实验报告

实验设计名称： 外观模式

专 业： 软 件 技术

班 级： 2018级

姓 名： 林志龙

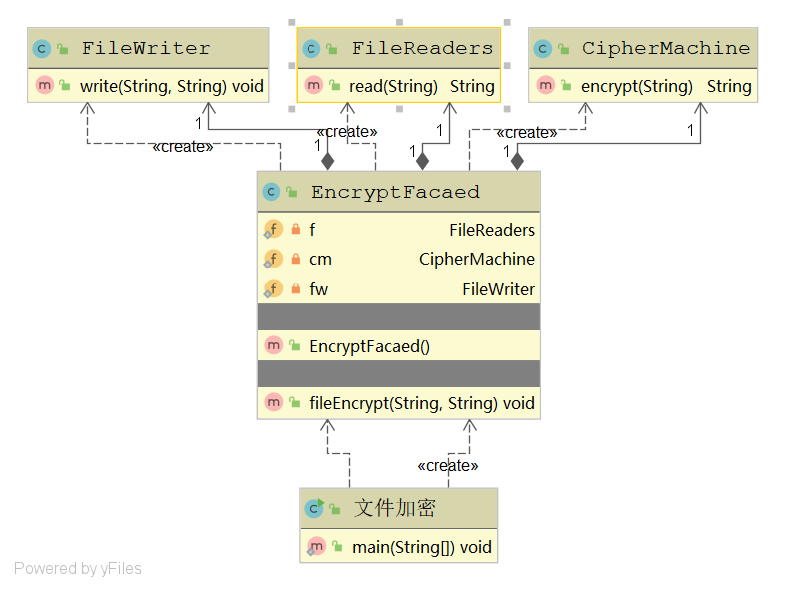
学 号： 1817700302

2020 年 4 月 17 日

1. 实验内容

某系统需要提供一个文件加密模块，加密流程包括三个操作，分别是读取源文件、加密、保存加密之后的文件。读取文件和保存文件使用流来实现，这三个操作相对独立，其业务代码封装在三个不同的类中。现在需要提供一个统一的加密外观类，用户可以直接使用该加密外观类完成文件的读取、加密和保存三个操作，而不需要与每一个类进行交互，使用外观模式设计该加密模块。

1. 实验设计类图(UML)



1. 实验代码

外观模式角色有：① (Facade) 外观角色；② (SubSystem)子系统角色。

1. 外观角色实现代码：

**package** 外观模式3;**public class** EncryptFacaed {  
  
 **private static** FileReaders *f*;  
 **private static** CipherMachine *cm*;  
 **private static** FileWriter *fw*;  
 **public** EncryptFacaed(){

*f* = **new** FileReaders();  
 *cm* = **new** CipherMachine();  
 *fw* = **new** FileWriter();  
 }  
 **public void** fileEncrypt(String fileNameSrc, String fileNameDes){  
 String p = *f*.read(fileNameSrc);  
 String e = *cm*.encrypt(p);  
 *fw*.write(e,fileNameDes);  
 }  
}

1. 子系统角色实现代码：

**package** 外观模式3;  
**import** java.io.FileInputStream;  
**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
*//子系统角色 文件读取类***public class** FileReaders {  
  
 **public** String read(String fileReaders){  
 System.***out***.println(**"读取文件，获取明文文件路径："**+fileReaders);  
 StringBuffer sb = **new** StringBuffer();  
  
 **try** {  
 *//使用FileInputStream的read()方法读取文本文件* FileInputStream inputStream = **new** FileInputStream(fileReaders);  
 **int** data;  
 **while**((data=inputStream.read())!=-1){  
 sb =sb.append((**char**) data);  
 }  
 inputStream.close();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 *//打印错误消息* System.***out***.println(e.getMessage());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 System.***out***.println(e.getMessage());  
 }  
 **return** sb.toString();  
 }  
}

**package** 外观模式3;  
**import** java.io.FileOutputStream;  
*//子系统角色 文件保存类***public class** FileWriter {  
 **public void** write(String encryText,String fileNameDes){  
 System.***out***.println(**"保存密文，写入文件。密文路径为："** +fileNameDes + **",内容为："** +encryText);  
 **try** {  
 FileOutputStream outputStream = **new** FileOutputStream(fileNameDes);  
 outputStream.write(encryText.getBytes());  
 outputStream.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 System.***out***.println(e.getMessage());  
 }  
 }  
}

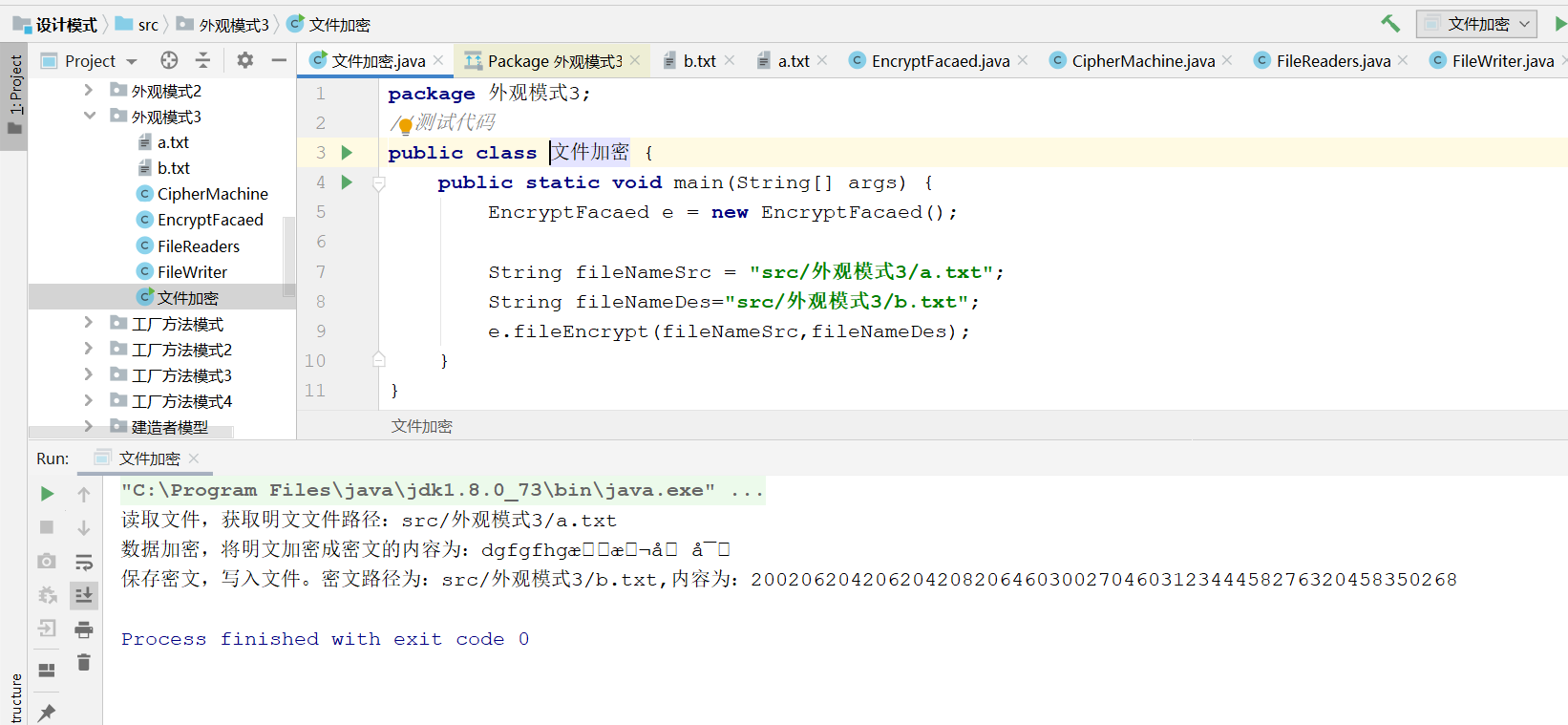
**package** 外观模式3;  
*//子系统角色 数据加密类***public class** CipherMachine {  
 **public** String encrypt(String plaintext){  
 System.***out***.println(**"数据加密，将明文加密成密文的内容为："** +plaintext);  
 String s =**""**;  
 **for**(**int** i=0;i<plaintext.length();i++){  
 *// \*2加密 运用字符串中的charAt方法获取获取下标为i的字符串并乘以2* String c = String.*valueOf*(plaintext.charAt(i)\*2);  
 s +=c;  
 }  
 **return** s;  
 }  
}

1. 客户端测试代码：

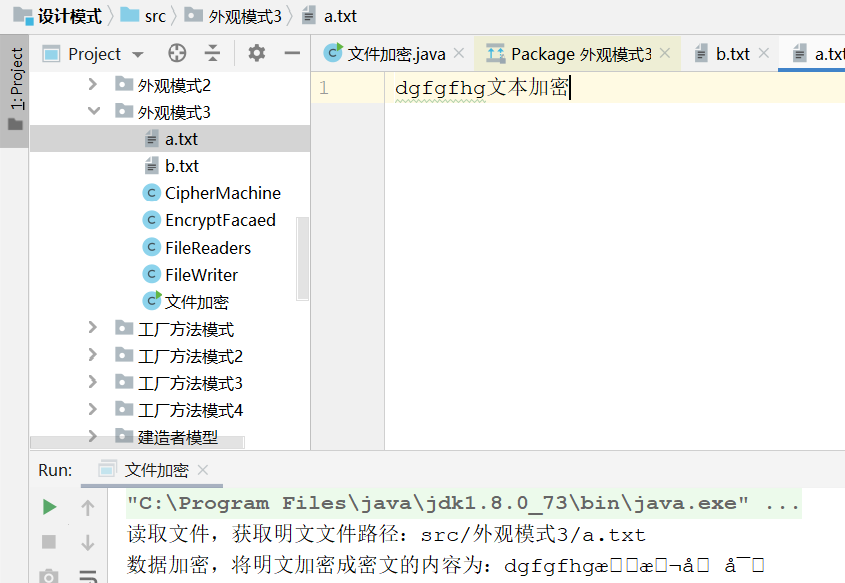
**package** 外观模式3;  
*//测试代码***public class** 文件加密 {  
 **public static void** main(String[] args) {  
 EncryptFacaed e = **new** EncryptFacaed();  
  
 String fileNameSrc = **"src/外观模式3/a.txt"**;  
 String fileNameDes=**"src/外观模式3/b.txt"**;  
 e.fileEncrypt(fileNameSrc,fileNameDes);  
 }  
}

1. 实验结果

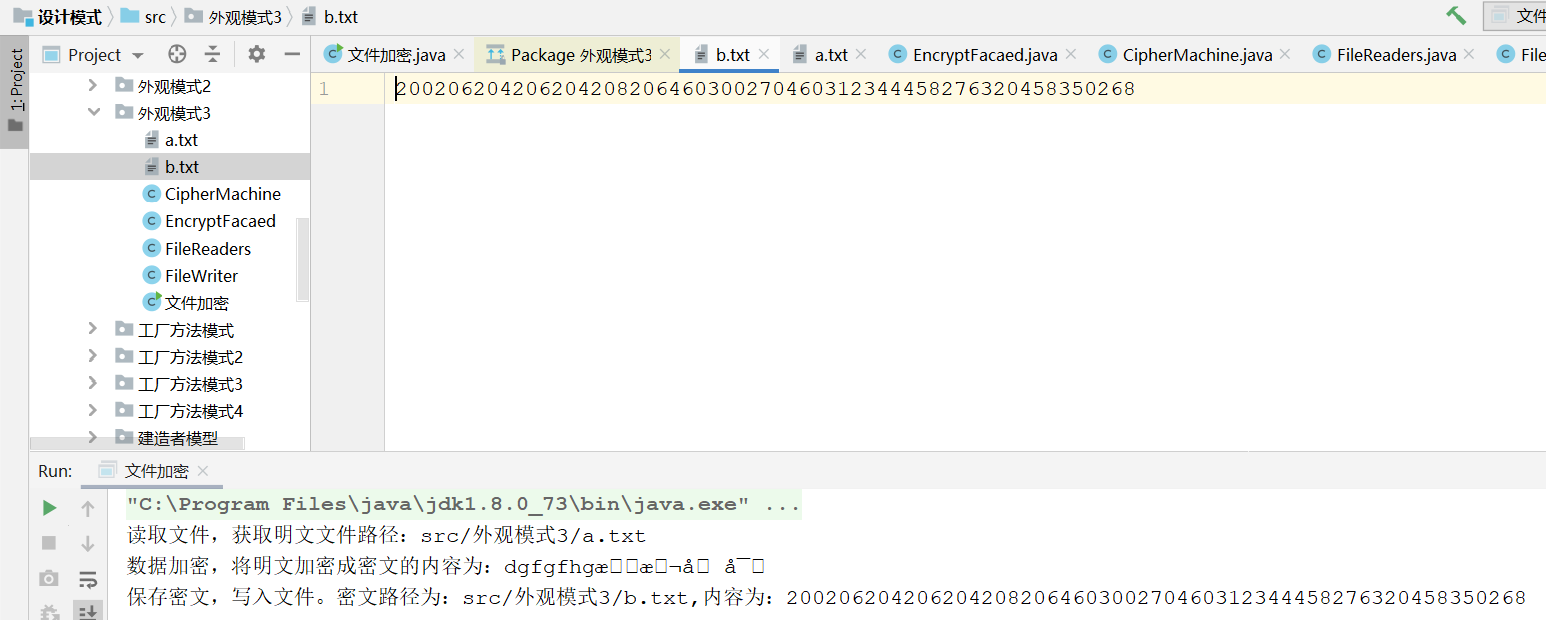
结果截图：



1. txt内容为：



1. txt内容为：



1. 实验分析与总结

在外观模式中，外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。外观模式又称为门面模式，它是一种对象结构型模式。

外观模式要求一个子系统的外部与其内部的通信通过一个统一的外观对象进行，外观类将客户端与子系统的内部复杂性分隔开，使得客户端只需要与外观对象打交道，而不需要与子系统内部的很多对象打交道。

外观模式主要优点在于对客户屏蔽子系统组件，减少了客户处理的对象数目并使得子系统使用起来更加容易，它实现了子系统与客户之间的松耦合关系，并降低了大型软件系统中的编译依赖性，简化了系统在不同平台之间的移植过程；其缺点在于不能很好地限制客户使用子系统类，而且在不引入抽象外观类的情况下，增加新的子系统可能需要修改外观类或客户端的源代码，违背了“开闭原则”。

外观模式适用情况包括：要为一个复杂子系统提供一个简单接口；客户程序与多个子系统之间存在很大的依赖性；在层次化结构中，需要定义系统中每一层的入口，使得层与层之间不直接产生联系。