COMP2511

Creational Patterns:

Factory Method
Abstract Factory Pattern

Prepared by

Dr. Ashesh Mahidadia

Creational Patterns

Creational Patterns

Creational patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

- Factory Method
  - provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
- Abstract Factory
  - let users produce families of related objects without specifying their concrete classes.
- Builder
  - let users construct complex objects step by step. The pattern allows users to produce different types and representations of an object using the same construction code.
- Singleton
  - Let users ensure that a class has only one instance, while providing a global access point to this instance.

Factory Method

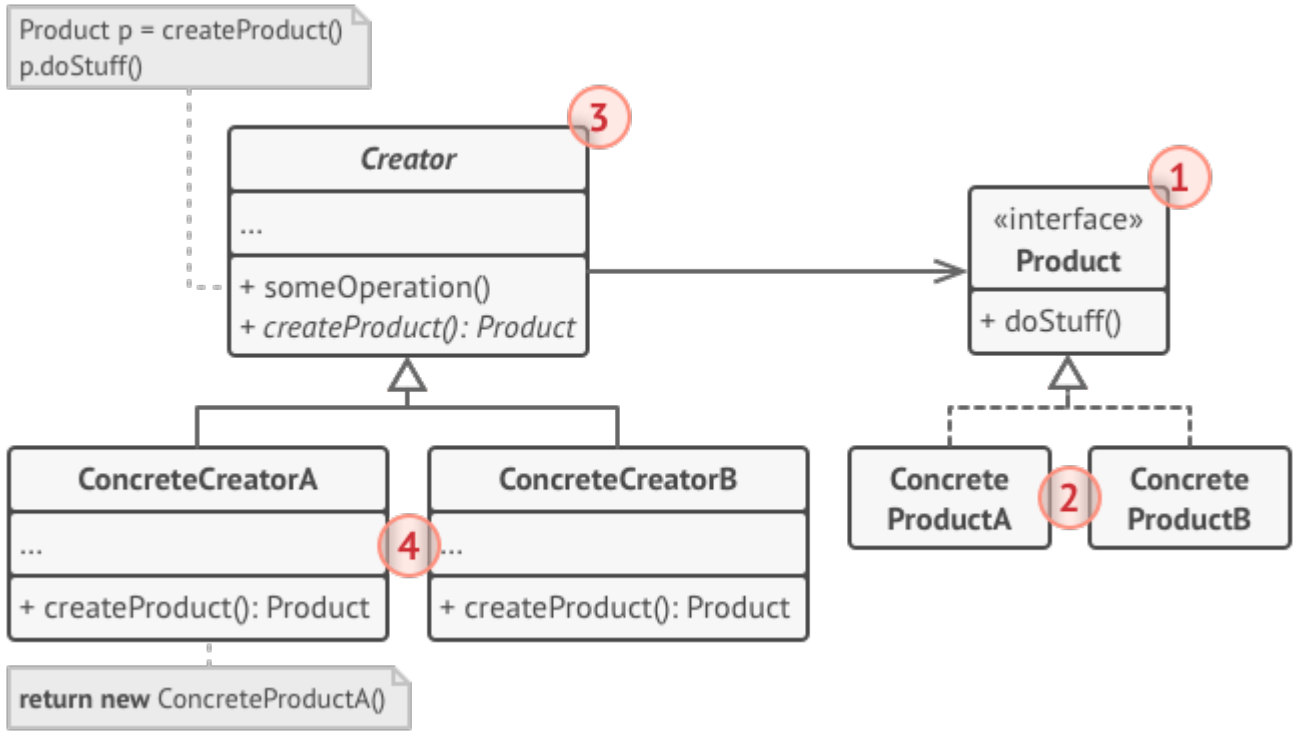
Factory Method

- Factory Method is a creational design pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.

- **Problem:**
  - creating an object directly within the class that requires (uses) the object is inflexible
  - it commits the class to a particular object and
  - makes it impossible to change the instantiation independently from (without having to change) the class.

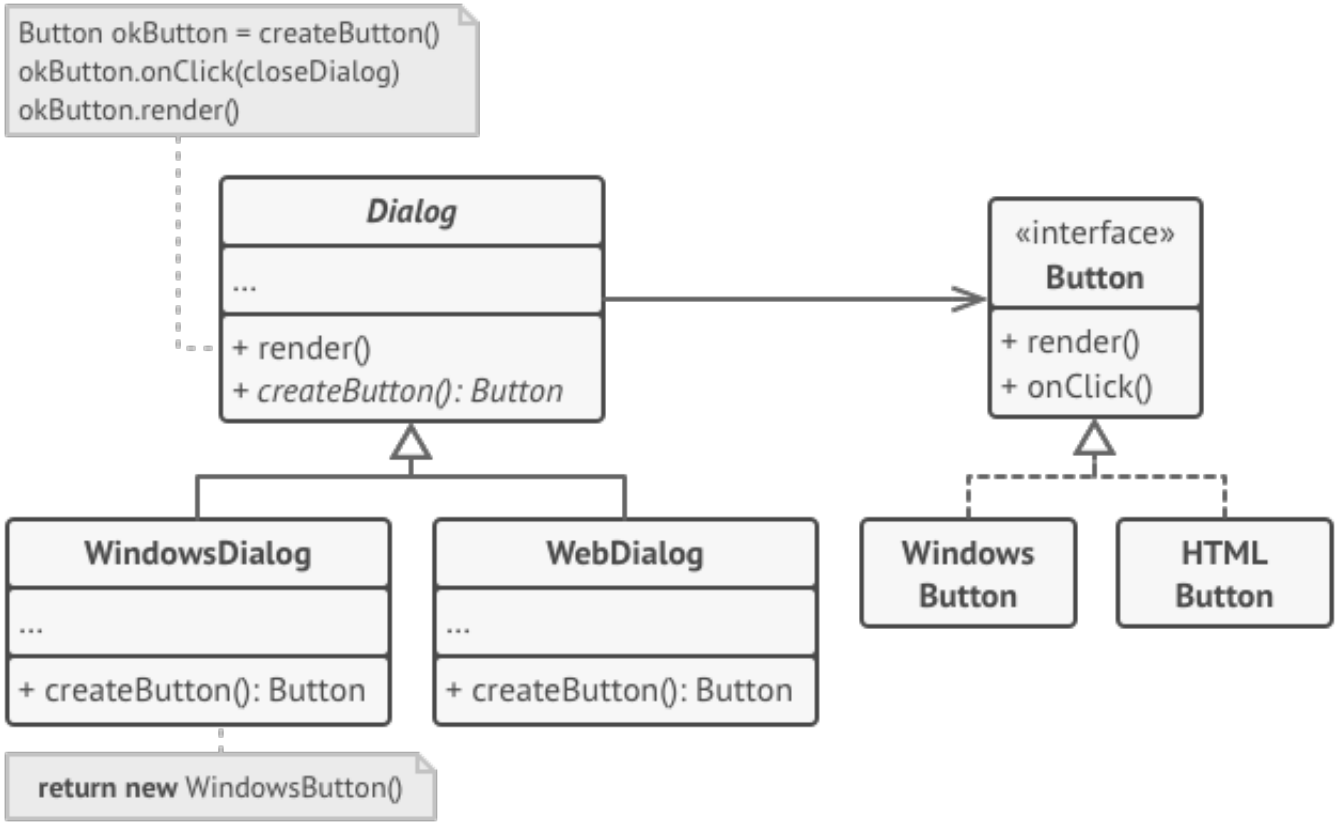
- **Possible Solution**:
  - Define a **separate** operation (factory **method**) for creating an object.
  - Create an object by calling a factory method.
  - This enables writing of subclasses to change the way an object is created (to redefine which class to instantiate).

Factory Method : Structure



1. The Product declares the interface, which is common to all objects that can be produced by the creator and its subclasses.
2. Concrete Products are different implementations of the product interface.
3. The Creator class declares the factory method that returns new product objects.
4. Concrete Creators override the base factory method so it returns a different type of product.

Factory Method : Example



Example in **Java (MUST read)**:

https://refactoring.guru/design-patterns/factory-method/java/example

Factory Method

For more, read the following:

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

Abstract Factory Pattern

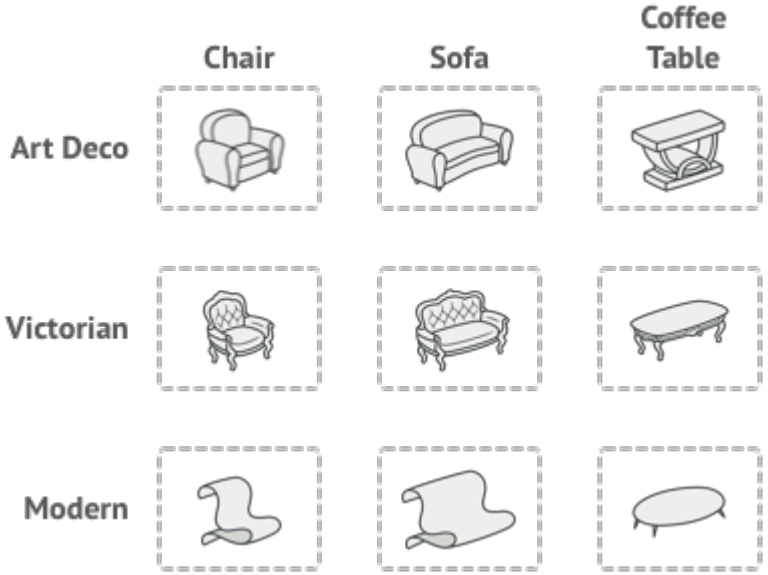
Abstract Factory Pattern

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

Problem:

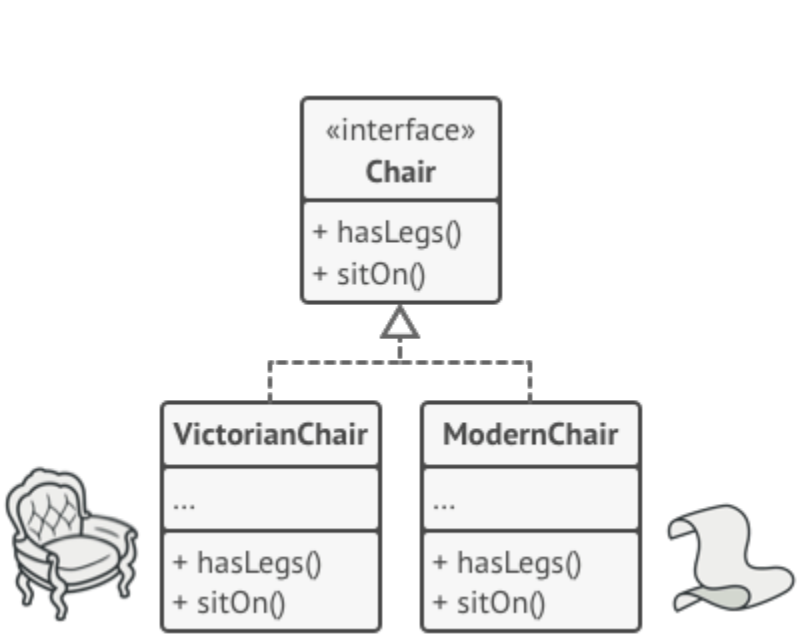
Imagine that you're creating a furniture shop simulator. Your code consists of classes that represent:

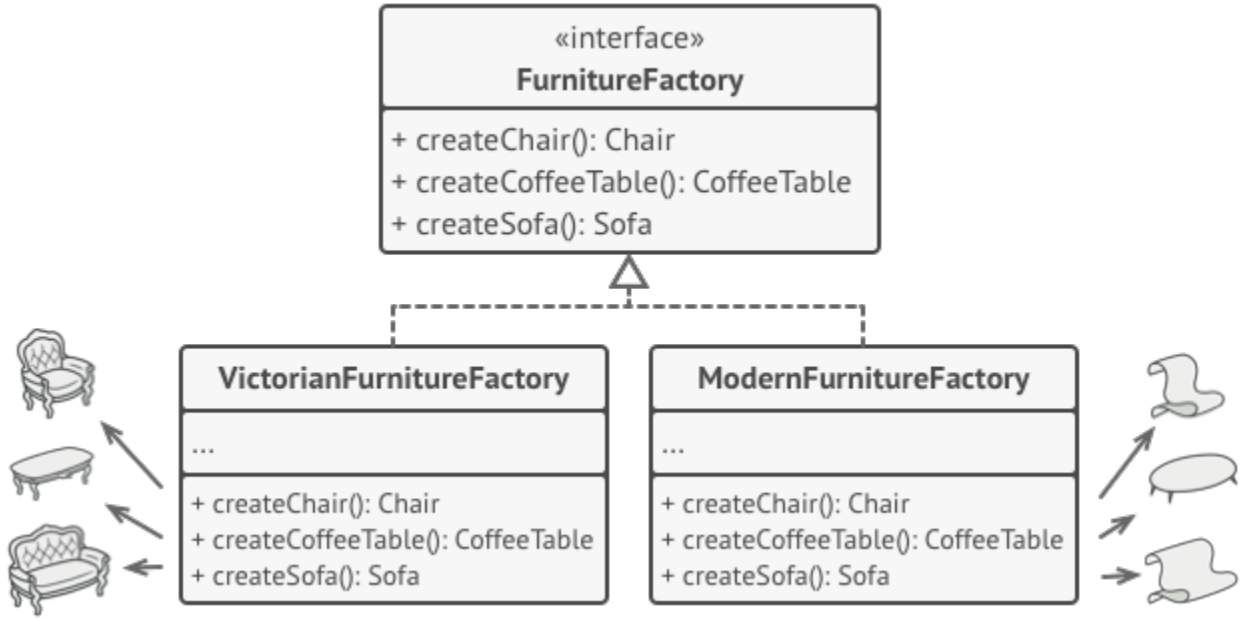
- A family of related products, say: Chair + Sofa + CoffeeTable.
- Several variants of this family.
- For example, products Chair + Sofa + CoffeeTable are available in these variants:



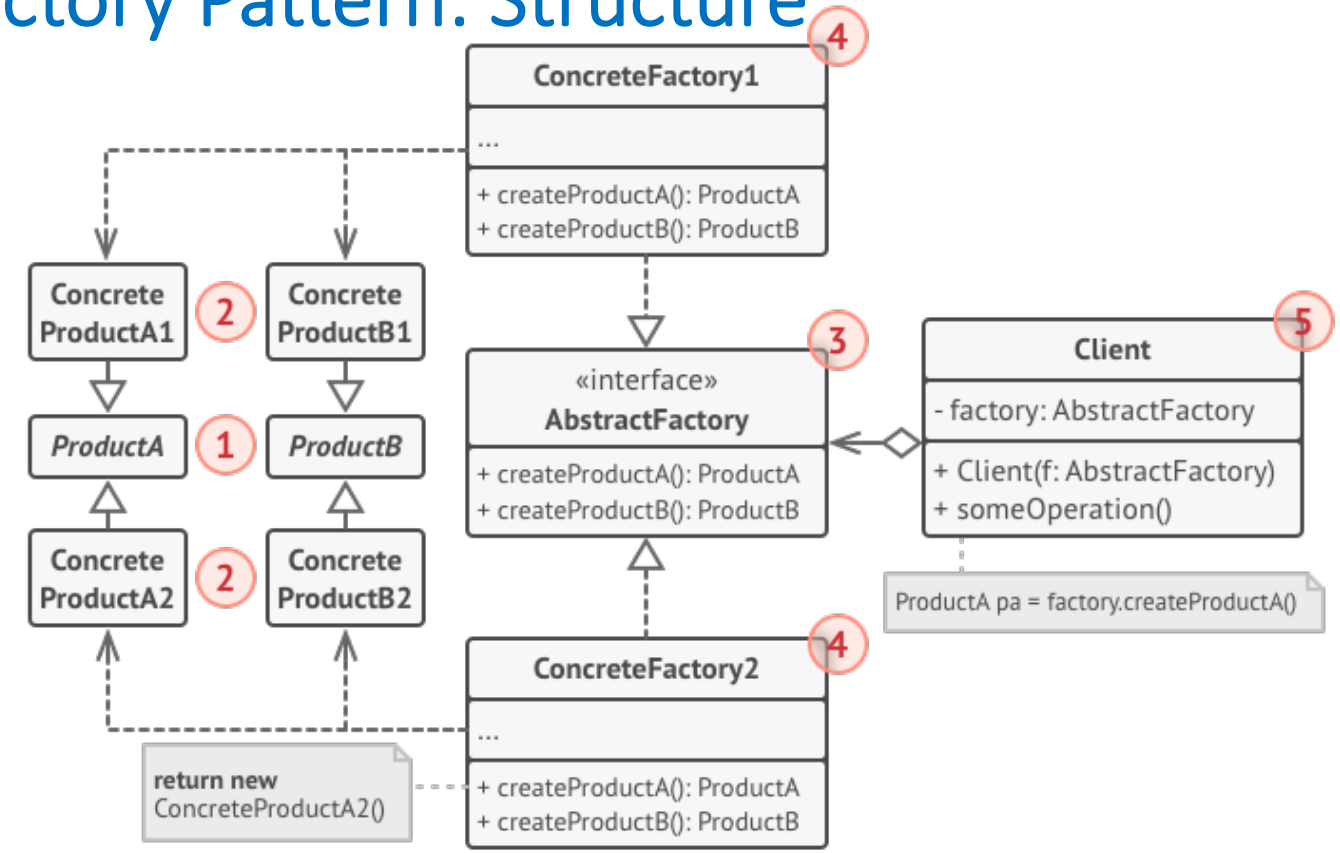
Abstract Factory Pattern:

Possible **Solution**:



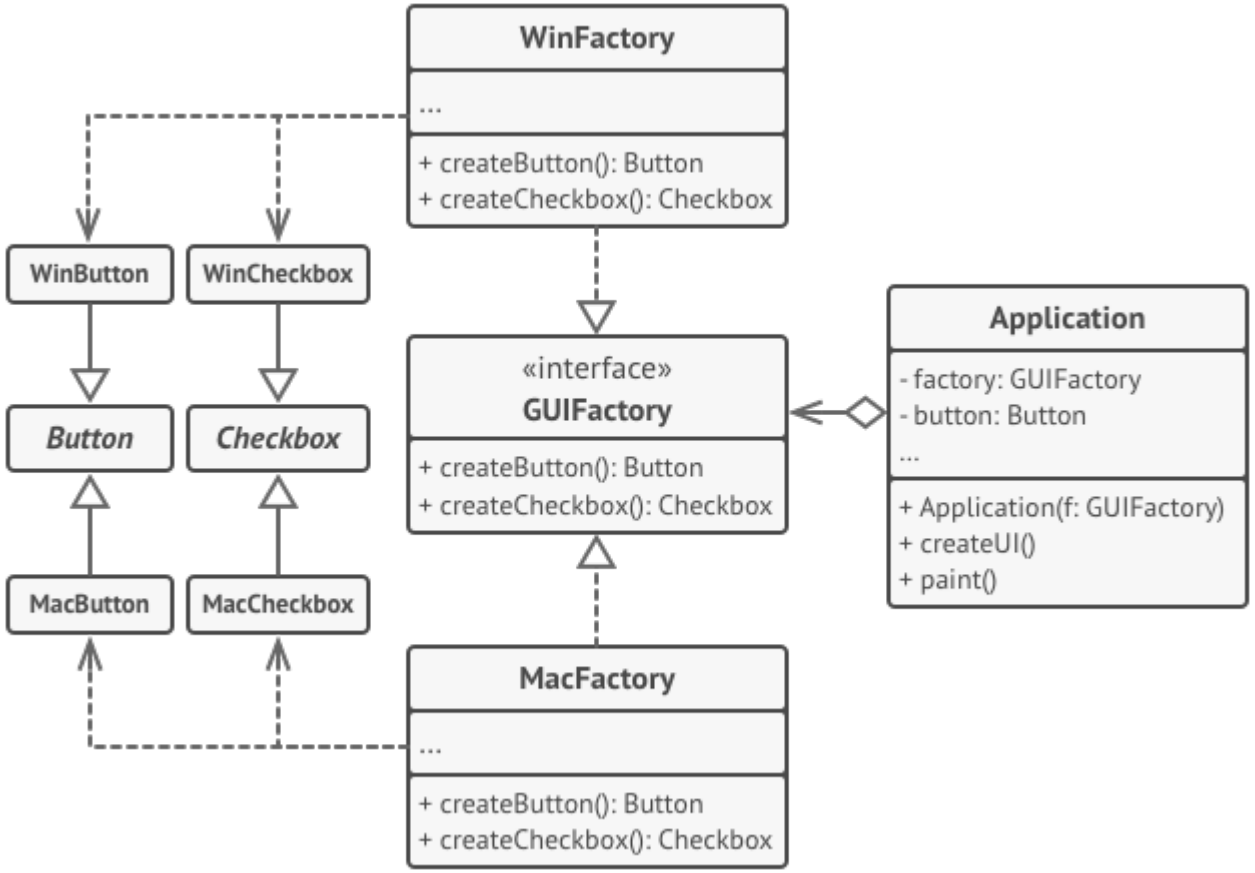


Abstract Factory Pattern: Structure



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. The **Client** can work with any concrete factory/product variant, as long as it communicates with their objects via abstract interfaces.

Abstract Factory Pattern: Example



Example in **Java (MUST read)**:

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

Abstract Factory Pattern

For more, read the following:

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

End