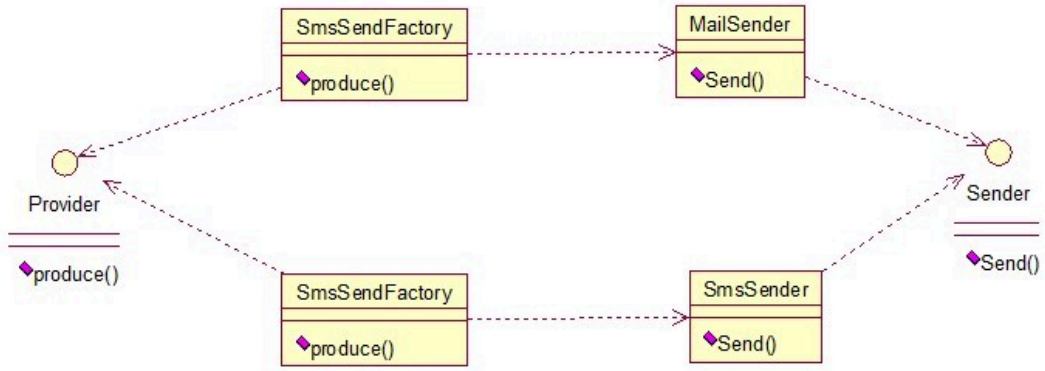
工厂方法模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，如何解决？就用到抽象工厂模式，创建多个工厂类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。因为抽象工厂不太好理解，我们先看看图，然后就和代码，就比较容易理解。




代码实现：

```
public interface Sender {
 void send();
}
```

两个实现类

```
public class MailSender implements Sender{
 @Override
 public void send() {
  System.out.println(" mail send ");
 }
}

public class SmsSender implements Sender{
 @Override
 public void send() {
  System.out.println("sms send");
 }
}
```

再提供一个接口

```
public interface Provider {
 public Sender produce();
}
```

两个工厂类

```
public class MailSenderFactory implements Provider {
 @Override
 public Sender produce() {
  return new MailSender();
 }

}

public class SmsSenderFactory implements Provider {
 @Override
 public Sender produce() {
  return new SmsSender();
 }

}
```

测试

```
public class Test {

 public static void main(String [] args){
  Provider  provider = new SmsSenderFactory();
  Sender send = provider.produce();
  send.send();

  Provider pro = new MailSenderFactory();
  pro.produce().send();
 }
}
```

这个模式的好处是，如果你现在想增加一个新功能：发及时信息，则只需做一个实现类，实现Sender接口，同时做一个工厂类，实现Provider接口，无需去改动原有的代码。这样做的目的是拓展性较好！

反射实现抽象工厂：

```
package com.chang.reflect;

interface Sender {
    void send();
}
class MailSender implements Sender {
    @Override
    public void send() {
        System.out.println("mail send");
    }
}
```

```
class SmsSender implements Sender {
    @Override
    public void send() {
        System.out.println("sms send");
    }
}
class ReflectFactory {
    static Sender getSender(String sender) throws Exception {
        return (Sender)Class.forName(sender).newInstance();
    }
}

public class TestReflectFactory {
    public static void main(String[] args) throws Exception{
        Sender sender =
ReflectFactory.getSender("com.chang.reflect.MailSender");
        sender.send();
    }
}
```