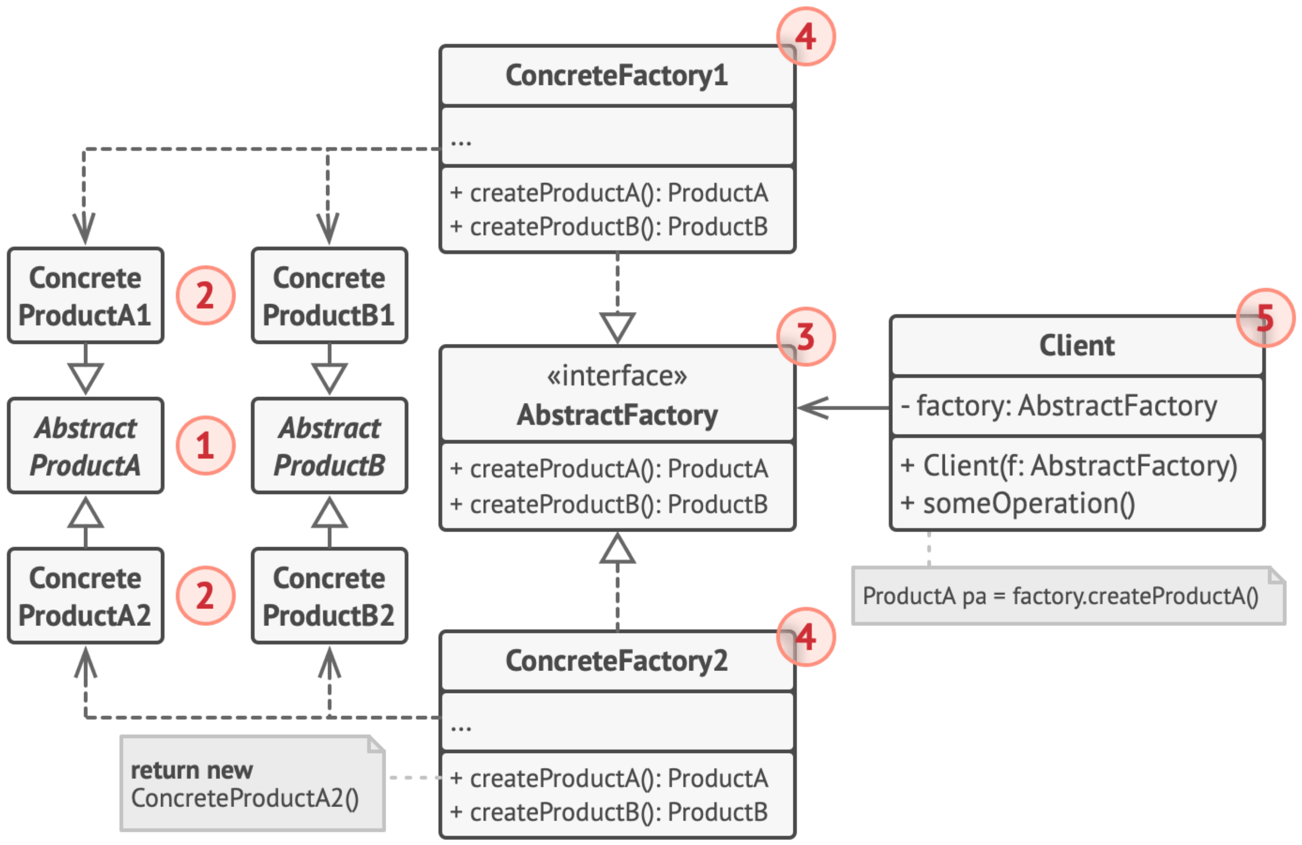
SUMMER SALE

factory's class, nor does it matter what kind of chair it gets. Whether it's a Modern model or a Victorian-style chair, the client must treat all chairs in the same manner, using the abstract `Chair` interface. With this approach, the only thing that the client knows about the chair is that it implements the `sitOn` method in some way. Also, whichever variant of the chair is returned, it'll always match the type of sofa or coffee table produced by the same factory object.

There's one more thing left to clarify: if the client is only exposed to the abstract interfaces, what creates the actual factory objects? Usually, the application creates a concrete factory object at the initialization stage. Just before that, the app must select the factory type depending on the configuration or the environment settings.

## Structure


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
Abstract ProductA
Abstract 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()
1
2
2
3
4
4
5

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

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).

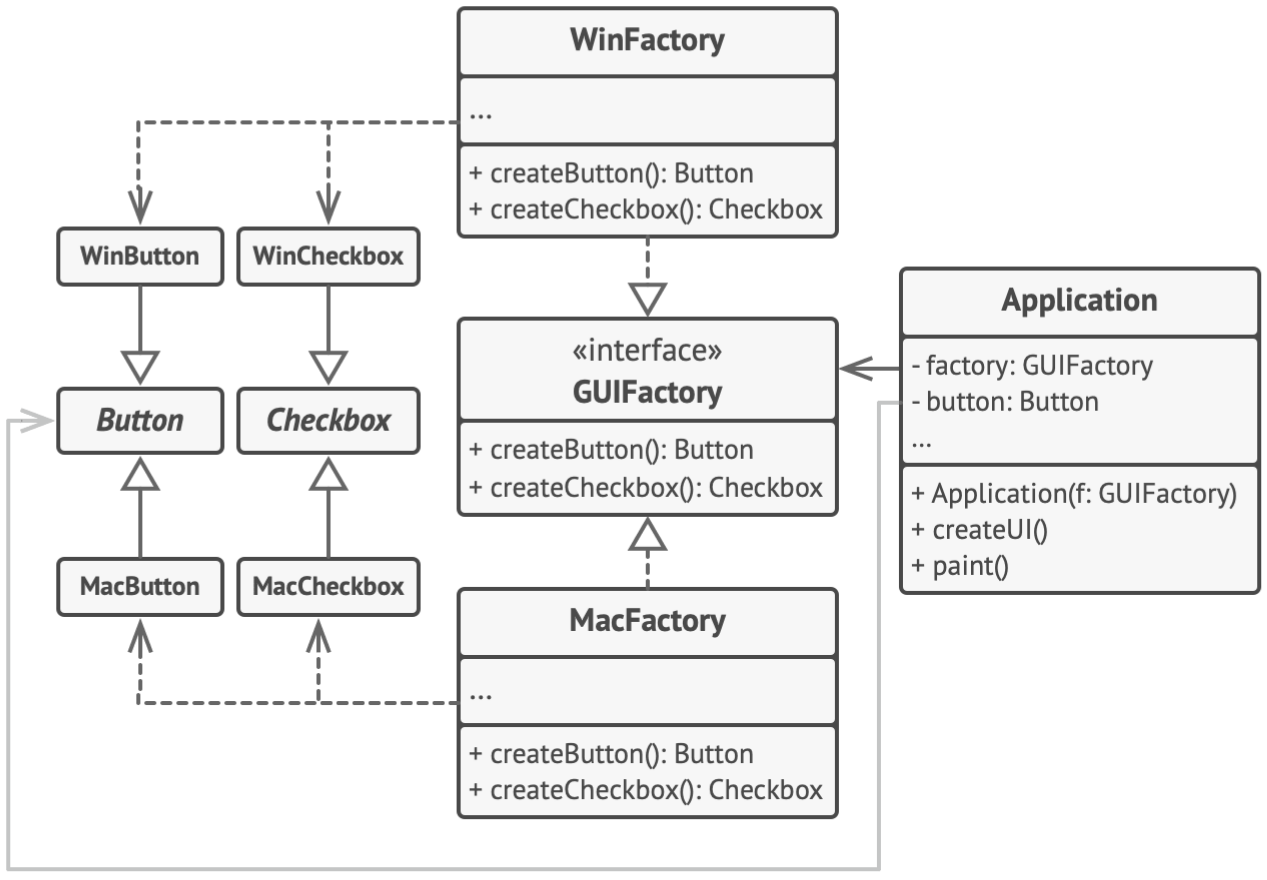
3. The **Abstract Factory** interface declares a set of methods for creating each of the abstract products.

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.

5. Although concrete factories instantiate concrete products, signatures of their creation methods must return corresponding *abstract* products. This way the client code that uses a factory doesn't get coupled to the specific variant of the product it gets from a factory. The **Client** can work with any concrete factory/product variant, as long as it communicates with their objects via abstract interfaces.

## # Pseudocode

This example illustrates how the **Abstract Factory** pattern can be used for creating cross-platform UI elements without coupling the client code to concrete UI classes, while keeping all created elements consistent with a selected operating system.



*The cross-platform UI classes example.*