Brief of UML : inheritance : triangle symbol, composition (has a) diamond symbol

A diagram of a class

Description automatically generated

-The code of the same pattern applied to two different programs may be different.

-On the other hand, a pattern is more like a blueprint: you can see what the result and its features are, but the exact order of implementation is up to you.

**-Unjustified use of knowing a single design pattern :**

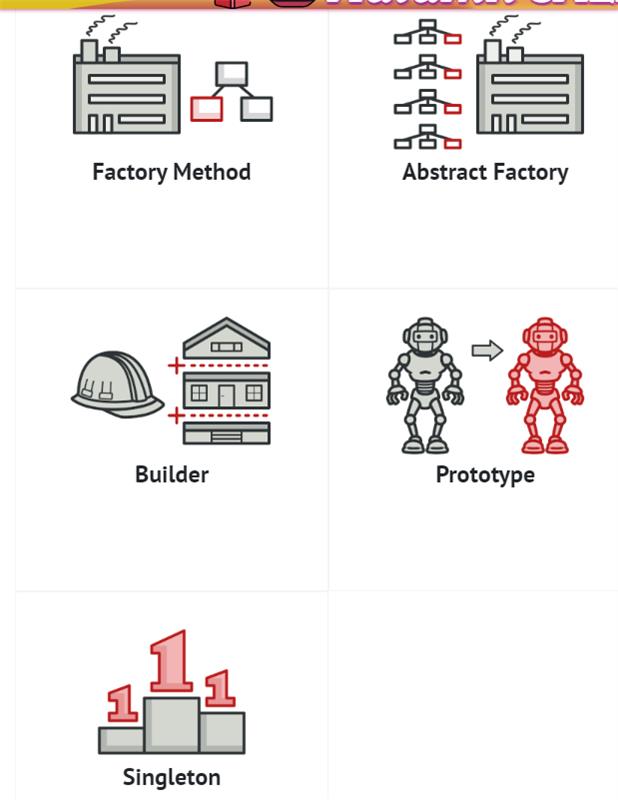
If all you have is a hammer, everything looks like a nail.

-The most basic and low-level patterns are often called *idioms*. They usually apply only to a single programming language.

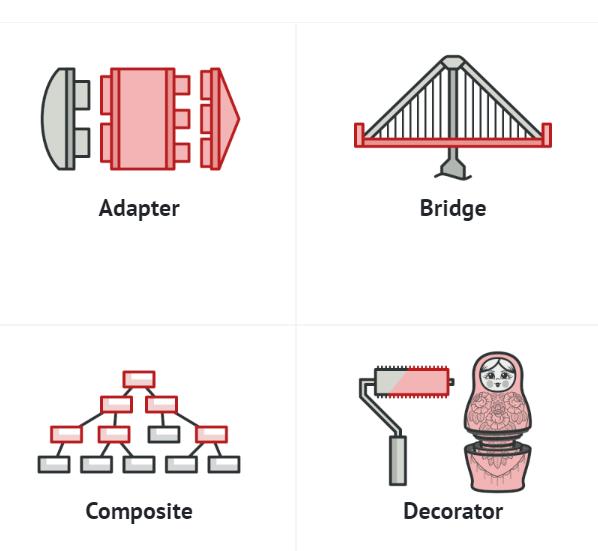
-The most universal and high-level patterns are *architectural patterns*. Developers can implement these patterns in virtually any language. Unlike other patterns, they can be used to design the architecture of an entire application.

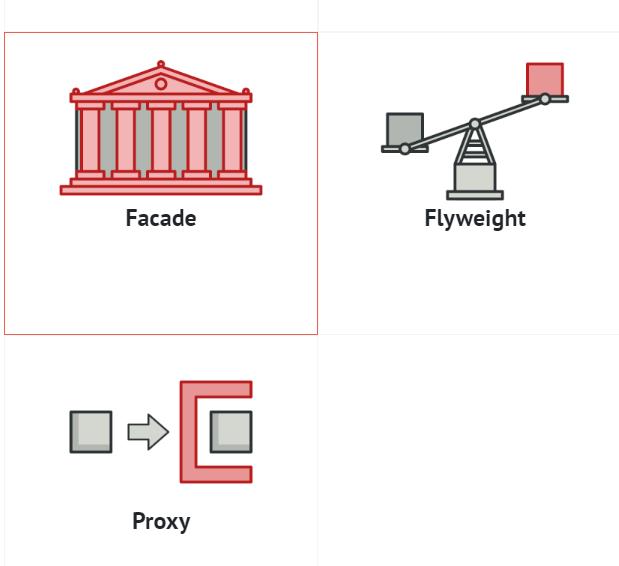
* **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
* **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.
* **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects.

creational :

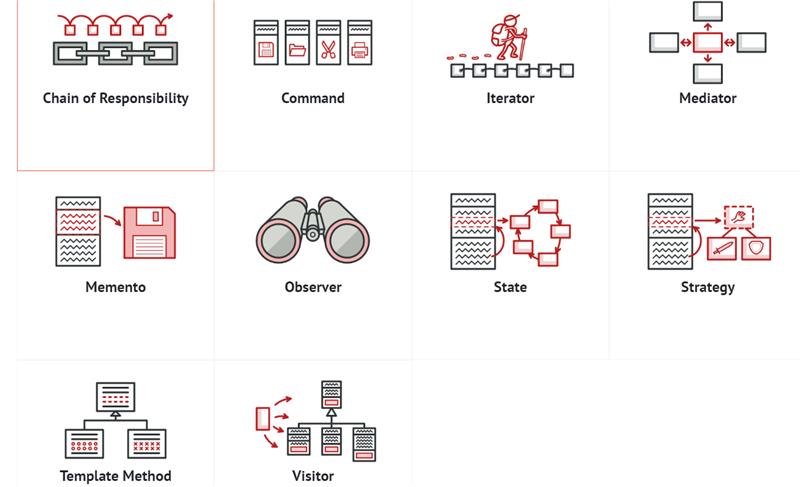


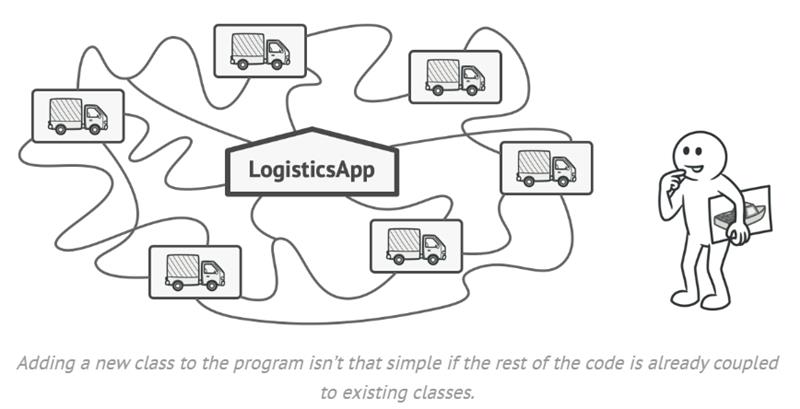
structural :





Behavioural :

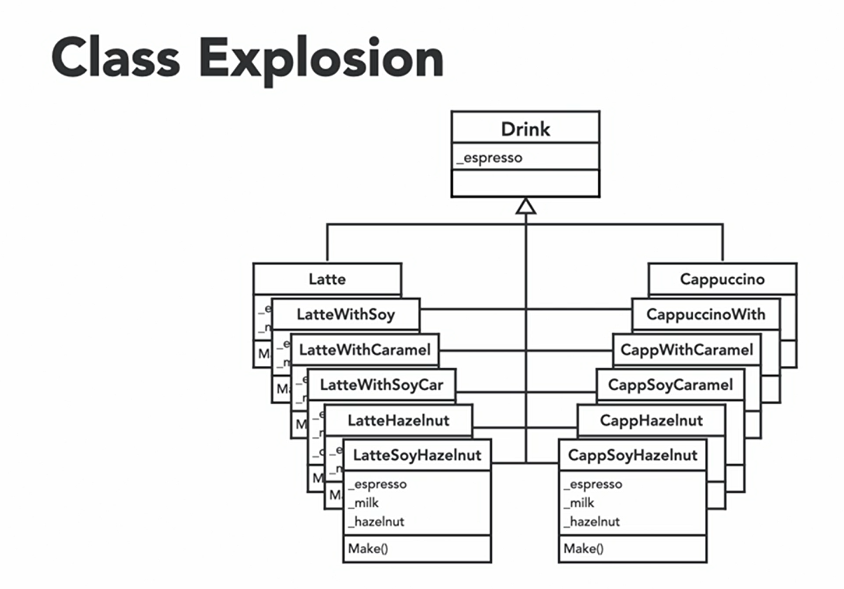




 The Factory Method pattern suggests that you replace direct object construction calls (using the new operator) with calls to a special *factory* method. Don’t worry: the objects are still created via the new operator, but it’s being called from within the factory method. Objects returned by a factory method are often referred to as *products.*

At first glance, this change may look pointless: we just moved the constructor call from one part of the program to another. However, consider this: now you can override the factory method in a subclass and change the class of products being created by the method.

Disadvantage of deriving classes (Inheritance) for every variation:



Solution to this problem is to use composition along with Inheritance.

if there's a virtual method in baseclass, that function must be overridden in all derived classes, otherwise we'll get a compiler error.

The base class of virtual methods is called abstract class and derived classes are called concrete classes.

Factory method : There’s a slight limitation though: subclasses may return different types of products only if these products have a common base class or interface. Also, the factory method in the base class should have its return type declared as this interface.

The Creator class declares the factory method that returns new product objects. It’s important that the return type of this method matches the product interface.

Concrete Creators override the base factory method so it returns a different type of product.

Note that the factory method doesn’t have to create new instances all the time. It can also return existing objects from a cache, an object pool, or another source.

Note that, despite its name, the creator's primary responsibility isn't creating products. It usually contains some core business logic that relies on product objects returned by the factory method. Subclasses can indirectly change that business logic by overriding the factory method and returning a different type of product from it.

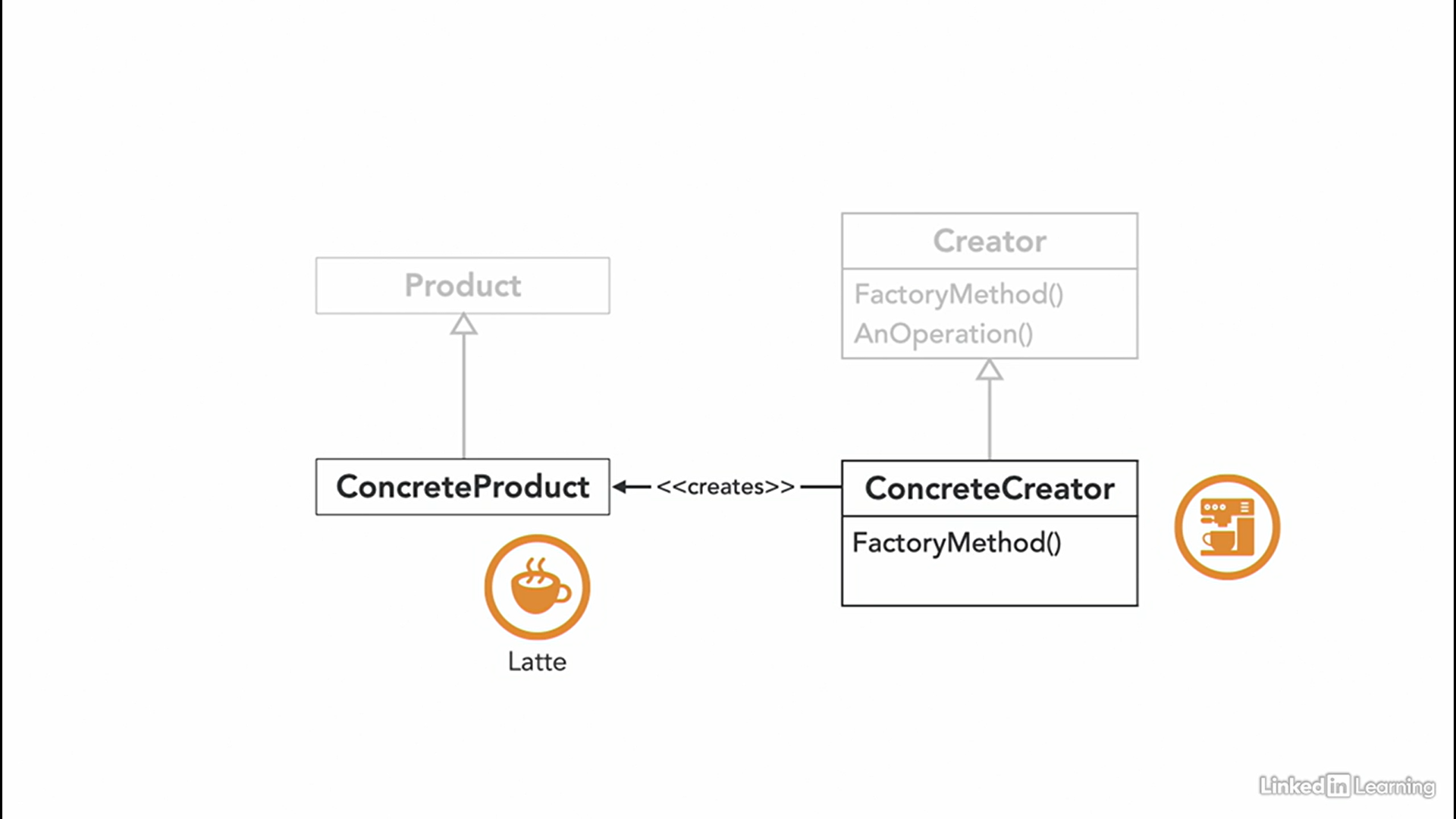
* **Use the Factory Method when you don’t know beforehand the exact types and dependencies of the objects your code should work with.**

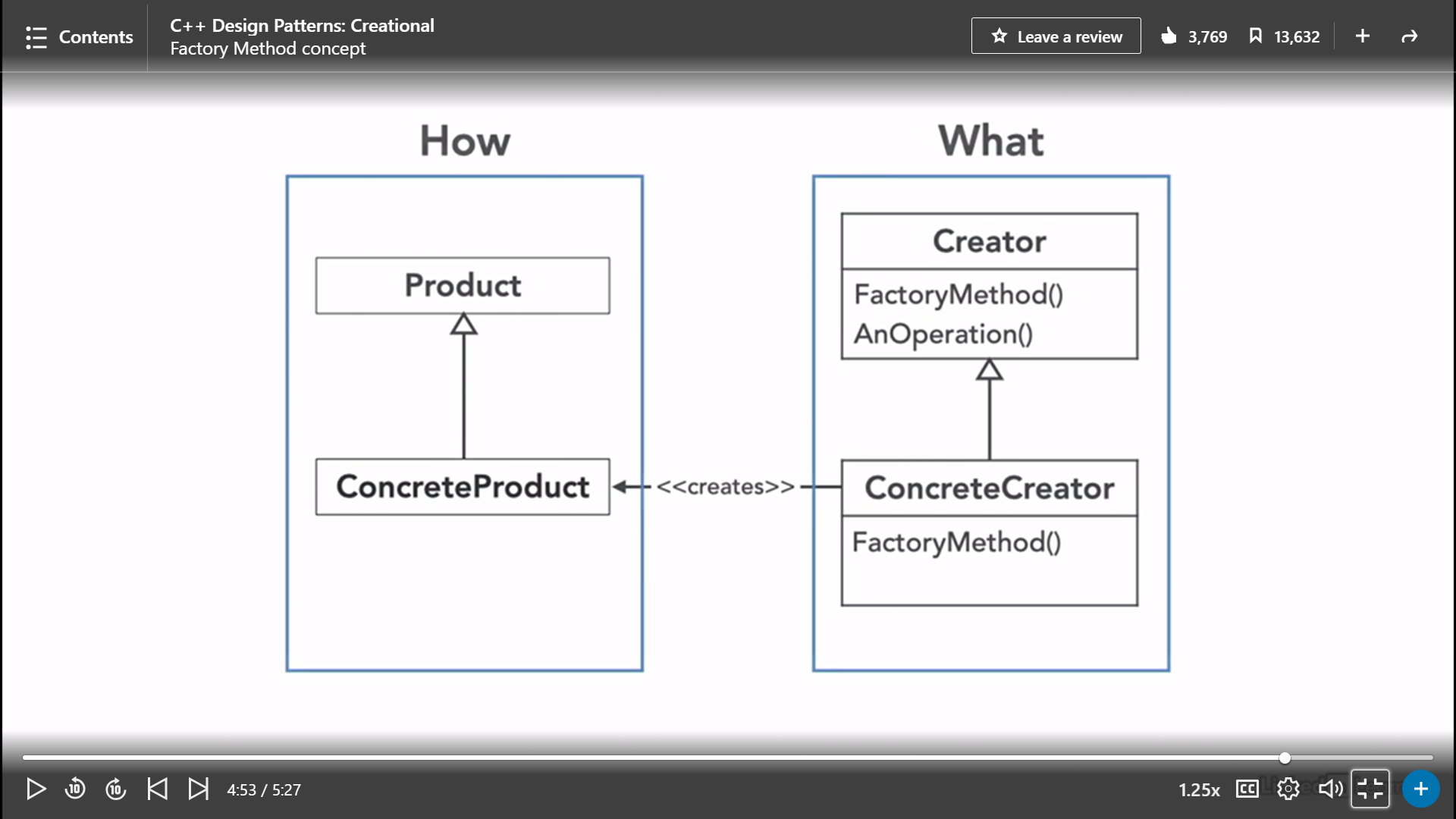
The Factory Method separates product construction code from the code that actually uses the product. Therefore it’s easier to extend the product construction code independently from the rest of the code.

For example, to add a new product type to the app, you’ll only need to create a new creator subclass and override the factory method in it.

* **Use the Factory Method when you want to provide users of your library or framework with a way to extend its internal components.**
* **Use the Factory Method when you want to save system resources by reusing existing objects instead of rebuilding them each time.**

Probably the most obvious and convenient place where this code could be placed is the constructor of the class whose objects we’re trying to reuse. However, a constructor must always return **new objects** by definition. It can’t return existing instances.

Factory Method : coffee machine example :  
  




Factory method implementation:

1. U can either ask in the constructor from user, which kind of object it wants to create.
2. Or u can make constructor as private, and force the user to call one of the public functions of the class to create object of their choice. The public functions will be using the private constructor internally.