软件体系结构 作业10

22920212204392 黄勖

1 改写本例，用于添加另一个具体工厂和具体产品。

我们建立一个抽象工厂，它定义了两个虚方法：创建货物和注册货物列表

```java
package framework;

import java.util.Vector;

public abstract class Factory {
    public final Product create(String owner){
        Product p = createProduct(owner);
        registerProduct(p);
        return p;
    }
    protected abstract Product createProduct(String owner);
    protected abstract void registerProduct(Product p);
    public abstract Vector getOwners();
}
```

我们再建立一个抽象产品，它可以被使用

```java
package framework;

public abstract class Product {
    public abstract void use();
}
```

接下来我们构建一个IDCard

```java
package idcard;
import framework.*;

public class IDCard extends Product{
    private String owner;
    IDCard(String owner){
        this.owner = owner;
        System.out.println("Create " + owner + "\'s card.");
    }

    public void use(){
        System.out.println("Use " + owner + "\'s card.");
    }
    public String getOwner() {
        return owner;
    }
}
```

然后构建IDCardFactory

```java
package idcard;

import framework.*;
import java.util.Vector;

public class IDCardFactory extends Factory {
    private Vector owners = new Vector();
    protected Product createProduct(String owner){
        return new IDCard(owner);
    }
    protected void registerProduct(Product product){
        owners.add(((IDCard)product).getOwner());
    }
    public Vector getOwners() {
        return owners;
    }
}
```

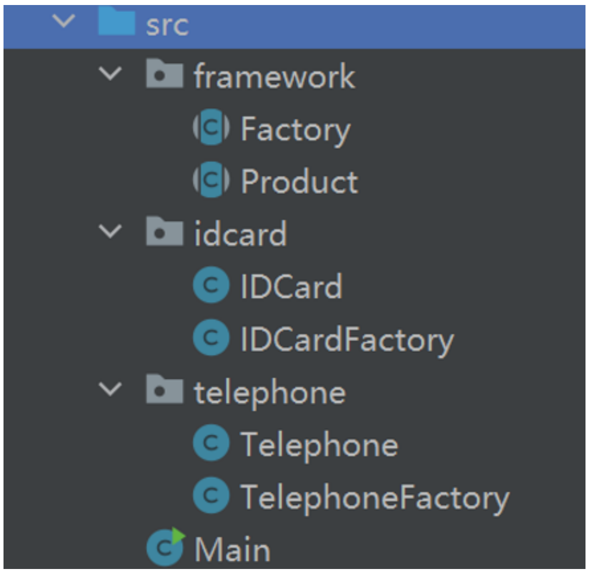
接下来我们构建一个Telephone

```java
package telephone;

import framework.*;

public class Telephone extends Product {

    private String owner;
    Telephone(String owner){
        this.owner = owner;
        System.out.println("Create " + owner + "\'s telephone.");
    }

    public void use(){
        System.out.println("Use " + owner + "\'s telephone.");
    }

    public String getOwner() {
        return owner;
    }
}
```

然后构建TelephoneFactory

```
package telephone;

import framework.*;
import java.util.Vector;

public class TelephoneFactory extends Factory{
    private Vector owners = new Vector();
    protected Product createProduct(String owner){
        return new Telephone(owner);
    }
    protected void registerProduct(Product product){
        owners.add(((Telephone)product).getOwner());
    }
    public Vector getOwners() {
        return owners;
    }
}
```

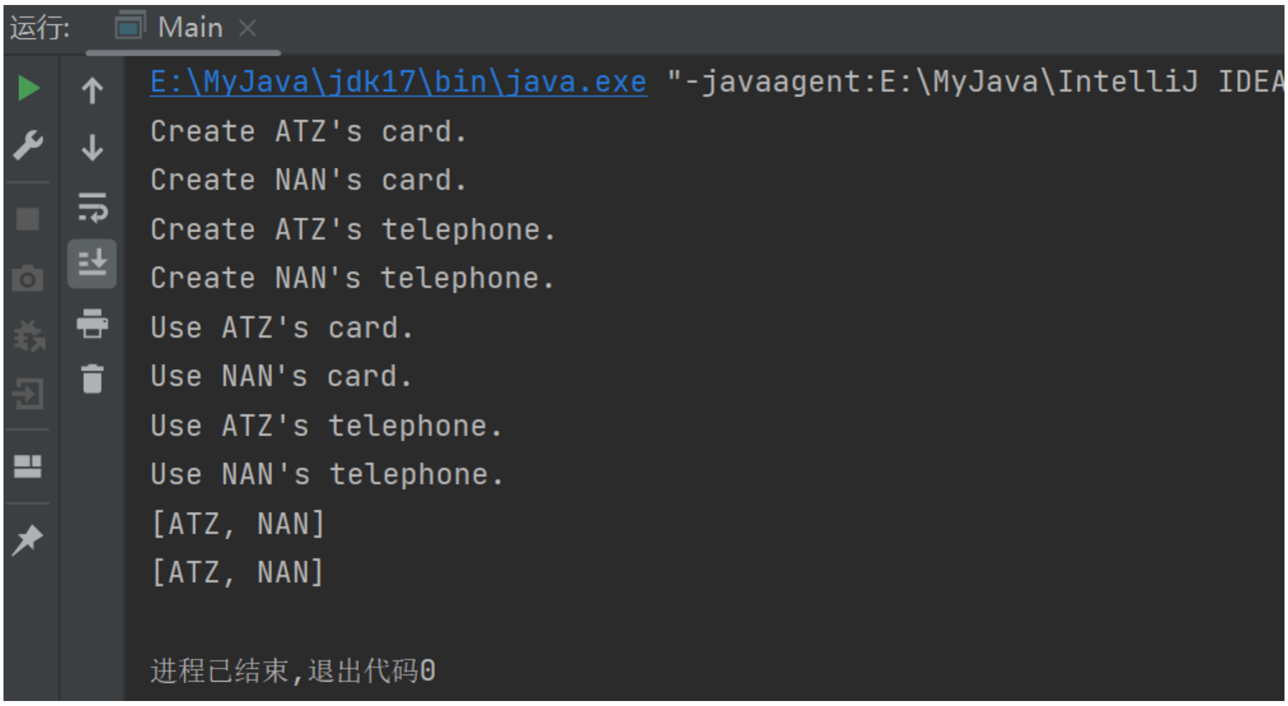
项目结果如下：



```
src
    framework
        Factory
        Product
    idcard
        IDCard
        IDCardFactory
    telephone
        Telephone
        TelephoneFactory
    Main
```

Main函数：

```
import framework.*;
import idcard.*;
import telephone.*;

public class Main {
    public static void main(String[] args) {
        Factory idCardFactory = new IDCardFactory();
        Factory telephoneFactory = new TelephoneFactory();

        Product p1 = idCardFactory.create("ATZ");
        Product p2 = idCardFactory.create("NAN");

        Product p3 = telephoneFactory.create("ATZ");
        Product p4 = telephoneFactory.create("NAN");

        p1.use();p2.use();p3.use();p4.use();
        System.out.println(idCardFactory.getOwners());
        System.out.println(telephoneFactory.getOwners());

    }
}
```



```
E:\MyJava\jdk17\bin\java.exe "-javaagent:E:\MyJava\IntelliJ IDEA
Create ATZ's card.
Create NAN's card.
Create ATZ's telephone.
Create NAN's telephone.
Use ATZ's card.
Use NAN's card.
Use ATZ's telephone.
Use NAN's telephone.
[ATZ, NAN]
[ATZ, NAN]

进程已结束,退出代码0
```

2 请举例说明其他的工厂模式的应用。

1. Java集合框架中的java.util.Collection接口的**iterator()**工厂方法使得具体的java集合类可以通过实现该方法返回一个Iterator迭代器对象
2. Java消息服务JMS的实现广泛使用工厂方法模式
3. JDBC中也大量使用工厂方法模式，例如在创建连接对象Connection、语句对象Statement和结果集对象ResultSet时都使用工厂方法
4. C++的rb_tree使用了工厂模式的抽象工厂模式。提供了创建一系列相关容器的接口，而无需指定显式指定容器。