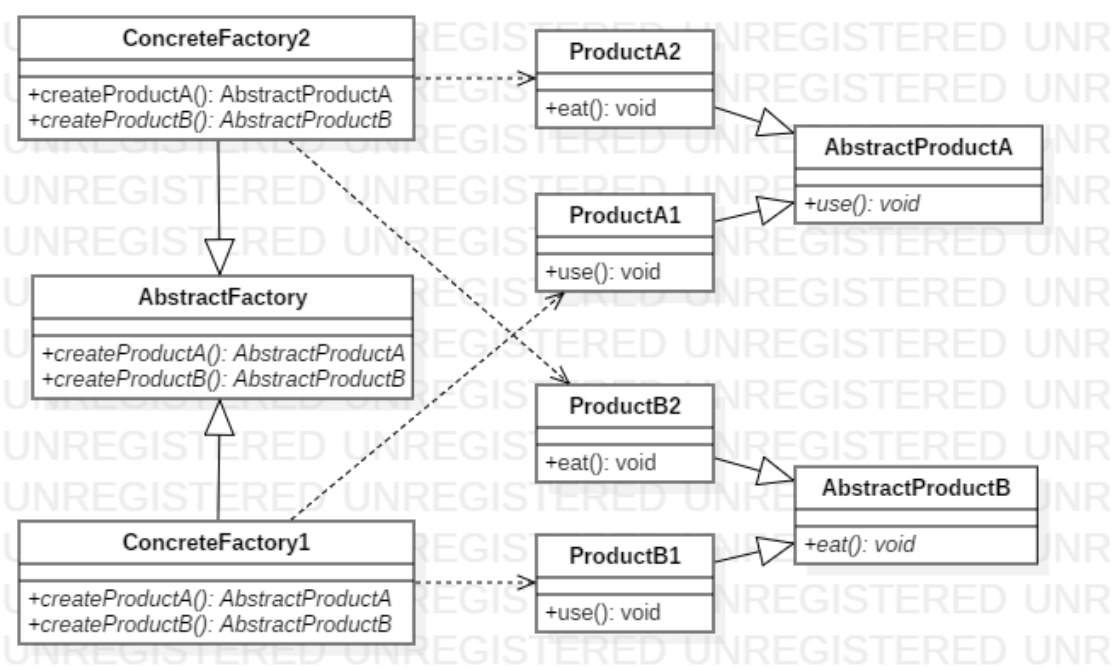


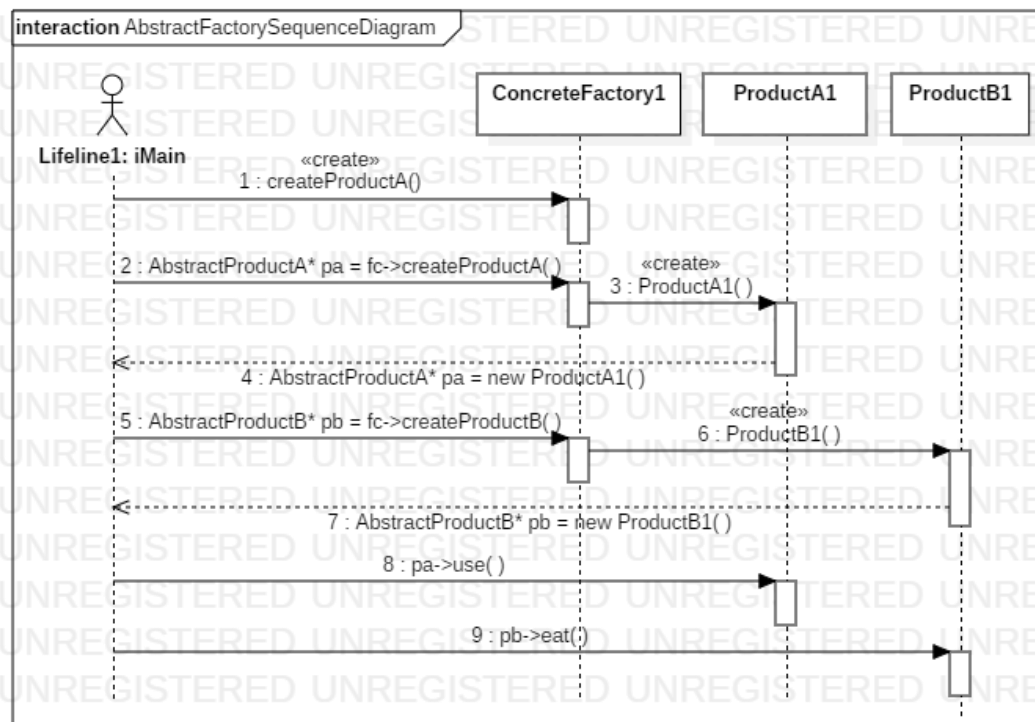
# 模式定义

**抽象工厂模式：** 提供一个创建一系列相关或相依赖对象的接口，而无须指定它们具体的类。

# 模式结构



# 时序图



## 关键代码

```
int main()
{
    AbstractFactory * fc = new ConcreteFactory1();
    AbstractProductA* pa = fc->createProductA();
    AbstractProductB* pb = fc->createProductB();
    pa->use();
    pb->eat();
}

AbstractProductA * ConcreteFactory1::createProductA()
{
    return new ProductA1();
}

AbstractProductB * ConcreteFactory1::createProductB()
{
    return new ProductB1();
}

void ProductA1::use()
{
    cout << "use Product A1" << endl;
}
```

```

void ProductB1::eat()
{
    cout << "eat Product B1" << endl;
}

```

## 测试结果

The screenshot shows a C++ IDE with several files open: main.cpp, ProductA2.cpp, ConcreteFactory2.cpp, ProductB1.cpp, ProductB2.cpp, and ProductC1.cpp. The main.cpp file is selected, showing the following code:

```

7
8 using namespace std;
9 int main()
10 {
11     AbstractFactory * fc = new ConcreteFactory1();
12     AbstractProductA* pa = fc->createProductA();
13     AbstractProductB* pb = fc->createProductB();
14     pa->use();
15     pb->eat();
16
17     AbstractFactory* fc2 = new ConcreteFactory2();
18     AbstractProductA* pa2 = fc2->createProductA();
19     AbstractProductB* pb2 = fc2->createProductB();
20     pa2->use();
21     pb2->eat();
22
23     delete fc;
24     delete pa;
25     delete pb;
26     delete fc2;
27     delete pa2;
28     delete pb2;
29 }

```

On the right side, there is a terminal window titled "D:\Design Pattern1" showing the output of the program:

```

use Product A1
eat Product B1
use Product A2
eat Product B2
请按任意键继续.

```