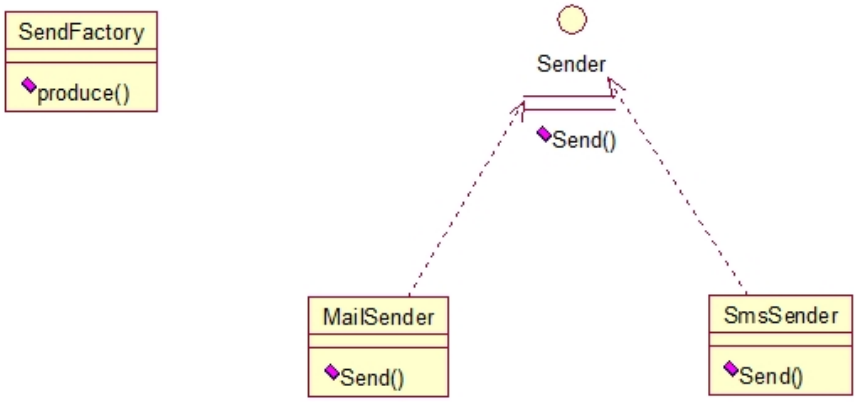
工厂方法模式分为三种：

11、普通工厂模式，就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。
首先看下关系图：



举例如下：（我们举一个发送邮件和短信的例子）

首先，创建二者的共同接口：

[java] view plaincopy

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

其次，创建实现类：

[java] view plaincopy

```
public class MailSender implements Sender {
    @Override
    public void Send() {
        System.out.println("this is mailsender!");
    }
}
```

[java] view plaincopy

```java
public class SmsSender implements Sender {

    @Override
    public void Send() {
        System.out.println("this is sms sender!");
    }
}
```

最后，建工厂类：

[java] view plaincopy

```java
public class SendFactory {

    public Sender produce(String type) {
        if ("mail".equals(type)) {
            return new MailSender();
        } else if ("sms".equals(type)) {
            return new SmsSender();
        } else {
            System.out.println("请输入正确的类型!");
            return null;
        }
    }
}
```

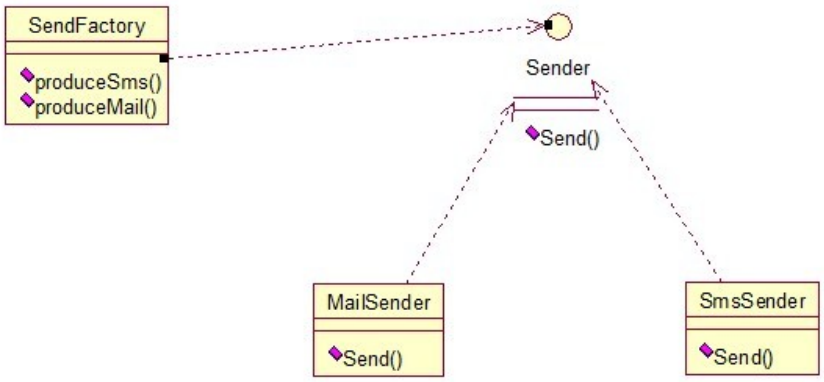
我们来测试下：

```java
public class FactoryTest {

    public static void main(String[] args) {
        SendFactory factory = new SendFactory();
        Sender sender = factory.produce("sms");
        sender.Send();
    }
}
```

输出：this is sms sender!

22、多个工厂方法模式，是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。关系图：



将上面的代码做下修改，改动下SendFactory类就行，如下：

```
[java] view plaincopypublic class SendFactory {

    public Sender produceMail(){
            return new MailSender();
        }

        public Sender produceSms(){
            return new SmsSender();
        }
    }
```

测试类如下：

```
[java] view plaincopy

    public class FactoryTest {

        public static void main(String[] args) {
            SendFactory factory = new SendFactory();
```

```
            Sender sender = factory.produceMail();
            sender.Send();
        }
    }
```

输出：this is mailsender!

33、静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

[java] view plaincopy

```
    public class SendFactory {

        public static Sender produceMail(){
            return new MailSender();
        }

        public static Sender produceSms(){
            return new SmsSender();
        }
    }
```

[java] view plaincopy

```
    public class FactoryTest {

        public static void main(String[] args) {
            Sender sender = SendFactory.produceMail();
            sender.Send();
        }
    }
```

输出：this is mailsender!

总体来说，工厂模式适合：凡是出现了大量的产品需要创建，并且具有共同的接口时，可以通过工厂方法模式进行创建。在以上的三种模式中，第一种如果传入的字符串有误，不能正确创建对象，第三种相对于第二种，不需要实例化工厂类，所以，大多数情况下，我们会选用第三种——静态工厂方法模式。