

西安电子科技大学

Java 课程上机报告



题 目： 上机作业 4

学 号： 17180210027

姓 名： 李 欣

教 师： 孙 聪

日期： 2019/11/12

一、上机内容或题目：

8-2 实现一个程序，该程序的输入是一个目录字符串和一个文件扩展名的字符串，程序递归地搜索该目录和其各级子目录，在这些目录中查找所有扩展名与指定的扩展名相同的文件，将这些文件的相对路径名记录下来并向控制台输出。

8-7 实现两个程序，第一个程序读入一个文件 plain.txt，将读入的每个字节的数值加 5，然后输出到另一个文件 cipher.txt 中；第二个程序从 cipher.txt 中读取每个字节，将每个字节数值减 5，再输出到文件 decrypt.txt 中。比较 plain.txt 与 decrypt.txt 中的内容。

10-6 在很多系统中，需要日志管理功能，该功能通常可由一个类实现，在系统生命周期中，日志管理类仅存在一个对象实例，实现将各个功能线程日志写入日志文件的功能，要求：

1. 实现日志管理类 Logger，实现唯一的日志文件写入日志的功能，并使用 Singleton 设计模式实现在系统生命周期中仅存在该类的唯一对象。
2. 实现多个功能线程，各个线程排他性地使用日志管理对象的日志写入功能，每隔一段随机的时间就执行写入日志的动作。系统终止时观察被写入的日志文件。

二、上机步骤及实验结果：

8-2

1. 新建一个静态私有方法，实现查找目录下所有目录和文件路径信息，存储在一个字符串列表中：

```
private static List<String> getAllFilePaths(File filePath, List<String> filePaths){
    File[] files = filePath.listFiles();
    if(files == null){
        return filePaths;
    }
    for(File f:files){
        if(f.isDirectory()){
            filePaths.add(f.getPath());
            getAllFilePaths(f, filePaths);
        }else{
            filePaths.add(f.getPath());
        }
    }
    return filePaths;
}
```

2. 编写 main 方法，获取指定目录下 .java 文件的路径，并把结果输出到控制台：

```
public class FileSearch{
    Run | Debug
    public static void main(String[] args){
        List<String> paths = new ArrayList<String>();
        String name = ".java";
        paths = getAllFilePaths(new File("./"),paths);
        for(String path:paths){
            if(path.indexOf(name) != -1){
                System.out.println(path);
            }
        }
    }
}
```

3. 运行结果:

```
Cmdr
E:\大学课程学习\Java程序设计\第四次上机
λ javac FileSearch.java

E:\大学课程学习\Java程序设计\第四次上机
λ java FileSearch
.\Decode.java
.\Encode.java
.\FileSearch.java
.\master\book-code\chap01\HelloWorld.java
.\master\book-code\chap02\ArrayCopy.java
.\master\book-code\chap02\ArrayLength.java
.\master\book-code\chap02\AssignConversion.java
.\master\book-code\chap02\BinaryConversion.java
.\master\book-code\chap02\BreakAndContinueWithLabel.java
.\master\book-code\chap02\CastRoundingNum.java
.\master\book-code\chap02\CharConst.java
.\master\book-code\chap02\EnhancedFor.java
.\master\book-code\chap02\PrimitiveConst.java
.\master\book-code\chap02\ShiftRight.java
.\master\book-code\chap02\ShortCircuit.java
.\master\book-code\chap02\UnaryConversion.java
.\master\book-code\chap02\UnaryConversion2.java
.\master\book-code\chap02\VariablesAndLocalVarInit.java
.\master\book-code\chap03\Downcasting.java
.\master\book-code\chap03\ImportStatic.java
.\master\book-code\chap03\inheritance\ConstructSubObj.java
.\master\book-code\chap03\inheritance\ConstructSubObj2.java
.\master\book-code\chap03\inheritance\TestInheritance.java
.\master\book-code\chap03\InstanceInitializer.java
.\master\book-code\chap03\NotOverriding.java
.\master\book-code\chap03\overloading\AmbiguousOverloading.java
.\master\book-code\chap03\overloading\ConstructorOverloading.java
.\master\book-code\chap03\overloading\TestOverloading.java
.\master\book-code\chap03\PackageImport.java
.\master\book-code\chap03\ParameterPass.java
.\master\book-code\chap03\pkg1\A.java
.\master\book-code\chap03\pkg1\C.java
.\master\book-code\chap03\pkg2\PrivateVsPackage.java
.\master\book-code\chap03\pkg2\ProtectedVsPackageAndPublic.java
.\master\book-code\chap03\pkg2\PublicVsPackage.java
```

8-7

1. 编写加密 Encode 类, 读取 plain.txt 文件, 判断是否存在, 存在的话打开 cipher.txt 文件, 然后定义一个输入流和输出流分别读取 plain.txt 文件和写入 cipher.txt 文件, 将读入的数据转化为 byte 类型, 然后加 5 后存入 buffer2, 然后将 buffer2 输出到 cipher.txt 文件:

```

1  import java.util.*;
2  import java.io.*;
3
4  public class Encode{
    Run | Debug
5      public static void main(String[] args) throws Exception {
6          String fileUrl = "plain.txt";
7          File file = new File(fileUrl);
8          String path = file.getPath();
9          if(!file.exists()){
10             return;
11         }
12         String destFile = "cipher.txt";
13         File dest = new File(destFile);
14         InputStream in = new FileInputStream(fileUrl);
15         OutputStream out = new FileOutputStream(destFile);
16         byte[] buffer = new byte[1024];
17         int r;
18         byte[] buffer2=new byte[1024];
19         while (( r= in.read(buffer)) > 0) {
20             for(int i=0;i<r;i++)
21             {
22                 byte b=buffer[i];
23                 buffer2[i]=(byte)(b+(byte)5);
24             }
25             out.write(buffer2, 0, r);
26             out.flush();
27         }
28         in.close();
29         out.close();
30         System.out.println("Encode success!");
31     }
32 }

```

加密结果:

The screenshot shows a Windows command prompt window titled 'Cmder' with the following commands and output:

```

E:\大学课程学习\Java程序设计\第四次上机
λ javac Encode.java

E:\大学课程学习\Java程序设计\第四次上机
λ java Encode
Encode success!

```

Below the command prompt, two text files are open:

- cipher.txt - 记事本**: Contains the encrypted text: `Of {f%nx%fs%nsyjwtjxyns1%htzwxj&`
- plain.txt - 记事本**: Contains the original text: `Java is an interesting course!`

2. 编写加密 Decode 类，读取 cipher.txt 文件，判断是否存在，存在的话打开 decrypt.txt 文件，然后定义一个输入流和输出流分别读取 cipher.txt 文件和写入 decrypt.txt 文件，将读入的数据转化为 byte 类型，然后减 5 后存入 buffer2，然后将 buffer2 输出到 decrypt.txt 文件：

```
E: > 大学课程学习 > Java程序设计 > 第四次上机 > Decode.java > Decode > main(String[])
1  import java.util.*;
2  import java.io.*;
3
4  public class Decode{
    Run | Debug
5      public static void main(String[] args) throws Exception {
6          String fileUrl = "cipher.txt";
7          File file = new File(fileUrl);
8          String path = file.getPath();
9          if(!file.exists()){
10             return;
11         }
12         String destFile = "decrypt.txt";
13         File dest = new File(destFile);
14         InputStream in = new FileInputStream(fileUrl);
15         OutputStream out = new FileOutputStream(destFile);
16         byte[] buffer = new byte[1024];
17         int r;
18         byte[] buffer2=new byte[1024];
19         while (( r= in.read(buffer)) > 0) {
20             for(int i=0;i<r;i++)
21             {
22                 byte b=buffer[i];
23                 buffer2[i]=(byte)(b-(byte)5);
24             }
25             out.write(buffer2, 0, r);
26             out.flush();
27         }
28         in.close();
29         out.close();
30         System.out.println("Decode success!");
31     }
32 }
```

运行结果：

```
E:\大学课程学习\Java程序设计\第四次上机
λ javac Decode.java

E:\大学课程学习\Java程序设计\第四次上机
λ java Decode
Decode success!

decrypt.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Java is an interesting course!

plain.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Java is an interesting course!|
```

解密后的数据与加密前的数据一致，证明成功完成了该题目的要求。

10-6:

1. 日志管理类 Logger :

Logger 所对应的属性文件在安装 jdk 目录下的 jre/lib/logging.properties

logging.properties 文件中的

handlers= java.util.logging.ConsoleHandler 将日志内容输出到控制台

handlers= java.util.logging.FileHandler 将日志文件输出到文件中

handlers=

java.util.logging.ConsoleHandler, java.util.logging.FileHandler 将日志内容同时输出到控制台和文件中

Limit the message that are printed on the console to INFO and above.

java.util.logging.ConsoleHandler.level = INFO // 日志输出级别

java.util.logging.FileHandler.pattern = %h/java%u.log 为生成的输出文件名称指定一个模式。

模式由包括以下特殊组件的字符串组成，则运行时要替换这些组件：

"/" 本地路径名分隔符

"%t" 系统临时目录

"%h" "user.home" 系统属性的值

"%g" 区分循环日志的生成号

"%u" 解决冲突的惟一号码

“%%” 转换为单个百分数符号 “%”

如果未指定 “%g” 字段，并且文件计数大于 1，那么生成号将被添加到所生成文件名末尾的小数点后面。

`java.util.logging.FileHandler.limit = 50000` 限制文件的大小，以字节为单位

`java.util.logging.FileHandler.count = 1` 指定有多少输出文件参与循环（默认为 1）。

`java.util.logging.FileHandler.formatter` =

`java.util.logging.XMLFormatter`

指定要使用的 `Formatter` 类的名称（默认为 `java.util.logging.XMLFormatter`）。另外一个 是：

`java.util.logging.SimpleFormatter`。`XMLFormatter` 是以 xml 样式输出，`SimpleFormatter` 是以普通样式输出。

`java.util.logging.FileHandler.append` 指定是否应该将 `FileHandler` 追加到任何现有文件上（默认为 `false`）。

.. `Logger`： 应用程序进行日志记录调用的主要实体。`Logger` 对象用于记录特定系统或应用程序的消息。

.. `LogRecord`： 用于在日志框架和单个记录处理程序之间传递记录请求。

.. `Handler`： 日志数据的最终输出处理器。它将 `LogRecord` 对象导出到各种目标，包括内存、输出流、控制台、文件和套接字。多种 `Handler` 子类可供用于这种用途。

.. `Level`： 定义一组标准的记录级别，可用于控制记录的输出。可以把程序配置为只输出某些级别的记录，而忽略其他级别的输出。

.. `Filter`： 精细过滤、控制记录的内容，比记录级别所提供的控制准确得多。

记录 API 支持通用的过滤器机制，这种机制允许应用程序代码添加任意过滤器以便控制记录的输出。

.. `Formatter`： 为 `LogRecord` 对象的格式化提供支持。

.. LogManager: Java Logging 框架中唯一的、全局的对象, 用于维护与 Logger 记录器及日志服务的一系列共享的数据结构及状态。它负责整个日志框架的初始化、维护一组全局性的 Handle 对象、维护一个树形结构的 Logger 的名字空间、监测日志框架配置文件的改变从而重新读入并应用相关的参数以及负责程序停止运行时整个日志框架的清理工作。

2. Singleton 设计模式 ---- 单例模式 (Singleton)

(1) 介绍: 也叫单子模式, 是一种常用的软件设计模式。在应用这个模式时, 单例对象的类必须保证只有一个实例存在。许多时候整个系统只需要拥有一个的全局对象, 这样有利于我们协调系统整体的行为。比如在某个服务器程序中, 该服务器的配置信息存放在一个文件中, 这些配置数据由一个单例对象统一读取, 然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。

(2) 实现单例模式的思路是: 一个类能返回对象一个引用 (永远是同一个) 和一个获得该实例的方法 (必须是静态方法, 通常使用 getInstance 这个名称); 当我们调用这个方法时, 如果类持有的引用不为空就返回这个引用, 如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用; 同时我们还将该类的构造函数定义为私有方法, 这样其他处的代码就无法通过调用该类的构造函数来实例化该类的对象, 只有通过该类提供的静态方法来得到该类的唯一实例。

代码如下:


```

1  import java.util.logging.*;
2  import java.io.*;
3  import java.util.Date;
4
5  class MyLogger1 extends Thread {
6      static void Logger1() {
7          Logger logger1 = Logger.getAnonymousLogger();
8          int rand1 = (int)(Math.random() * 100);
9          for(int i = 0; i < 100; i++) {
10             if(i == rand1) {
11                 logger1.log(Level.INFO, "logger1's log record\n");
12             }
13         }
14     }
15 }
16
17 class MyLogger2 {
18     static int i = 1;
19     private static MyLogger2 mylogger2 = new MyLogger2();
20     public MyLogger2() {
21     }
22
23     public static MyLogger2 getInstance2() {
24         return mylogger2;
25     }
26
27     public void print() {
28         int rand2 = (int)(Math.random() * 100);
29         for(int t = 0; t < 100; t++) {
30             if(t == rand2) {
31                 Date date = new Date();
32                 System.out.println(date.getTime() + "\n" + "logger2's log record\n");
33             }
34         }
35     }
36 }

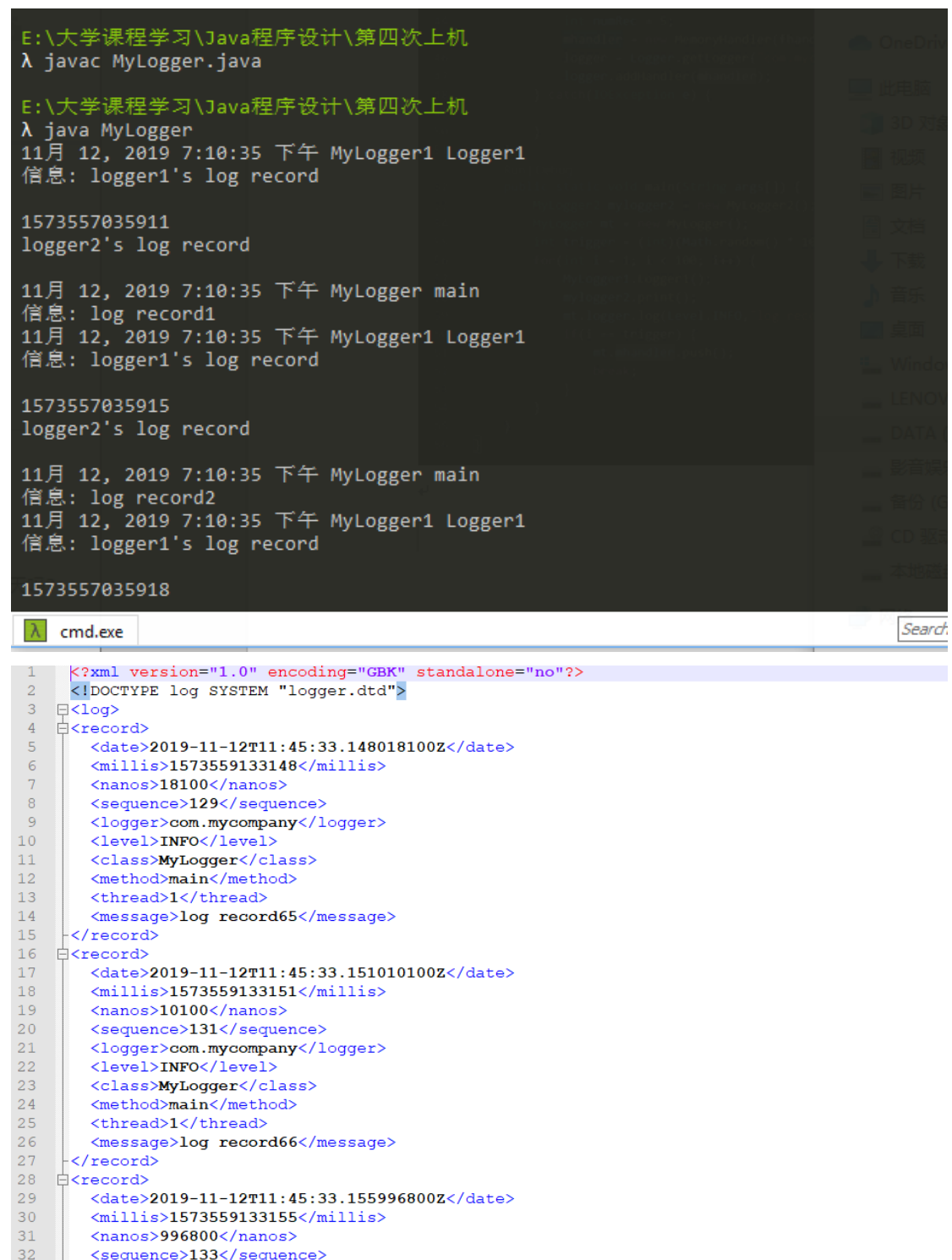
```

```

36
37 public class MyLogger {
38     FileHandler fhandler;
39     Logger logger;
40     MemoryHandler mhandler;
41     MyLogger() {
42         try {
43             fhandler = new FileHandler("my.log", true);
44             int numRec = 5;
45             mhandler = new MemoryHandler(fhandler, numRec, Level.OFF);
46             logger = Logger.getLogger("com.mycompany");
47             logger.addHandler(mhandler);
48         } catch(IOException e) {
49         }
50     }
51 }
52
53 Run | Debug
54 public static void main(String args[]) {
55     MyLogger2 mylogger2 = new MyLogger2();
56     MyLogger mt = new MyLogger();
57     int trigger = (int)(Math.random() * 100);
58     for(int i = 1; i < 100; i++) {
59         MyLogger1.Logger1();
60         mylogger2.print();
61         mt.logger.log(Level.INFO, "log record" + i);
62         if(i == trigger) {
63             mt.mhandler.push();
64             break;
65         }
66     }
67 }

```

运行结果：



The screenshot displays a Windows command prompt window and an XML file editor. The command prompt shows the execution of a Java program named MyLogger, which outputs log records. The XML editor shows the resulting log records in XML format, including timestamps, milliseconds, nanoseconds, sequence numbers, logger names, levels, classes, methods, threads, and messages.

```
E:\大学课程学习\Java程序设计\第四次上机
λ javac MyLogger.java

E:\大学课程学习\Java程序设计\第四次上机
λ java MyLogger
11月 12, 2019 7:10:35 下午 MyLogger1 Logger1
信息: logger1's log record

1573557035911
logger2's log record

11月 12, 2019 7:10:35 下午 MyLogger main
信息: log record1
11月 12, 2019 7:10:35 下午 MyLogger1 Logger1
信息: logger1's log record

1573557035915
logger2's log record

11月 12, 2019 7:10:35 下午 MyLogger main
信息: log record2
11月 12, 2019 7:10:35 下午 MyLogger1 Logger1
信息: logger1's log record

1573557035918
```

```
1 <?xml version="1.0" encoding="GBK" standalone="no"?>
2 <!DOCTYPE log SYSTEM "logger.dtd">
3 <log>
4 <record>
5 <date>2019-11-12T11:45:33.148018100Z</date>
6 <millis>1573559133148</millis>
7 <nanos>18100</nanos>
8 <sequence>129</sequence>
9 <logger>com.mycompany</logger>
10 <level>INFO</level>
11 <class>MyLogger</class>
12 <method>main</method>
13 <thread>1</thread>
14 <message>log record65</message>
15 </record>
16 <record>
17 <date>2019-11-12T11:45:33.151010100Z</date>
18 <millis>1573559133151</millis>
19 <nanos>10100</nanos>
20 <sequence>131</sequence>
21 <logger>com.mycompany</logger>
22 <level>INFO</level>
23 <class>MyLogger</class>
24 <method>main</method>
25 <thread>1</thread>
26 <message>log record66</message>
27 </record>
28 <record>
29 <date>2019-11-12T11:45:33.155996800Z</date>
30 <millis>1573559133155</millis>
31 <nanos>996800</nanos>
32 <sequence>133</sequence>
```

三、上机总结

本次上机熟悉了 Java 的文件流的操作，能熟练使用文件读写操作，掌握 logger 类的使用和 Singleton 设计模式，学以致用，受益颇深。