Creational Design Pattern

Hung Tran

Fpt software

August 20, 2021

Outline

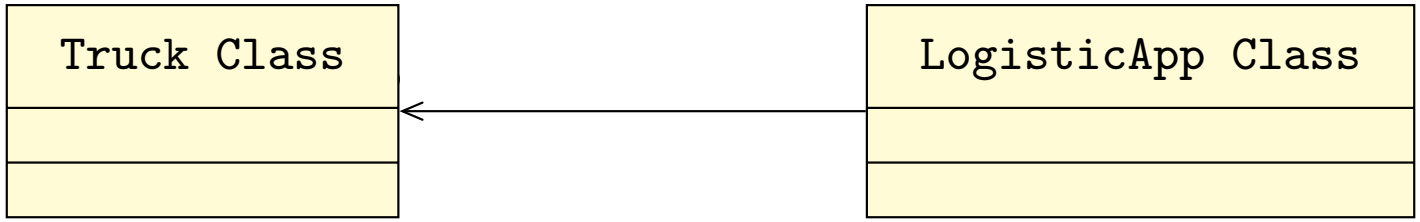
Creational Pattern Overview

Construction process of an object.

- **Singleton**: Ensure only one instance.
- **Factory Method**: Create instance without depending on its concrete type.
- **Object pool**: Reuse existing instances.
- **Abstract factory**: Create instances from a specific family.
- **Prototype**: Clone existing objects from a prototype.
- **Builder**: Construct a complex object step by step.

Logistic App

Logistic App



- Your app can only handle transportation by trucks.
- The bulk of your code lives inside the Truck class.

You have more request from oversea regions, how to incorporate sea transportation with current app?

- At present, your code is tightly coupled with Truck class.
- Adding new Ship class resulting in change all code base.
- Logistic app depends on transportation object.

Using "new" operator

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Box {
6  private:
7    double length;
8    double breadth;
9    double height;
10 };
11
12 int main(void) {
13   Box *pBox = new Box();
14   delete pBox;
15   return 0;
16 }
```

- Need name of class
- Tightly coupled with the name
- Add new class, modify the existing code
- Compiler does not know which instance created at compile time or an instance has to be created at runtime?

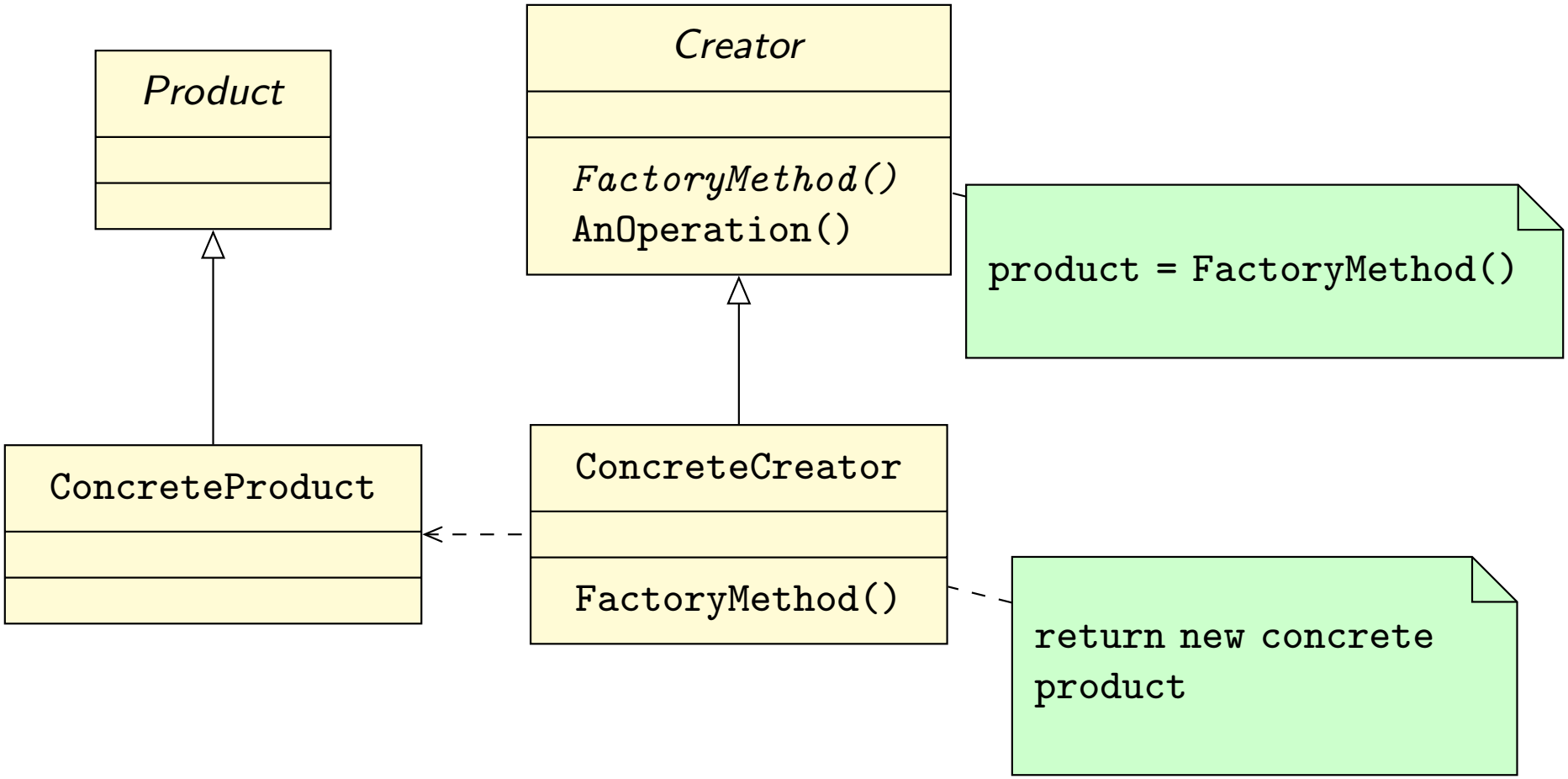
The Intent of Factory Method Design Pattern

Define an interface for creating an object, but let subclasses which class to instantitate. Factory method lets class defer instantiation to subclasses.

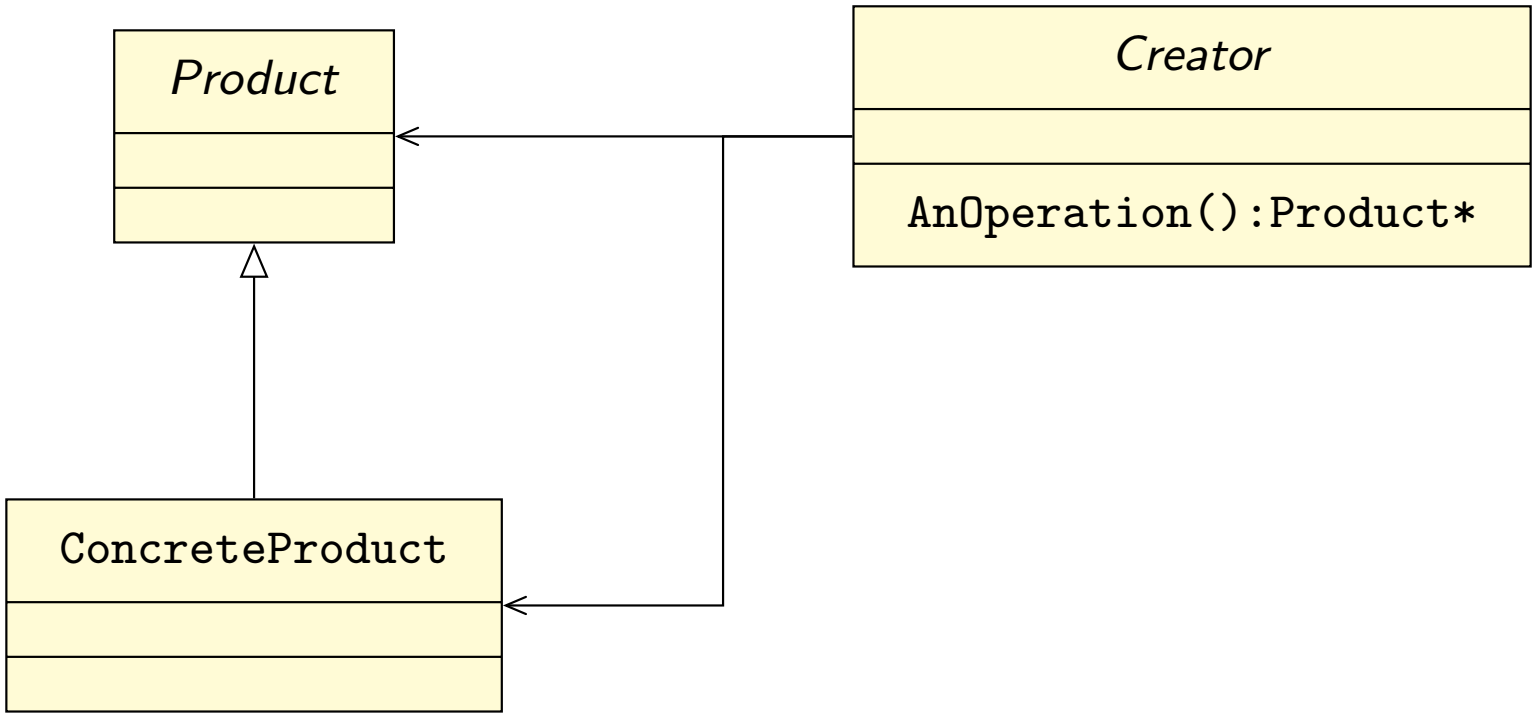
How to Implement of Factory Method Design Pattern?

- Different ways to implement
- An overridable method is provide that returns an instance of a class
- This method can be overridden to return instance of a subclass
- Behave likes **constructor**
- However, the constructor always returns the same instance
- The factory method can returns any sub-type
- The factory method also called **virtual constructor**
- C++ language does not allow virtual constructor

Structure of Factory Method Design Pattern

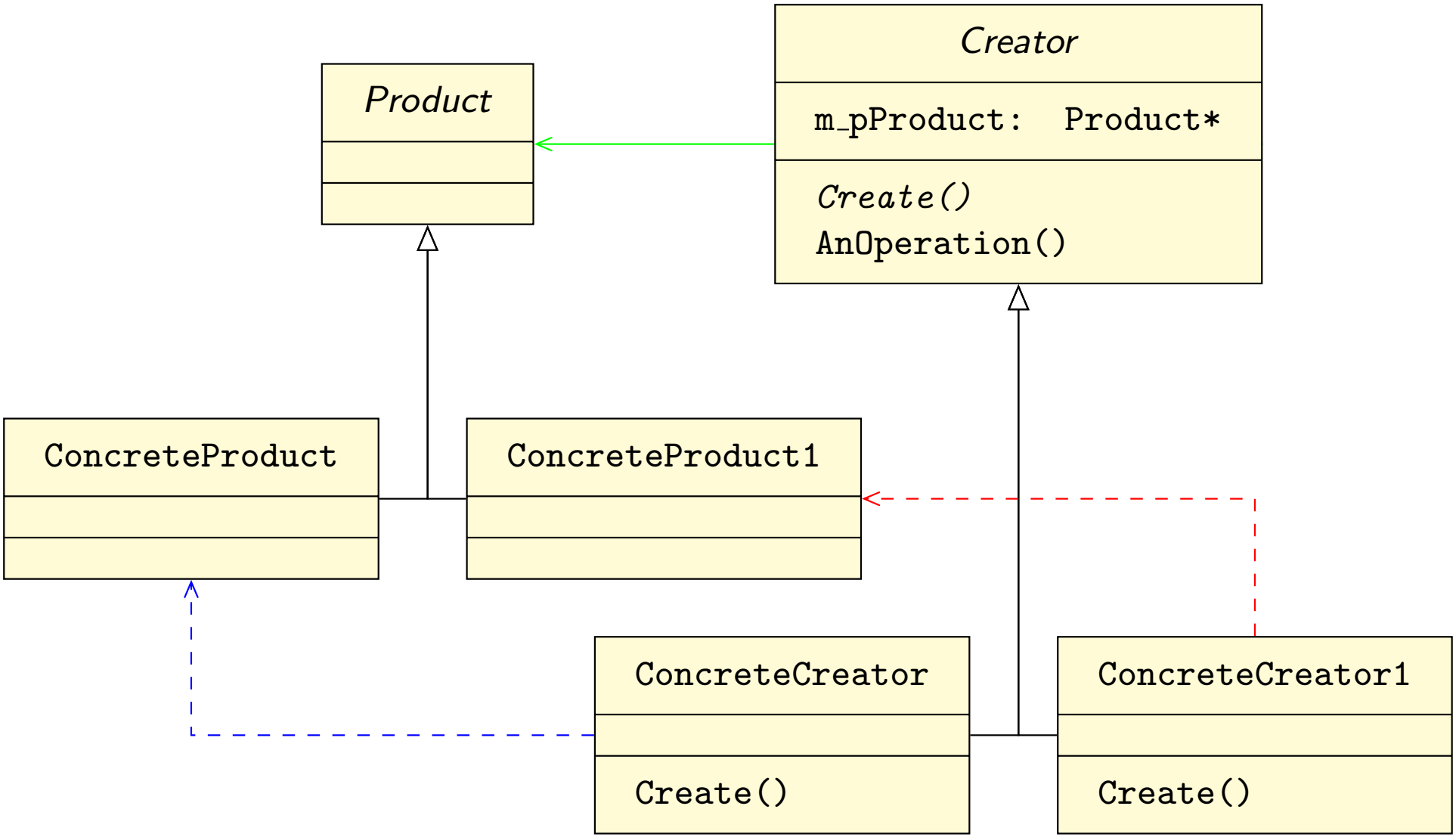


Modify existing code problem



- Operation() method returns ConcreteProduct object
- Add more ConcreteProduct class, modify AnOperation() method
- End up with if else condition.
- Always return the same class (using new operator)

Class Diagram Explaining



Create a ConcreteProduct object

main.cpp

```
#include "Creator.h"
#include "ConcreteCreator.h"
#include "ConcreteCreator1.h"
int main() {
  ConcreteCreator1 ct;
  ct.AnOperation();
}
```

- Create() Method (or factory method) is overridden in derived classes.
- The dependency on ConcreteProduct is no longer required
- AnOperation method calls Create() method.
- With the main function, which Create() method will invoke?
- No need to modify the AnOperation()
- Create() method behaves like virtual constructor

Real World Example: Application Framework?

We want to create an framework!

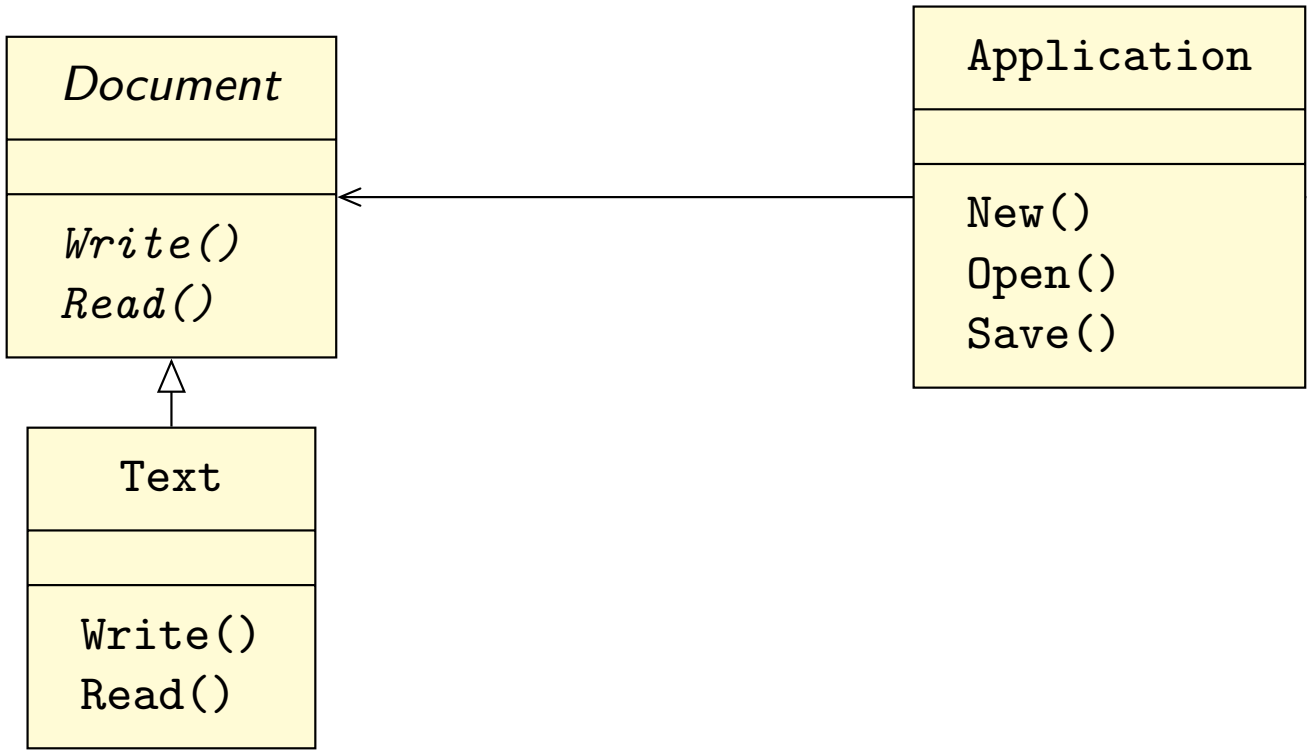
- The programmer can create an Application based on App framework.
- Framework provides managing different kinds of documents.
- Framework provides infrastructure of application.
- By using app framework, programmer can create an application which can display different kinds of data (Text or Graphics).
- To manage the data, the framework provides support through which is easy to maintain data.
- We add Application class for managing data and display them.
- We need add an Document class that Application manages

Real World Example: Application Framework?

- Application class uses the features of Document class (display them).
- However, Application does not know what kind of data is.
- Document class need to be abstract in our framework.
- **Document class has methods Read(), Write() which are overridden in derived classes.**
- **Application class uses the features of Document class.**

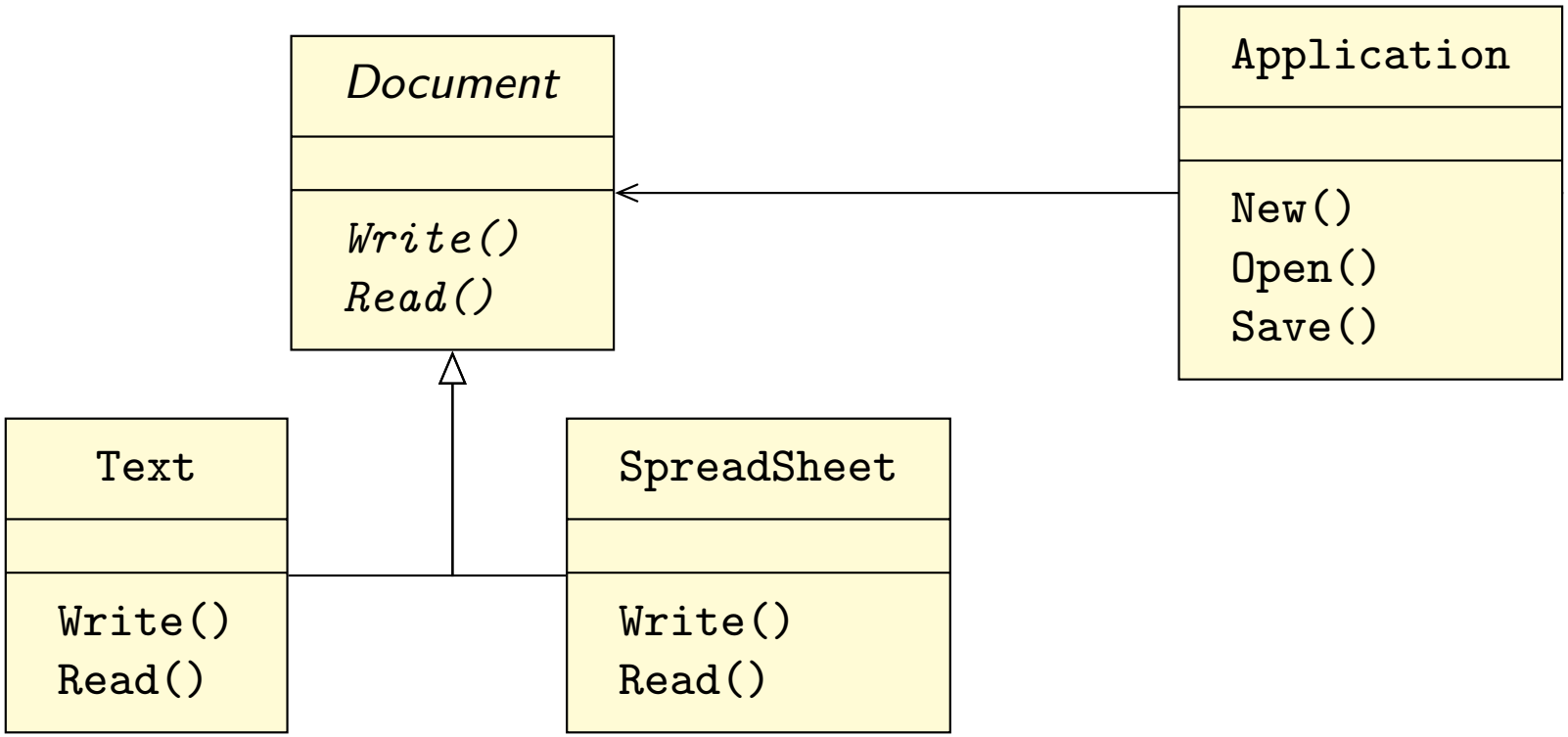
Using Framework for managing data

- Programmer want to use the framework for managing the Text data.
- Create a class Text which is derived class of Document class.
- Application can use Text through Document class.

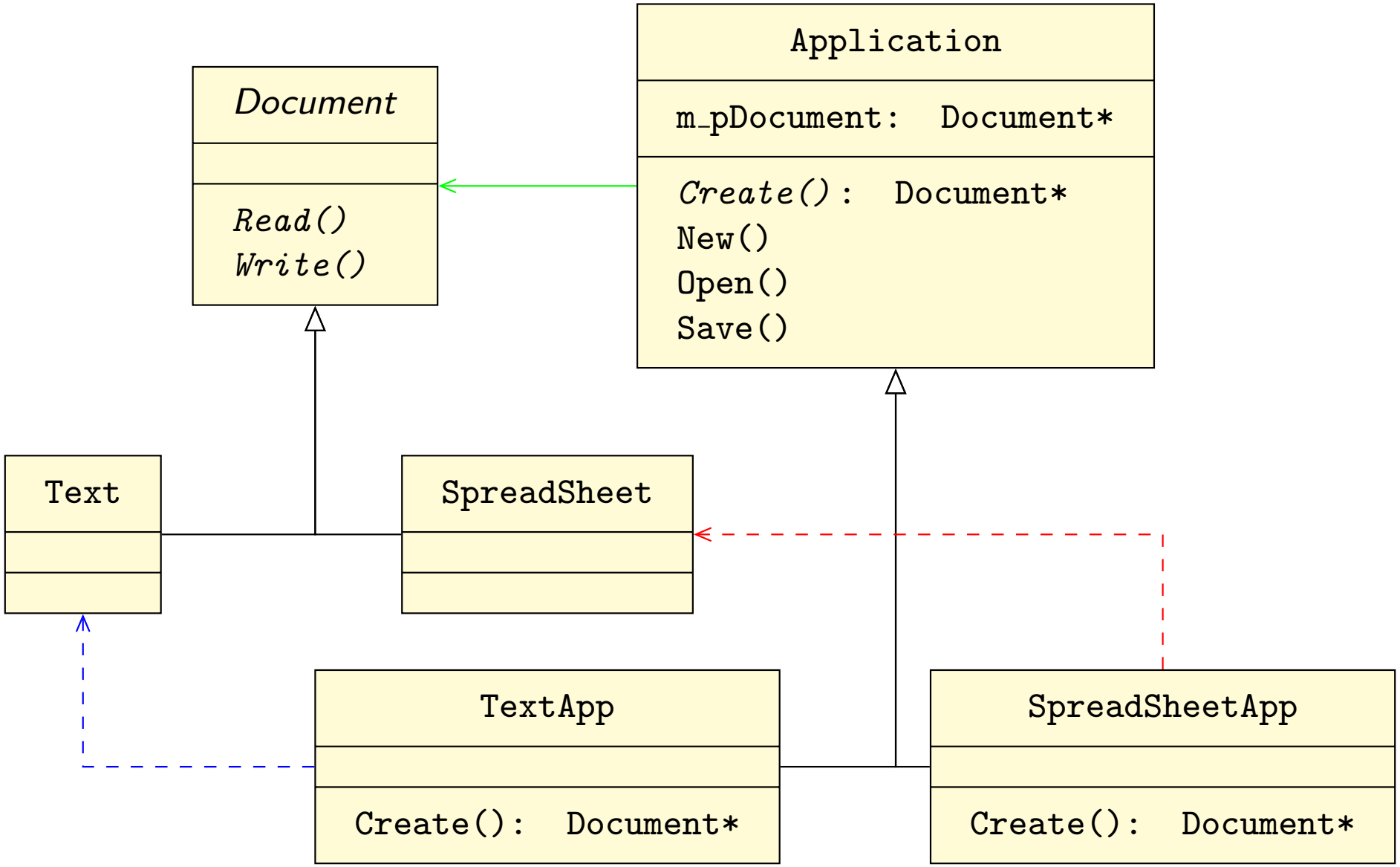


Adding SpreadSheet class

- New(), Open(), Save() are tight coupled with Text class.
- Adding new Document class, change the Application class.
- The framework does not allow modifying.



App framework with factory method



The Intent of Factory Method Design Pattern

Define an interface for creating an object, but let subclasses which class to instantitate. Factory method lets class defer instantiation to subclasses.

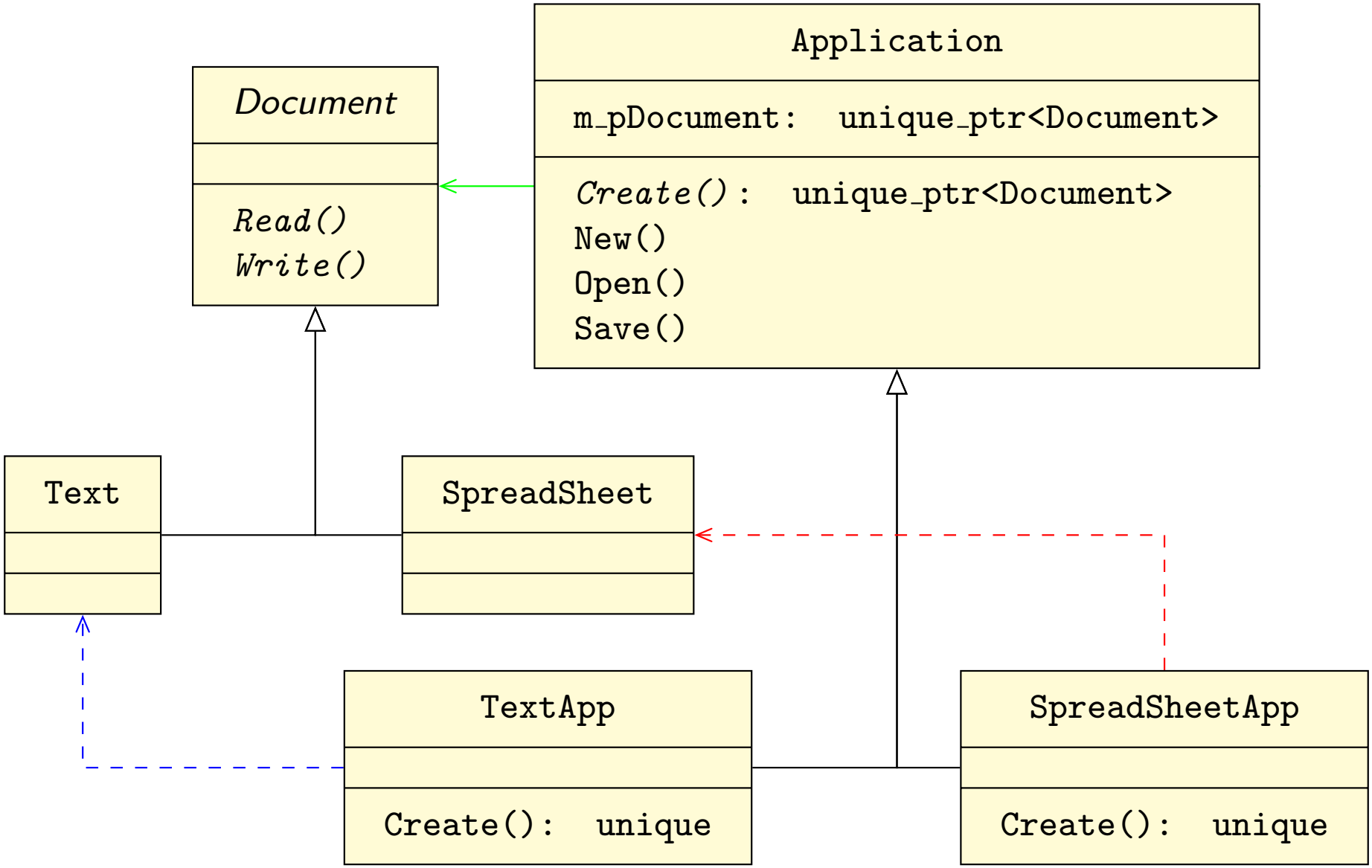
- Application class is an interface.
- Create() method is factory method.
- The subclasses (TextApp and SpreadSheetApp) instantiate objects.

The Intent of Factory Method Design Pattern

```
#include "TextApplication.h"
#include "TextDocument.h"
Document* TextApplication::Create()
        {
   return new TextDocument{};
}
```

- How to free the instance?
- Using smart pointers

Using smart pointer

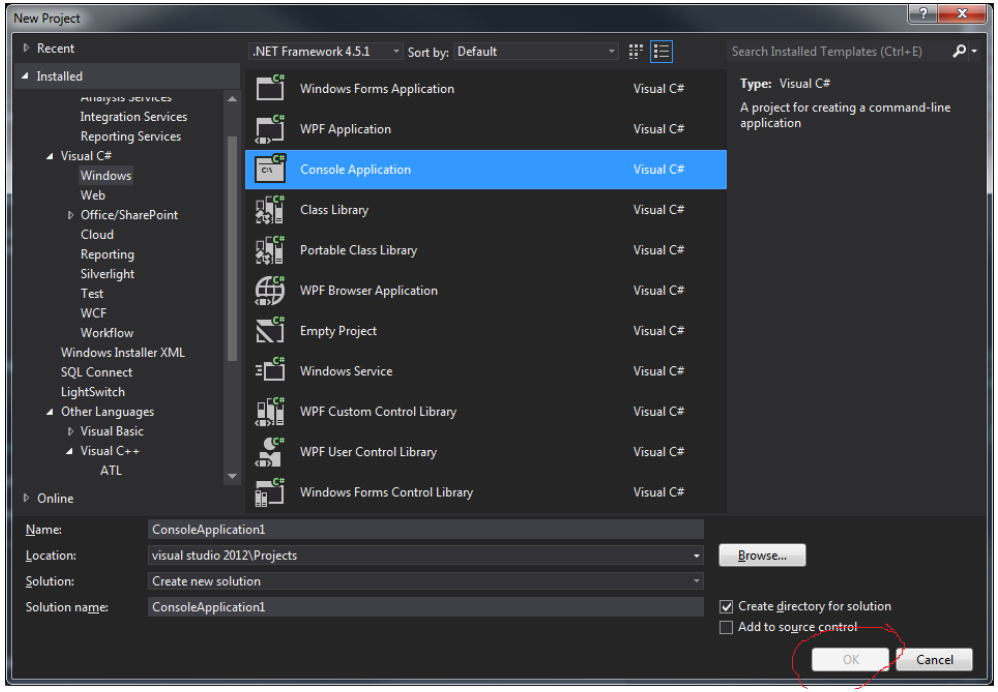


Add new Document, add more Application?

How to create multiple instances without creating corresponding application class?

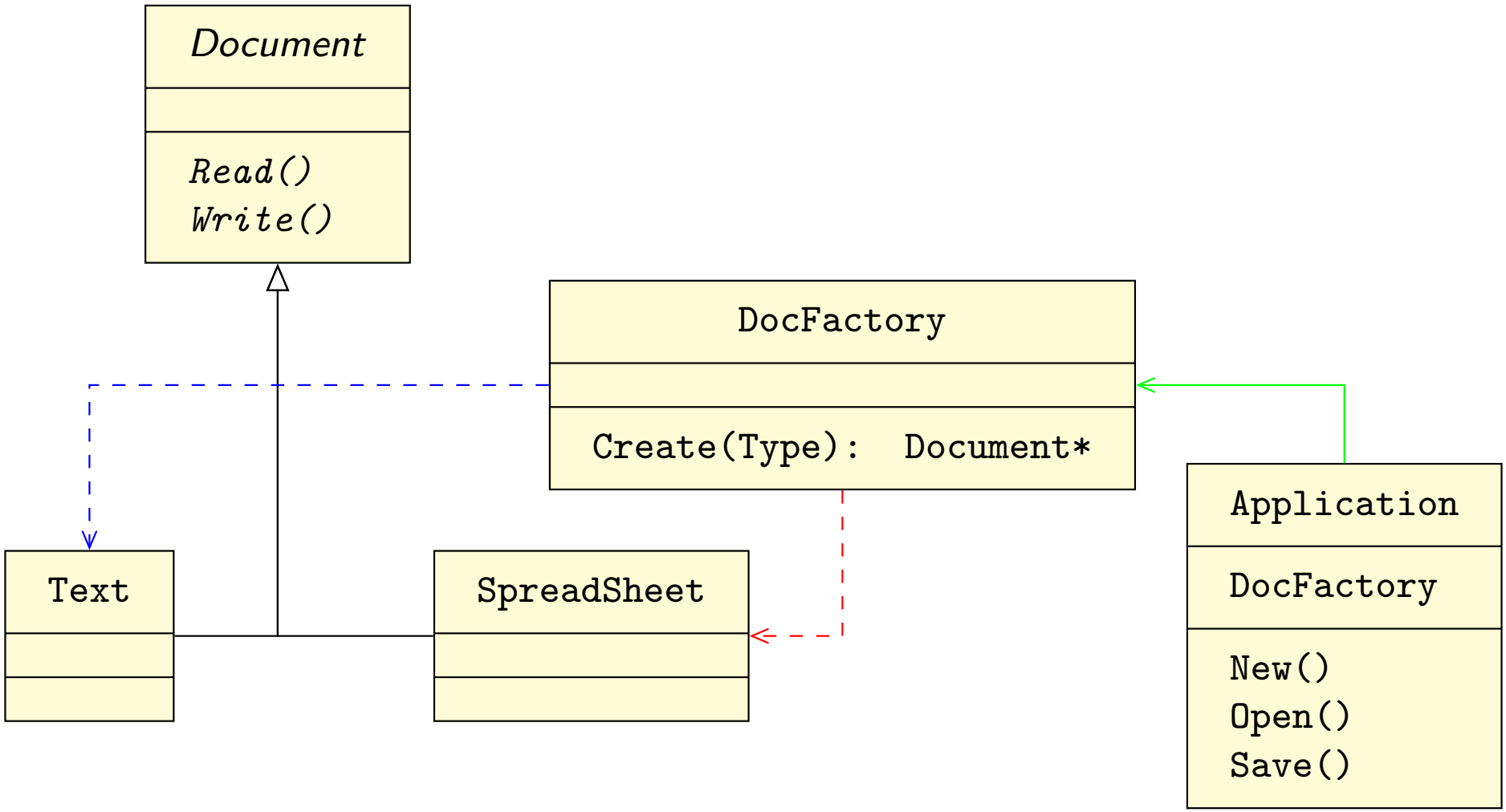
Using Parameterized Factory

What if you want to read different kinds of document?



- Like dialog shows options which clients can chose.
- You need to know how many applications before implementations.
- Input Type through user interface.

Classes structure: Parameterized Factory



Pros and Cons

Pros

- Instances can be created at runtime
- Promote loose coupling
- Construction becomes simple due to abstraction
- Construction becomes encapsulated
- May not return new instance every time (return a cache instance), useful for object pool

Cons

- Every new product class may require a corresponding factory class.

Where to use?

- A class does not know which instance it needs at runtime.
- A class does not want to depend on concrete classes that it uses.
- You want to encapsulate the creation process.