FACTORY METHOD

GROUP 2 - APCS15 – CS202

1. Problem:

We need a framework needs to standardize the architecture for many applications, but also allow for each applications to define their own objects and provide for their instantiation.

2. Solution: Using factory method

“The Factory Method Pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.”- Gang of four.

The Factory Method encapsulates the instantiations of concrete types. It lets subclasses decide which class to instantiate, not because the pattern allows subclasses themselves to decide at runtime, but since a creator class is written without knowledge of the actual products which will be created, which is decided all by the choice of the subclass that is used.

3.Definition

In class-based programming, the factory method pattern is a creational 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. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

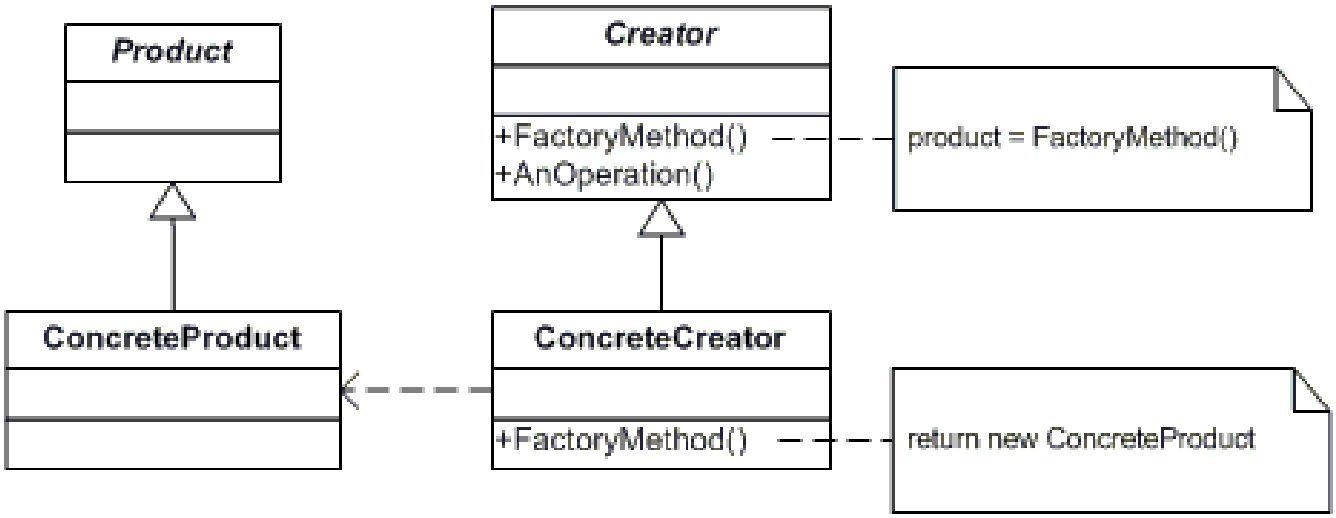
4. What is a factory in programming?

In object-oriented **programming** (OOP), a **factory** is an object for creating other objects – formally a **factory** is a function or method that returns objects of a varying prototype or class from some method call, which is assumed to be "new".

Participants

The classes and objects participating in this pattern are:

- **Product (Phone)**
  - defines the interface of objects the factory method creates
- **ConcreteProduct (SamsungPhone, HTCPhone, NokiaPhone)**
  - implements the Product interface
- **Creator (Client)**
  - declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
  - may call the factory method to create a Product object.
- **ConcreteCreator** overrides the generating method for creating ConcreteProduct objects



STEP TO DO

1. *Create an abstract interface class (ConcreateProduct class)*

```
#include <iostream>
using namespace std;

enum BrandName
{
    Samsung;
    HTC;
    Nokia;
};

class Phone
{
public:
    virtual void printInformation()=0;
    static Phone* create( BrandName name )
    {
        switch(name)
        {
            case Samsung: return new SamsungPhone();
            case HTC: return new HTCPhone();
            case Nokia: return new NokiaPhone();
            default: return NULL;
        }
    }
};
```

2. *Create implementation classes which dervied from interface class (Product class)*

```
class SamsungPhone : public Phone
{
    void printInformation()
    {
        cout << "This is a Samsung phone" << endl;
    }
};

class HTCPhone : public Phone
{
    void printInformation()
    {
        cout << "This is a HTC phone" << endl;
    }
};

class NokiaPhone : public Phone
{
    void printInformation()
    {
        cout << "This is a Nokia phone" << endl;
    }
};
```

3. *Create a client class (Creator class)*

```
class Client
{
private:
    Phone* MyPhone;
public:
    Client( int name )
    {
        switch(name)
        case 1: MyPhone = Phone::create(Samsung);
                    break;
        case 2: MyPhone = Phone::create(HTC);
                  break;
        case 3: MyPhone = Phone::create(Nokia);
                     break;
        default: MyPhone = NULL;
    }
    ~Client()
    {
        if (MyPhone)
        {
            delete []MyPhone;
            MyPhone = NULL;
        }
    }
    Phone* getPhone()
    {
        return MyPhone;
    }
};
```

4. *Main function*

```
int main()
{
    // 1 : Samsung
    // 2 : HTC
    // 3 : Nokia
    Client* pClient = new Client(2);
    Phone* pPhone = Client->getPhone();
    pPhone->printInformation();
    return 0;
}
```

5.Advantages

- Simplest Design

- Hide concrete classes from client

- The code deals with the Interface

- Advocate for the component based development

- Factory methods eliminate the need to bind application-specific classes into your code.

- The code only deals with the product interfaces; therefore, it can work with any user-defined concrete product classes.

- Factory methods provide hooks for sub-classes to create different concrete products. However, we can change this behavior in the derived class to create an edit control which accepts only floating point values.

- Factory methods connect parallel class hierarchies in such a way that it localizes the knowledge of which classes belong together.

- Defines the connection between the two class hierarchies.

6.Disadvantages

- Since the pattern relies on using a private constructor, the class cannot be extended

- Re-factoring an existing class to use factories breaks existing clients

- This provides a hook so that we can derive a sub-class to create different controls to display the data.

- Also, the Factory method connects the class hierarchies with minimum coupling.

----------THE END----------