CSci 3081W: Program Design and Development

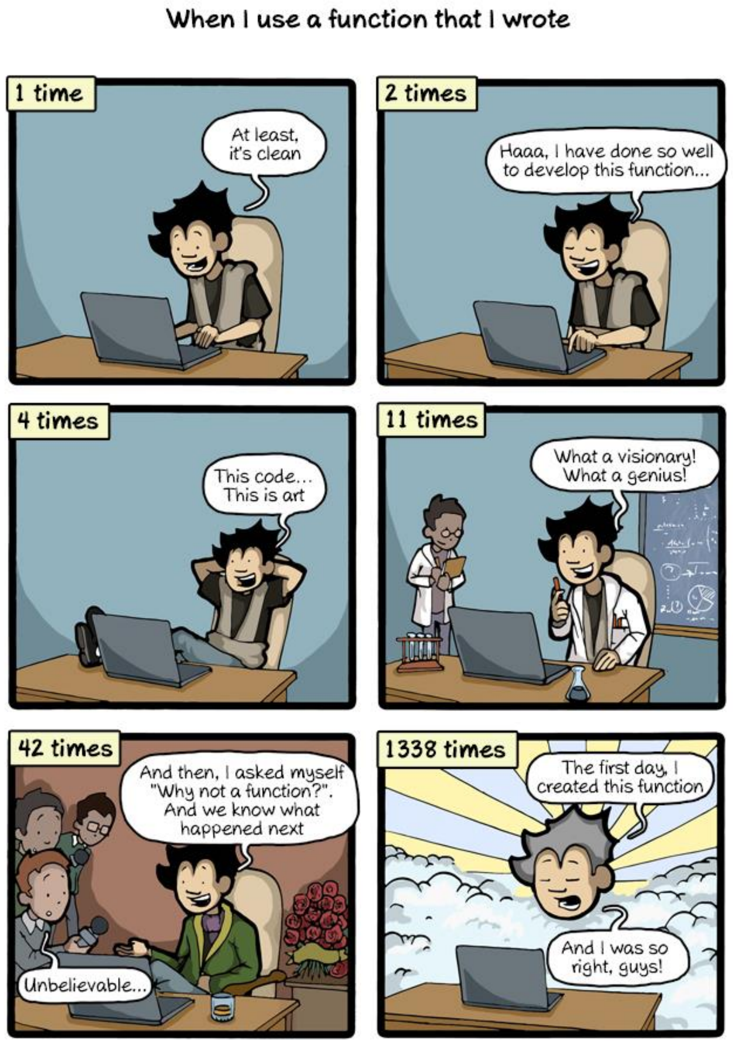
Lecture 8 – Creational Design Patterns
Factory Method Pattern

Creational Design Patterns

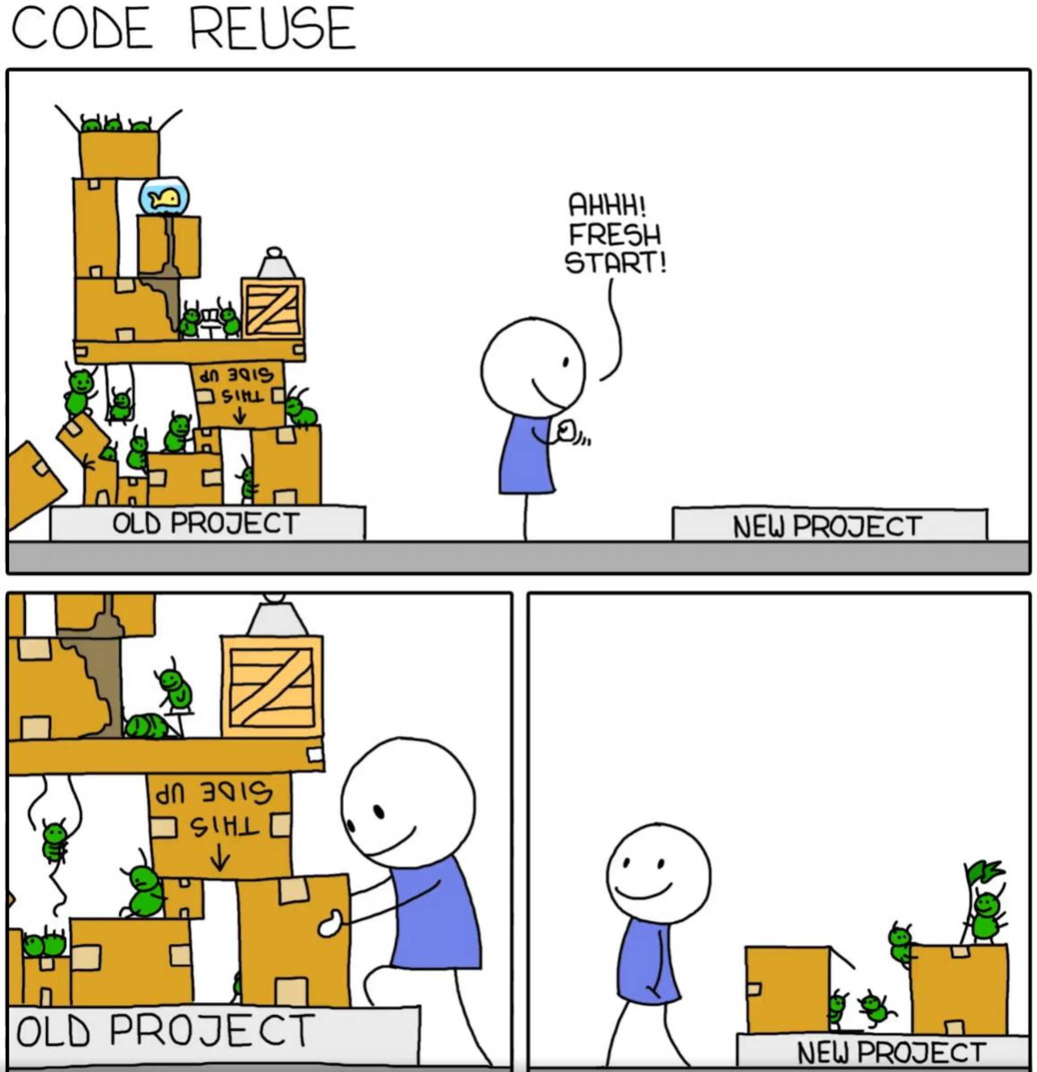
Various object creation mechanisms

Why? To increase flexibility and reuse of existing code

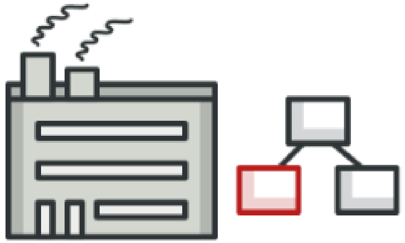
Code Reuse


When I use a function that I wrote
1 time
At least, it's clean
2 times
Haaa, I have done so well to develop this function...
4 times
This code... This is art
11 times
What a visionary! What a genius!
42 times
And then, I asked myself "Why not a function?". And we know what happened next
Unbelievable...
1338 times
The first day, I created this function
And I was so right, guys!

Code Reuse


CODE REUSE
AHHH! FRESH START!
THIS SIDE UP
OLD PROJECT
NEW PROJECT
THIS SIDE UP
OLD PROJECT
NEW PROJECT

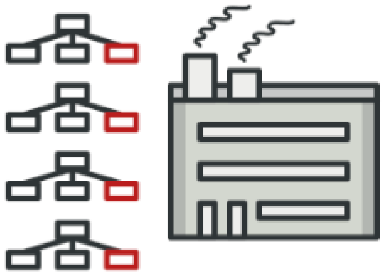
Creational Design Patterns



Factory Method

Provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

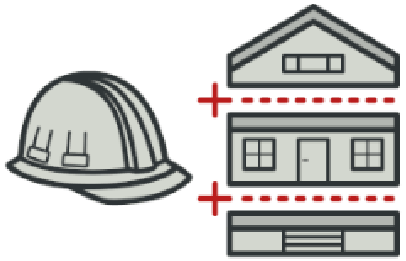
Creational Design Patterns



Abstract Factory

Lets you produce families of related objects without specifying their concrete classes.

Creational Design Patterns

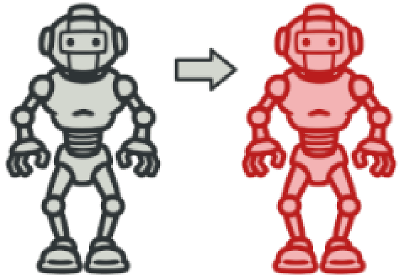


Builder

Lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

We are not doing this one

Creational Design Patterns

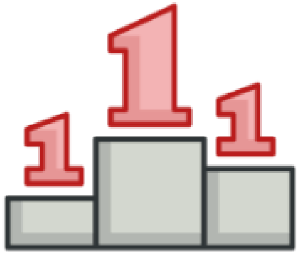


Prototype

Lets you copy existing objects without making your code dependent on their classes.

We are not doing this one

Creational Design Patterns



Singleton

Lets you ensure that a class has only one instance, while providing a global access point to this instance.

Factory Method aka Virtual Constructor

Definition:

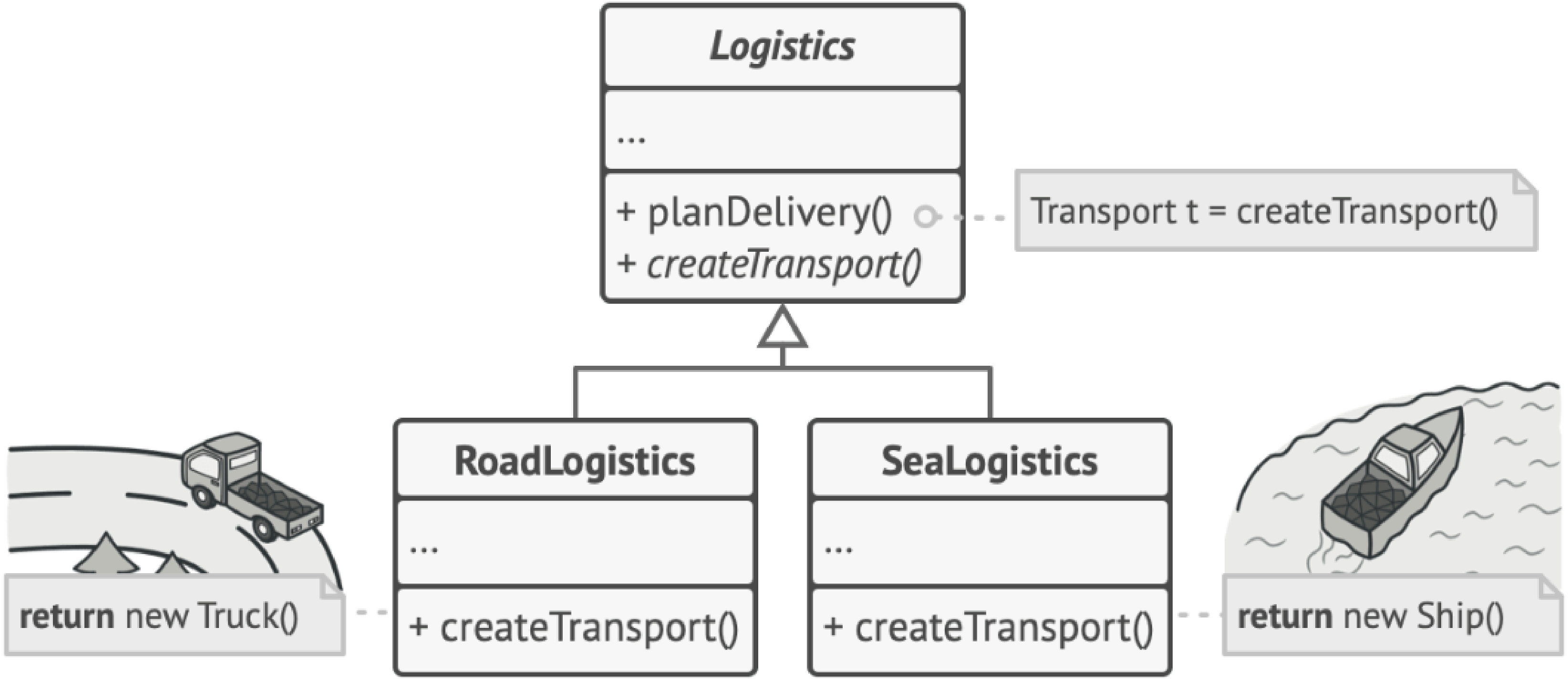
Provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created

Problem

LogisticsApp

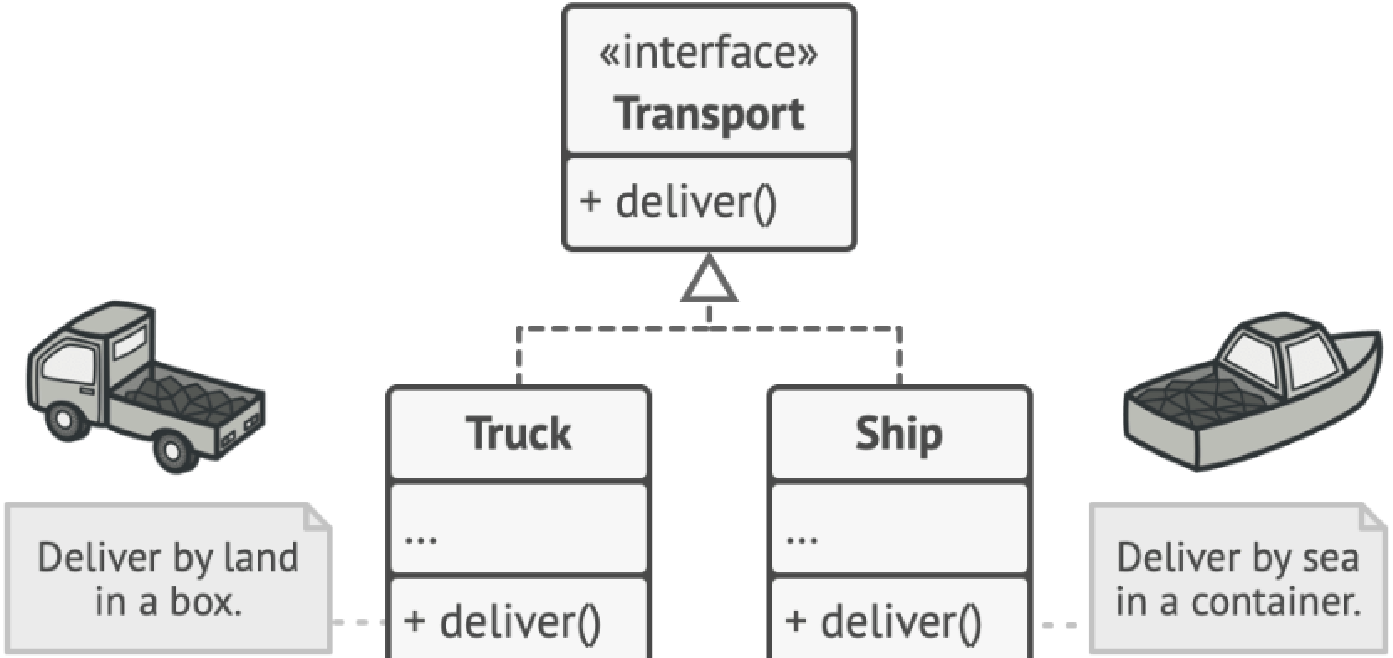
Adding a new class to the program isn't that simple if the rest of the code is already coupled to existing classes.

Solution



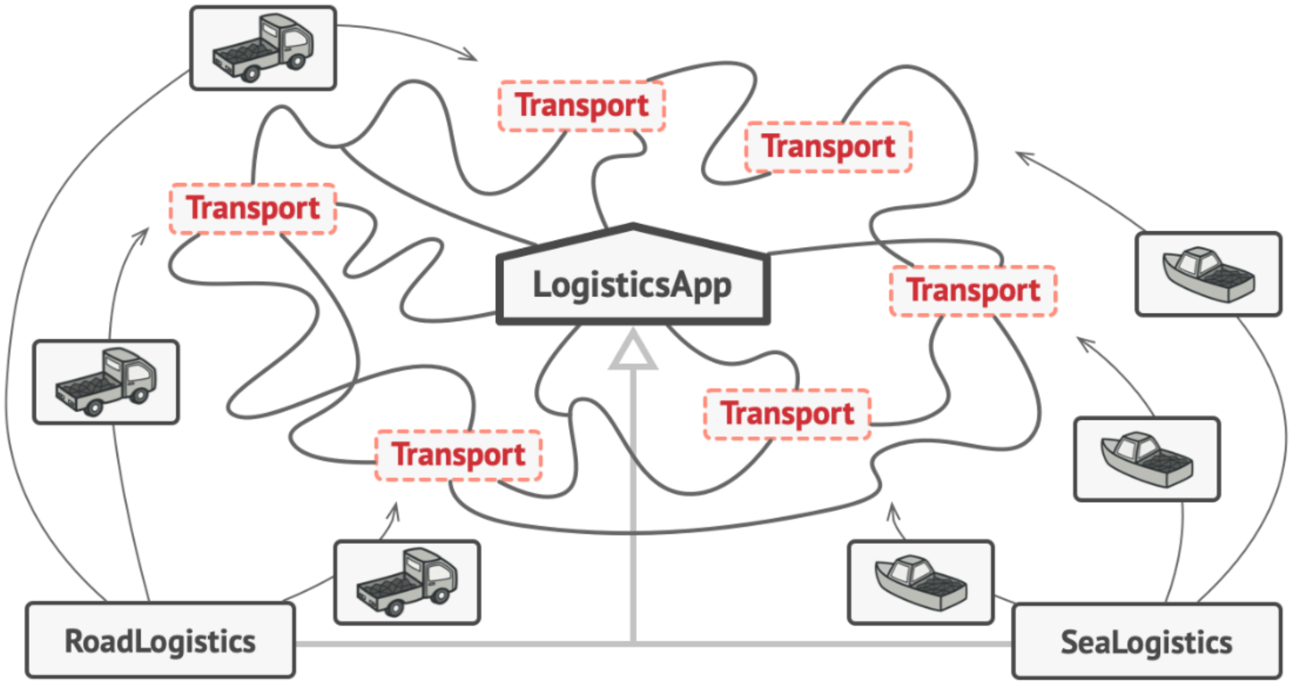
Subclasses can alter the class of objects being returned by the factory method.

Solution



All products must follow the same interface.

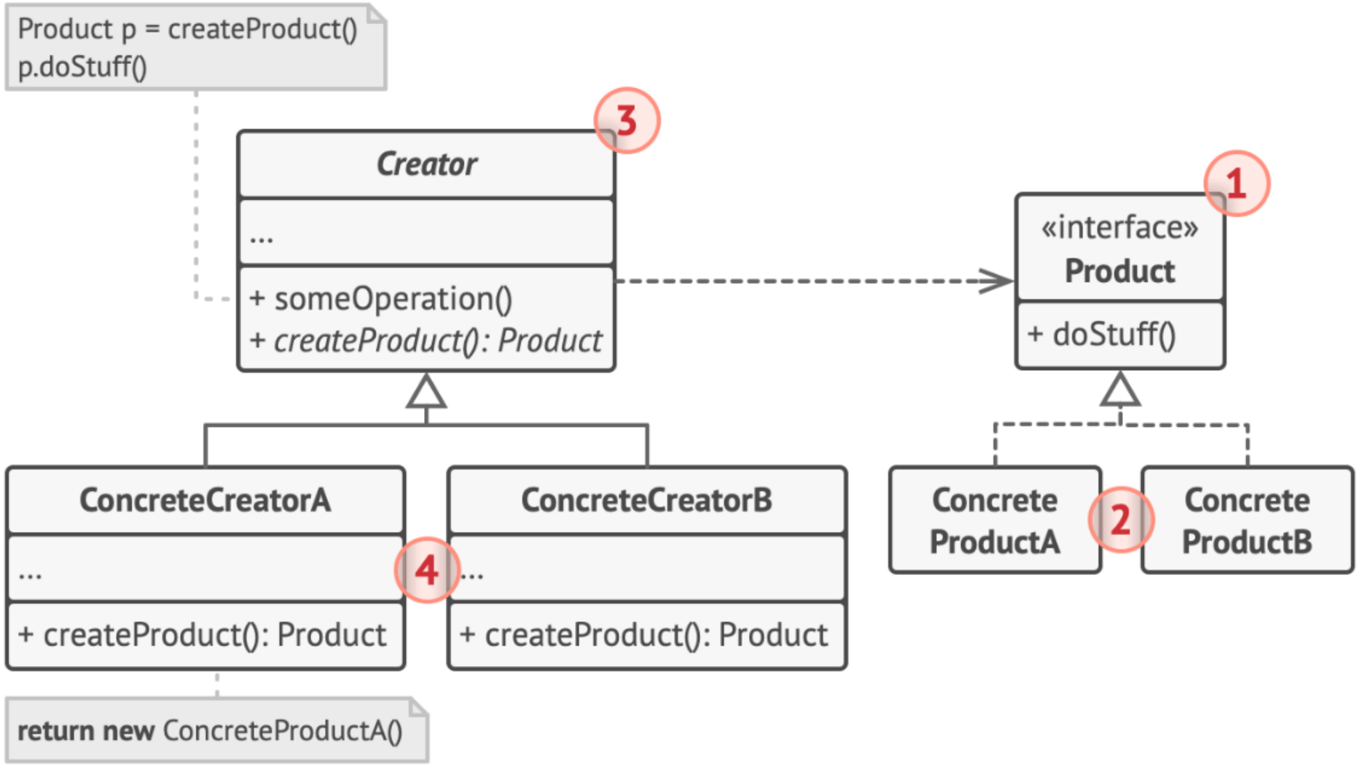
Solution



As long as all product classes implement a common interface, you can pass their objects to the client code without breaking it.

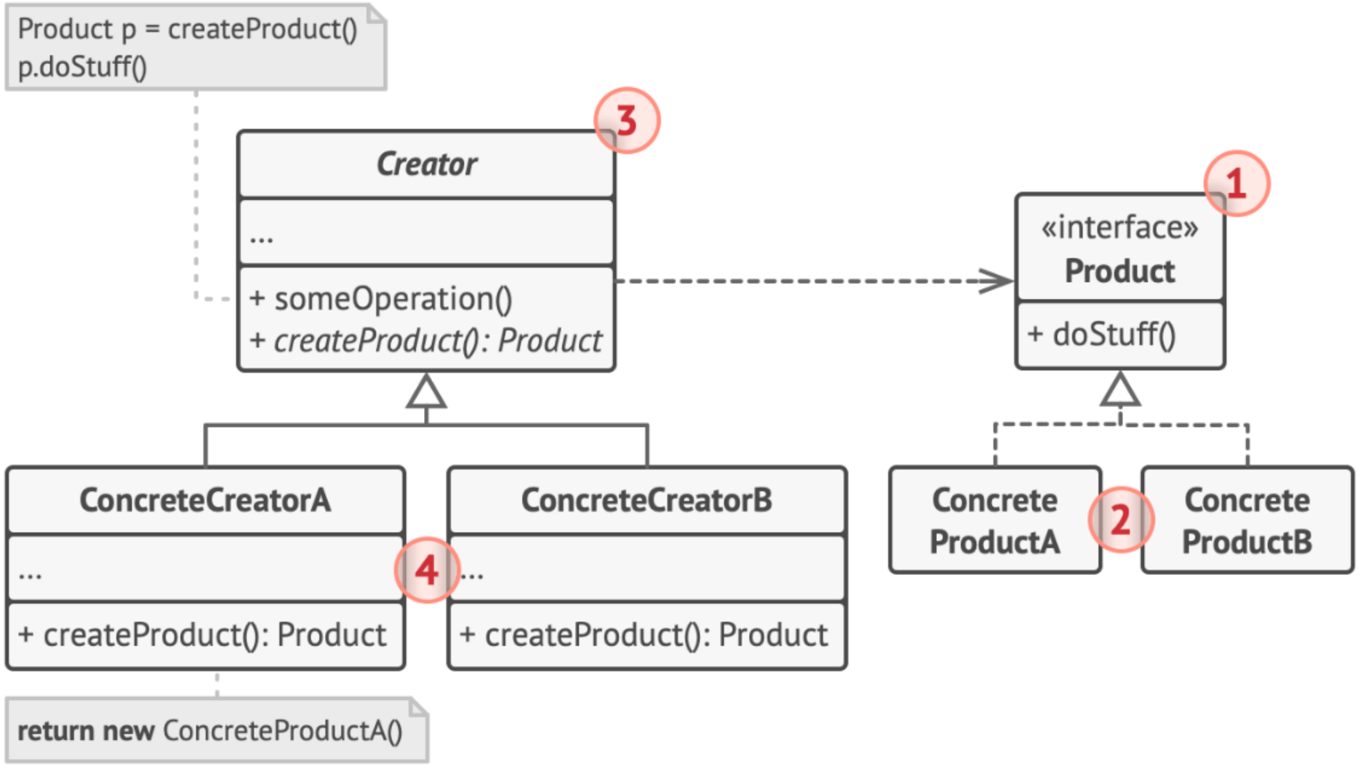
UML Class Diagram Structure

1. The **Product** declares the interface, which is common to all objects that can be produced by the creator and its subclasses.



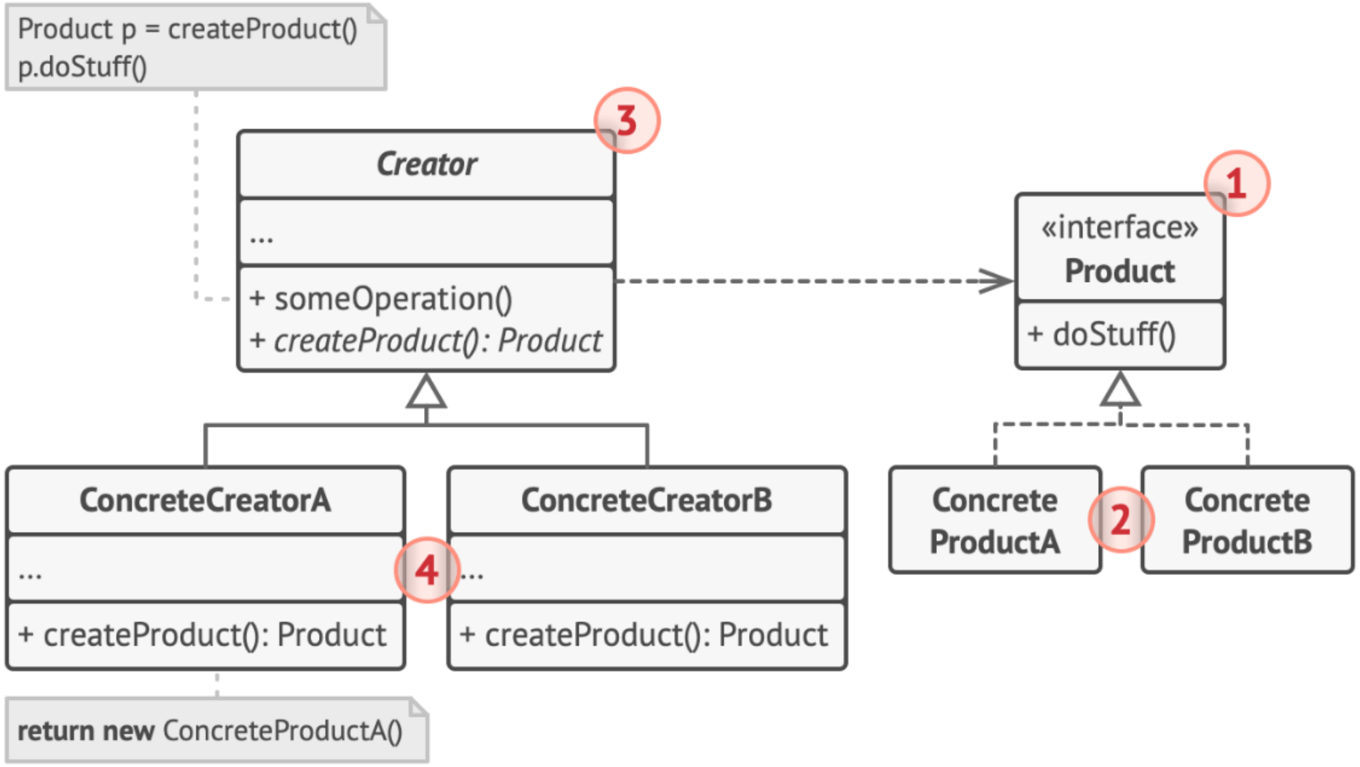
UML Class Diagram Structure

2. **Concrete Products** are different implementations of the product interface.



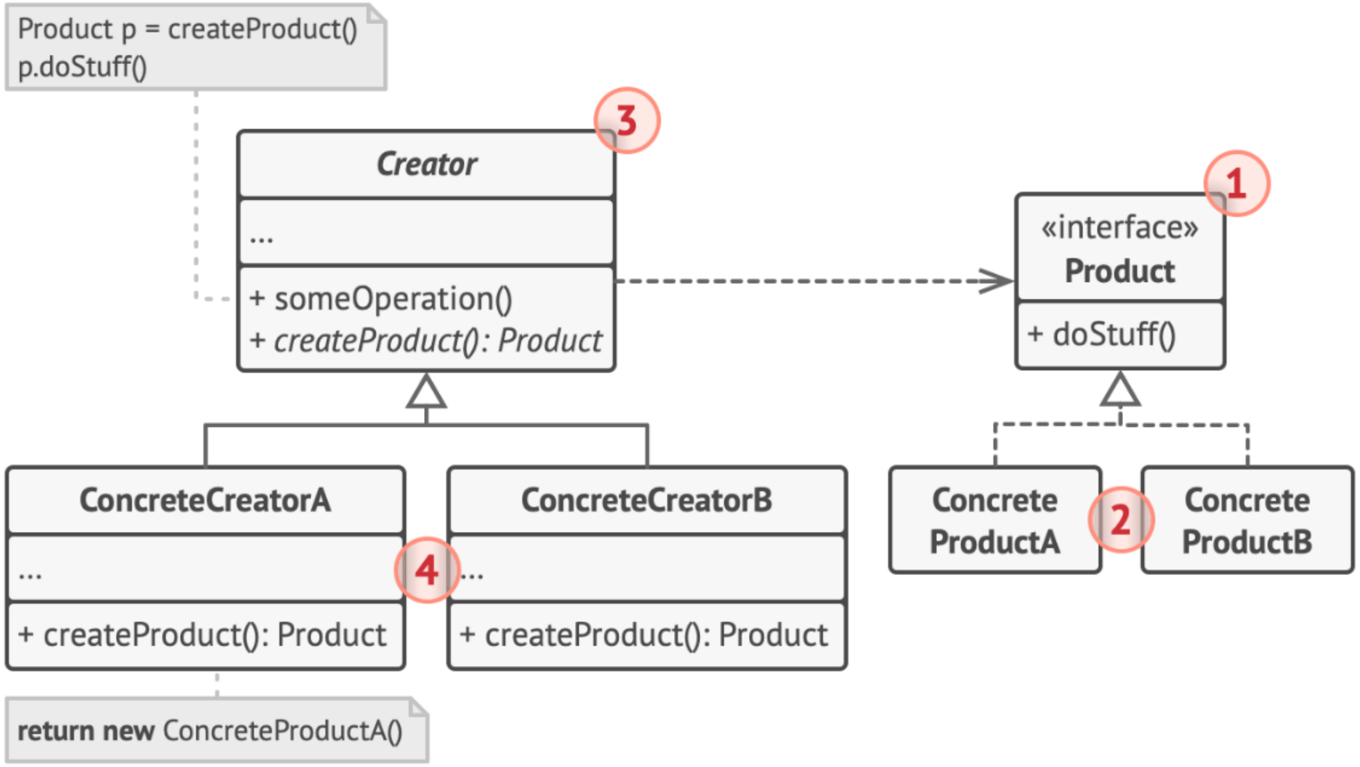
UML Class Diagram Structure

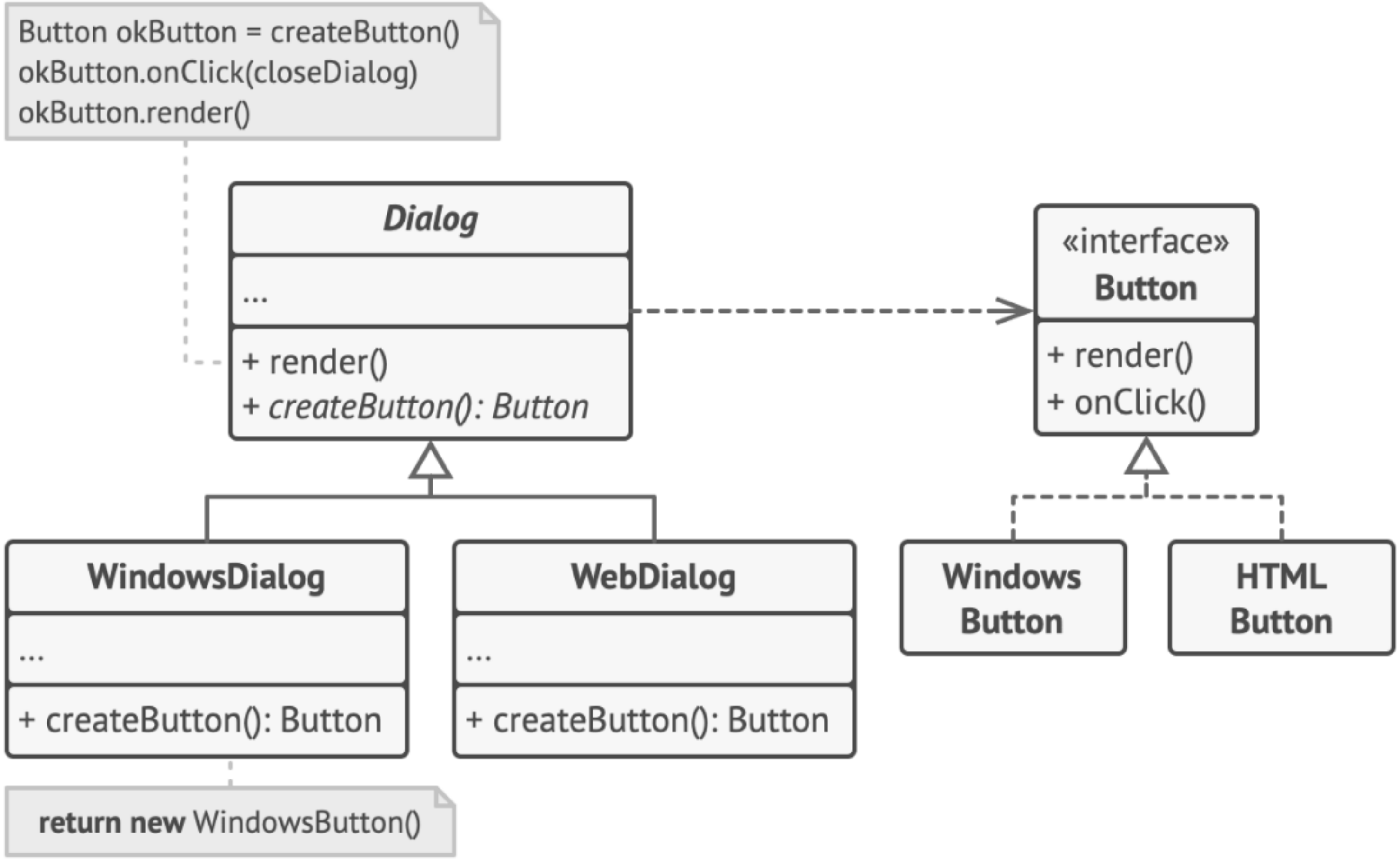
3. The **Creator** class declares the factory method that returns new product objects. It's important that the return type of this method matches the product interface.



UML Class Diagram Structure

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





The cross-platform dialog example.

Factory Method - Applicability

When you don’t know beforehand the exact types and dependencies of the objects your code should work with.

When you want to provide users of your library or framework with a way to extend its internal components.

When you want to save system resources by reusing existing objects instead of rebuilding them each time.

The big pros

Avoid tight **coupling** between the creator and the concrete products

S from SOLID

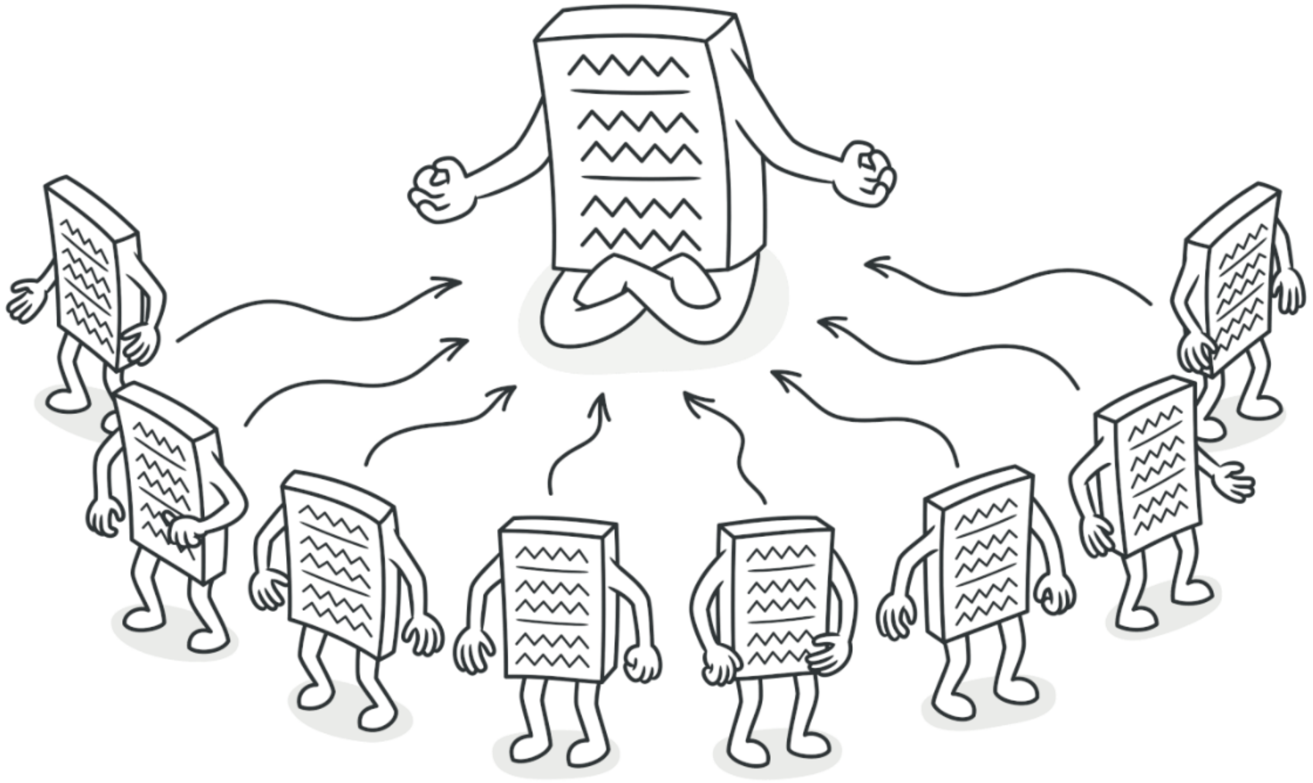
O from SOLID

Demo and Walkthrough of Factory Method

Singleton Pattern

Definition:
Let's you ensure that a class has only one instance, while providing a global access point to this instance

Singleton Pattern

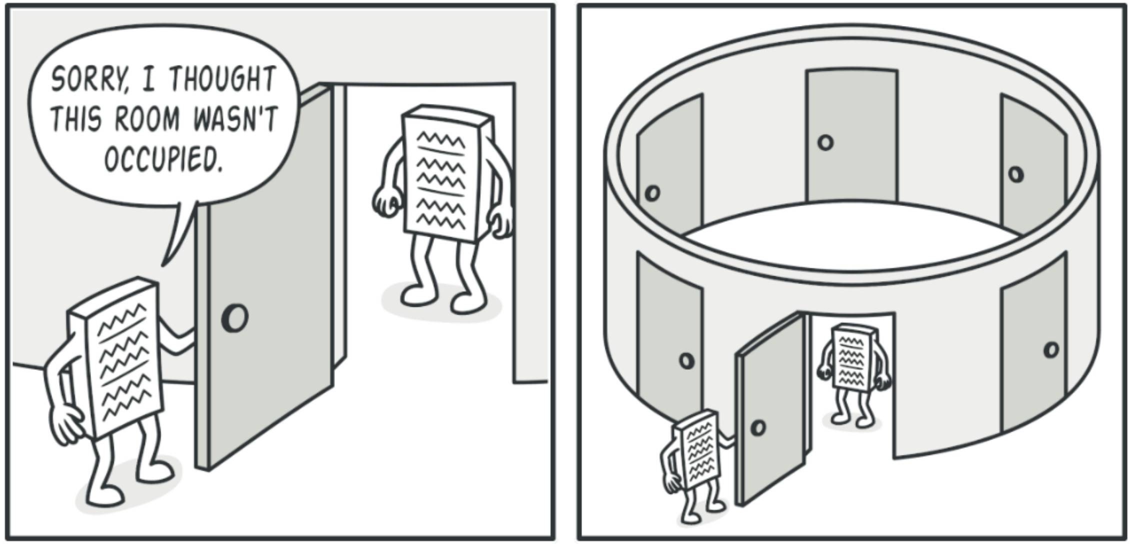


Singleton Pattern

This pattern breaks the **S** of SOLID

Does 2 things:

1. Ensures that a class has just a single instance
2. Provides a global access point to that instance

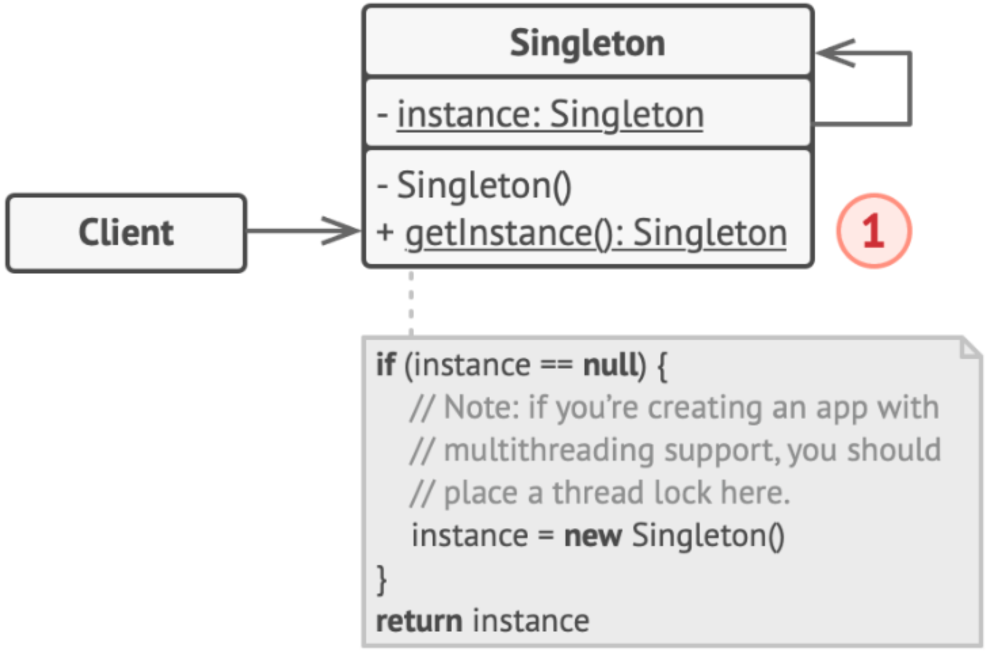


Clients may not even realize that they're working with the same object all the time.

UML Class Diagram Structure

1. The **Singleton** class declares the static method getInstance that returns the same instance of its own class.

The Singleton’s constructor should be hidden from the client code. Calling the getInstance method should be the only way of getting the Singleton object.



```
if (instance == null) {
    // Note: if you're creating an app with
    // multithreading support, you should
    // place a thread lock here.
    instance = new Singleton()
}
return instance
```

Singleton Applicability

When you need strict control over global variables

When your program should have exactly one instance available to all clients/users

The big pros

The class only has one instance

Global access point to that instance

Only initialized once – the first time

Singleton – no code example

Singleton was one of the most used extensions
for the project in this course

Data Collection Class