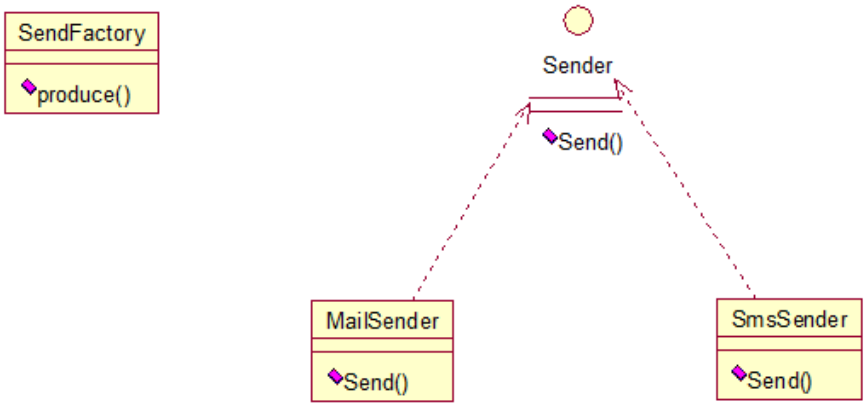
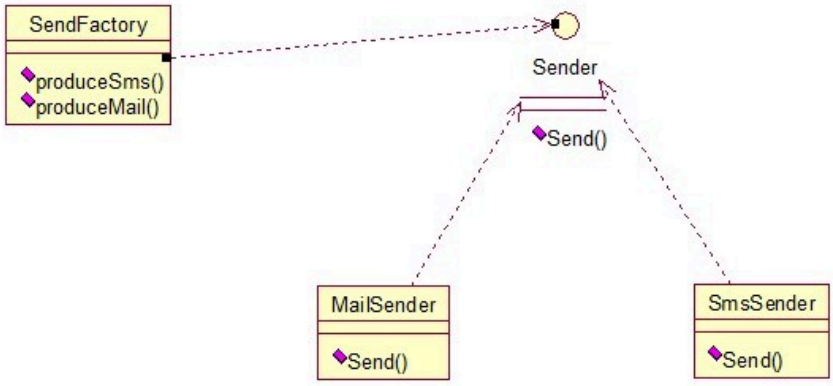
普通工厂模式：就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。



多个工厂方法模式：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。



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。

代码如下：

公共接口

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

邮件实现类

```
public class MailSender implements Sender{
 @Override
 public void send() {
  System.out.println("邮件发送短信");
 }
}
```

短信实现类

```
public class SmsSender implements Sender{
 @Override
 public void send() {
  System.out.println("调用发送短信的方法");
 }
}
```

构建工厂类

```
//1. 普通的工厂模式，依赖于传参
public class SendFactory {
 public Sender produce(String type){
  if("sms".equals(type)){
   return new SmsSender();
  }else if("mail".equals(type)){
   return new MailSender();
  }else{
   System.out.println("请输入正确的类型");
   return null;
  }
 }

 //2. 多个工厂模式，不依赖于传参
 public Sender produceSms(){
  return new SmsSender();
 }
 public Sender produceMail(){
  return new MailSender();
 }


 //3. 静态工厂模式
 public static Sender produceSms2 (){
  return new SmsSender();
 }

 public static Sender produceMail2(){
  return new MailSender();
 }
}
```

测试类

```
public class SenderFactoryTest {

 public static void main(String[] args){

  SendFactory sendFactory = new SendFactory();

  //Sender send = sendFactory.produce("mail");
  //send.send();

  Sender send = sendFactory.produceMail();
  send.send();

  //调用静态方法
  Sender send2 = SendFactory.produceMail2();
  send2.send();
 }
}
```

关于工厂模式总结：当我们需要大量创建某一类产品时，并且具有共同的接口，可以通过工厂模式进行创建。