CSE 776-Design Patterns Factory Method

Mrunal Dharmendra Maniar
Yuxuan Xie

INTRODUCTION

- Factory method is a creational design pattern
- INTENT :

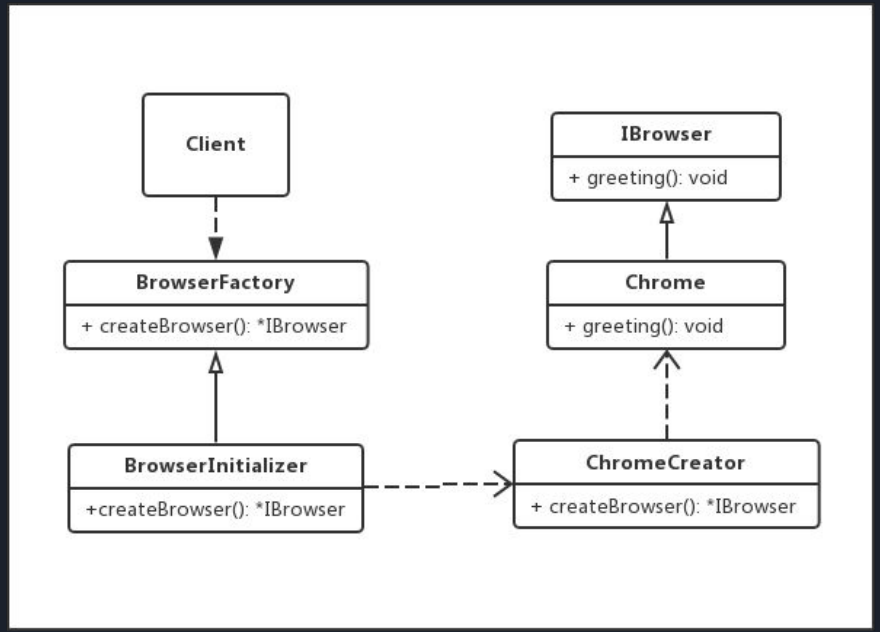
  > “Define an interface for creating an object, but let subclasses decide which class to instantiate”

- Also known as virtual constructor

MOTIVATION

- FRAMEWORKS :
  - Most of the frameworks use abstract classes to define various objects and maintain relationships between them. They are also responsible for application specific object creation
  - Unless specified, the type of object to be created is not known to the library. In such cases, factory method can be used.

MOTIVATION EXAMPLE



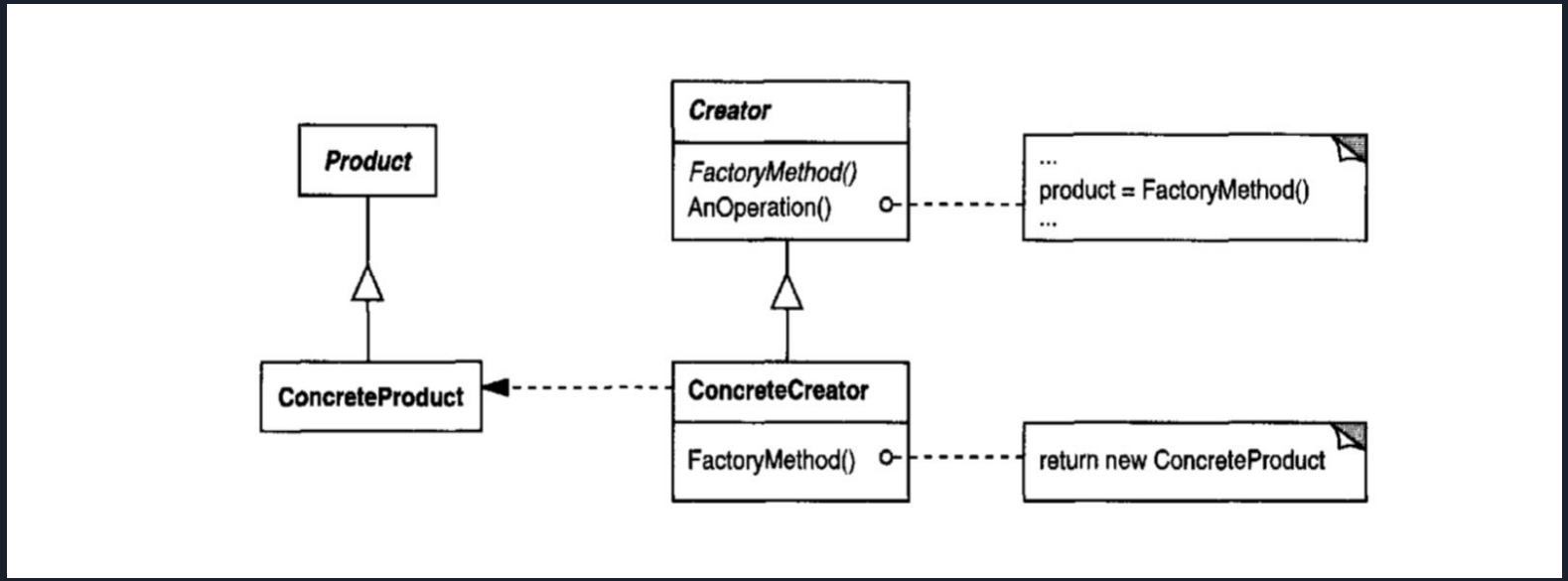
FORCES

- The need to use reusable classes with the flexibility to extend them
- The provision to operate on objects without knowing what type of object creation in advance

APPLICABILITY

- Needed when required type of object instantiation is unknown at compile time.
- When the class needs its subclass to specify the object it creates.
- Developer wants to localize the knowledge of helper subclasses

BASIC STRUCTURE



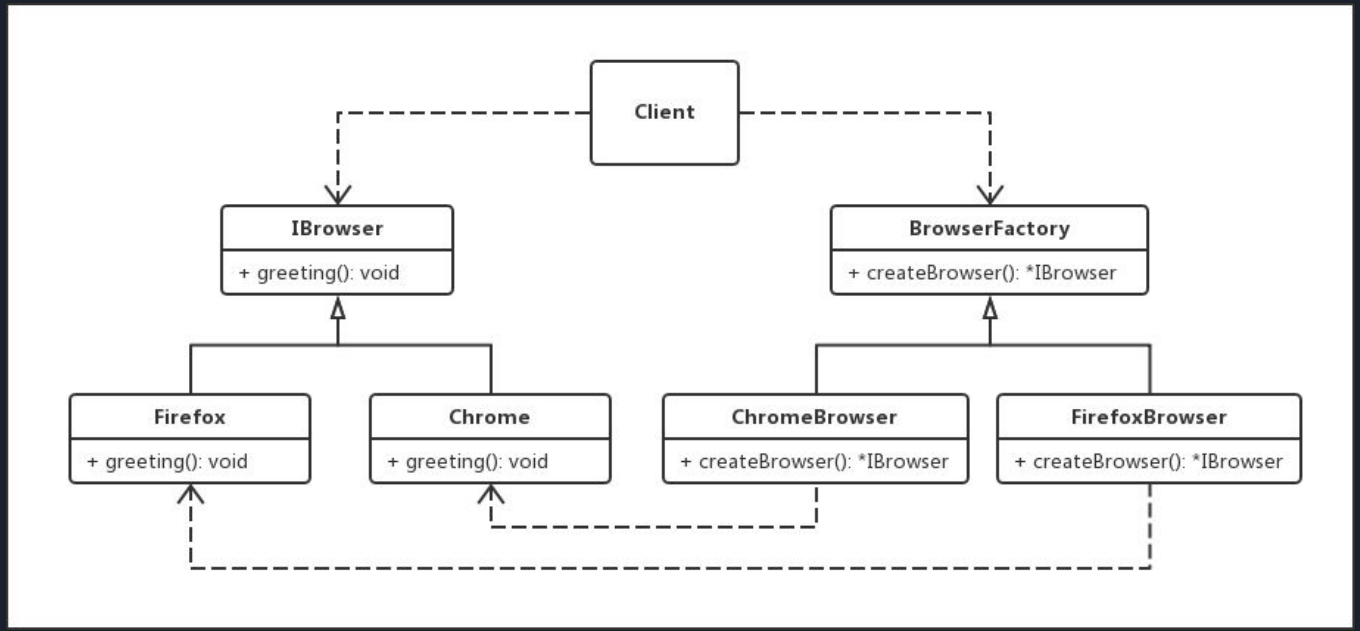
PARTICIPANTS

- Product(IBrowser)
  - Defines an interface which is implemented by classes whose objects are returned by the factory method
- Concrete Product(Firefox, Chrome)
  - Classes that implement the the above interfaces
- Creator(BrowserFactory)
  - Factory method is declared here. There may or may not be a default implementation
- Concrete Creator (ChromeBrowser)
  - Overrides the factory method and returns an object of concrete product

COLLABORATORS

- Creator depends on its subclasses for creation of objects of Concrete Product
- Creator may or may not perform series of operations on the object created and simply returns a reference to Product

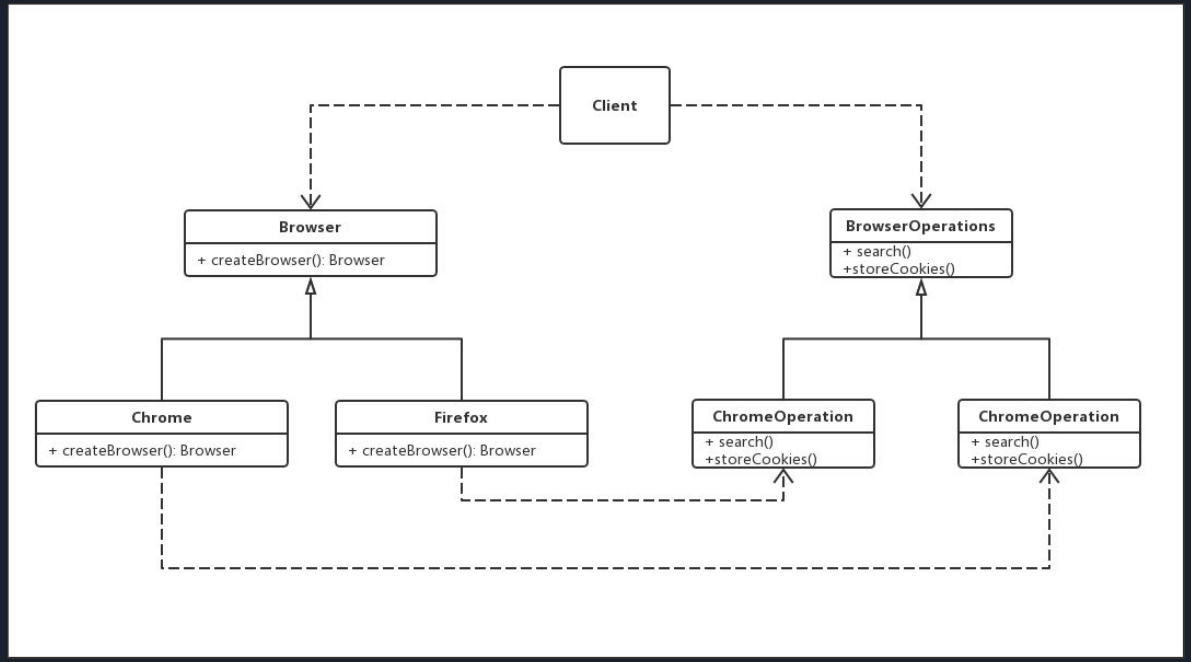
RESULTING ARCHITECTURE



CONSEQUENCES

- Details of concrete subclasses are decoupled from the client
- New concrete subclass can be added easily
- It might lead to creation of many subclasses if the object of Product needs an additional object.

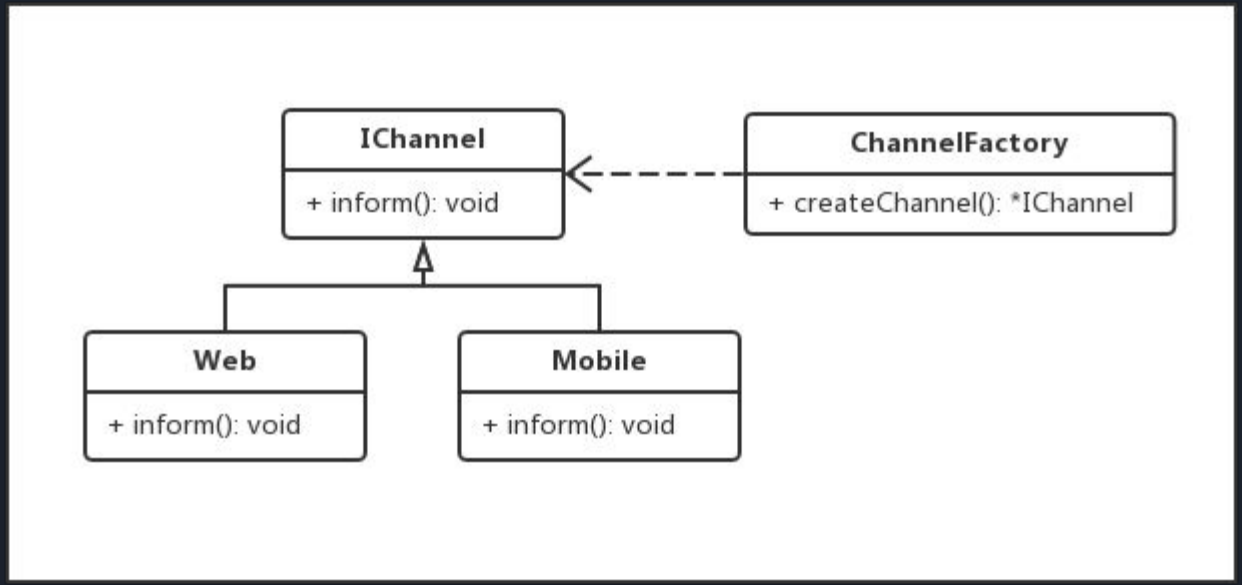
CONSEQUENCES (continued)



IMPLEMENTATION

- Abstract creators with no default implementation
- Concrete creator class with default implementation
- Parameterized methods
- Templates

PARAMETERIZED FACTORY



PARAMETERIZED FACTORY - (continued)

```
class ChannelFactory
{
public:
    IChannel* createChannel(const char* device);
};

IChannel* ChannelFactory::createChannel(const char* device)
{
    IChannel* channel = nullptr;
    if (device == "Mobile")
    {
        channel = new Mobile();
    }
    else if (device == "Web")
    {
        channel = new Web();
    }
    return channel;
}

//In the client
IChannel* mobile_channel = ChannelFactory().createChannel("Mobile");
IChannel* web_channel = ChannelFactory().createChannel("Web");
```

TEMPLATIZED FACTORY

```
1  class ChannelFactory {
2  public:
3      ChannelFactory() { }
4      virtual IChannel* CreateChannel() = 0;
5  };
6
7  template <class T>
8  class Channel : public ChannelFactory {
9  public:
10     IChannel* CreateChannel() {
11         return new T;
12     }
13 };
14 // In the Client
15 Channel<Mobile> myMobile;
```

KNOWN USES

- Android
- Several places in Java API
- TestNG
- .NET framework class library

DISADVANTAGES

- Codebase tends to become huge because of so many subclasses.
- Reading and understanding the code becomes difficult because of the high level of abstraction

REFERENCES

- https://en.wikipedia.org/wiki/Factory_method_pattern
- Design Patterns, Elements of Reusable Object-Oriented Software, Erich Gamma, et. al., Addison-Wesley, 1994, ISBN 0-201-63361-2
- https://sourcemaking.com/design_patterns/factory_method
- Instructor presentation of Prof. Fawcett

THANK YOU