信息系统开发（Java）
实验：补充理论知识

王晓东
wangxiaodong@ouc.edu.cn

中国海洋大学

March 30, 2023

大纲

模拟扑克牌游戏的洗牌
- 枚举类型
- Comparable 接口

Java 命令行文件操作程序设计
- 文件操作
- Java 执行外部程序
- 配置文件：属性信息导入/导出

接下来…

模拟扑克牌游戏的洗牌
- 枚举类型
- Comparable 接口

Java 命令行文件操作程序设计
- 文件操作
- Java 执行外部程序
- 配置文件：属性信息导入/导出

接下来…

什么是枚举类型

Java SE 5.0 开始，引入了一种新的引用数据结构**枚举类型**。枚举类型均自动继承 java.lang.Enum 类，使用一组常量值来表示特定的数据集合，该集合中数据的数目确定（通常较少），且这些数据只能取预先定义的值。

```
1  public enum Week {
2    MON, TUE, WED, THU, FRI, SAT, SUN
3  }
```

☛ 无枚举类型前如何解决上述需求？

一般采用声明多个整型常量的做法实现枚举类的功能。

```
1  public class Week {
2    public static final int MON = 1;
3    public static final int TUE = 2;
4    ...
5  }
```

组合使用枚举类型与 switch

课程配套代码 ▸ package sample.advance.enumclass

☛ 注意

1. case 字句必须省略其枚举类型的前缀，即只需要写成 case SUN:，而不允许写成 case Week.SUN:，否则编译出错。
2. 不必担心系统无法搞清这些常量名称的出处，因为 switch 后的小括号中的表达式已经指明本次要区分处理的是 Week 类型常量。

组合使用枚举类型与 switch

课程配套代码 ▸ package sample.advance.enumclass

☛ 注意

1. case 字句必须省略其枚举类型的前缀，即只需要写成 case SUN:，而不允许写成 case Week.SUN:，否则编译出错。
2. 不必担心系统无法搞清这些常量名称的出处，因为 switch 后的小括号中的表达式已经指明本次要区分处理的是 Week 类型常量。

接下来…

Comparable 接口

java.lang.Comparable 接口中定义的 compareTo() 方法用于提供对其实现类的对象进行整体排序所需的比较逻辑，所为的排序可以理解为按照某种标准来比较对象的大小以确定其次序。

- 实现类基于 compareTo() 方法的排序被称为自然排序。
- compareTo() 方法被称为它的自然比较方法，具体的排序原则可由实现类根据需要而定。

方法格式

```
1    int compareTo(Object o) {

3    }
```

Comparable 接口

课程配套代码 ▸ sample.setlistmap.NatrualOrderingSample.java

❖ 对上述程序的几点说明

1. 用户在重写 compareTo() 方法以定制比较逻辑时，需要确保其与等价性判断方法 equals() 保持一致，即确保条件“(x.compareTo(y) == 0) == (x.equals(y))”永远成立，否则逻辑上不合理。所以上例同时重写了 equals() 方法。
2. 为保证能够实现元素的排序功能，集合容器要求向其加入的对象元素必须是 Comparable 接口的实现类的实例，否者程序运行时会抛出造型异常。
3. Comparable 接口并不专用于集合框架。

Comparable 接口

课程配套代码 ▸ sample.setlistmap.NatrualOrderingSample.java

❖ 对上述程序的几点说明

1. 用户在重写 compareTo() 方法以定制比较逻辑时，需要确保其与等价性判断方法 equals() 保持一致，即确保条件“(x.compareTo(y) == 0) == (x.equals(y))”永远成立，否则逻辑上不合理。所以上例同时重写了 equals() 方法。
2. 为保证能够实现元素的排序功能，集合容器要求向其加入的对象元素必须是 Comparable 接口的实现类的实例，否者程序运行时会抛出造型异常。
3. Comparable 接口并不专用于集合框架。

Comparable 接口

课程配套代码 ▸ sample.setlistmap.NatrualOrderingSample.java

❖ 对上述程序的几点说明

1. 用户在重写 compareTo() 方法以定制比较逻辑时，需要确保其与等价性判断方法 equals() 保持一致，即确保条件“(x.compareTo(y) == 0) == (x.equals(y))”永远成立，否则逻辑上不合理。所以上例同时重写了 equals() 方法。
2. 为保证能够实现元素的排序功能，集合容器要求向其加入的对象元素必须是 Comparable 接口的实现类的实例，否者程序运行时会抛出造型异常。
3. Comparable 接口并不专用于集合框架。

接下来…

模拟扑克牌游戏的洗牌
- 枚举类型
- Comparable 接口

Java 命令行文件操作程序设计
- 文件操作
- Java 执行外部程序
- 配置文件：属性信息导入/导出

接下来…

模拟扑克牌游戏的洗牌
- 枚举类型
- Comparable 接口

Java 命令行文件操作程序设计
- 文件操作
- Java 执行外部程序
- 配置文件：属性信息导入/导出

文件操作对象

❖ 创建 File 类对象

java.io 包中定义与数据输入、输出功能有关的类，包括提供文件操作功能的 File 类。

```
public File(String pathname)
```

通过给定的路径/文件名字符串创建一个新 File 实例。

```
public File(String parent, String child)
```

通过分别给定的 parent 路径名和 child 文件名（也可以是子路径名）或字符串来创建一个新 File 实例。

使用 File 类

课程配套代码 ➧ sample.commandline.FileOperationSample.java

File 类的主要方法 ❶

❖ 文件/目录名操作

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()

❖ 设置和修改操作

- boolean delete()
- void deleteOnExit()
- boolean createNewFile()
- setReadOnly()
- boolean renameTo(File dest)

File 类的主要方法 ❷

❖ 测试操作

- boolean exists()
- boolean canWrite()
- boolean canRead()
- boolean isFile()
- boolean isDirectory()
- boolean isAbsolute()

❖ 目录操作

- boolean mkdir()
- String[] list()
- File[] listFiles()

❖ 获取常规文件信息操作

- long lastModified()
- long length()

文件 I/O 有关读写类

❖ 常见的文本文件 I/O 操作的类

java.io.FileReader 类

提供 read() 方法以字符为单位从文件中读入数据。

java.io.FileWrite 类

提供 write() 方法以字符为单位向文件写出数据。

java.io.BufferedReader 类

提供 readLine() 方法以行为单位读入一行字符。

java.io.PrintWriter 类

提供 print() 和 println() 方法以行为单位写出数据。

读取文件内容

CODE ReadFile.java

```
1   import java.io.*;
2
3   public class ReadFileSample {
4     public static void main (String[] args) {
5       String fname = "test.txt";
6       File f = new File(fname);
7
8       try {
9         FileReader fr = new FileReader(f); // 1
10        BufferedReader br = new BufferedReader(fr);
11        String s = br.readLine();
12        while (s != null) { // 2
13          System.out.println("读入: " + s);
14          s = br.readLine(); }
15        br.close();
16      } catch (FileNotFoundException e1) {
17        System.err.println("File not found: " + fname);
18      } catch (IOException e2) {
19        e2.printStackTrace();
20      }
21    }
22  }
```

读取文件内容

☛ 上述代码几点说明

1. FileReader 的构造方法被重载过，接受以字符串形式给出的文件名，上述代码等价于：

```
1       FileReader fr = new FileReader("test.txt");
```

2. 使用 BufferedReader 的 readLine() 方法读文件，遇到文件结尾则返回 null，而不是""，与读取键盘输入遇到空回车时返回空字符串的情况不同。

输出内容到文件

CODE ▸ WriteFileSample

```
1   import java.io.*;

3   public class TestWriteFile {
4     public static void main (String[] args) {
5       File file = new File("tt.txt");
6       try {
7         InputStreamReader is = new InputStreamReader(System.in);
8         BufferedReader in=new BufferedReader(is);
9         FileWriter fw = new FileWriter(file);
10        PrintWriter out = new PrintWriter(fw);
11        String s = in.readLine();
12        while(!s.equals("")) { // 从键盘逐行读入数据输出到文件
13          out.println(s);
14          s = in.readLine();
15        }
16        in.close(); // 关闭 BufferedReader 输入流
17        out.close(); // 关闭连接文件的 PrintWriter 输出流
18      } catch (IOException e) {
19        e.printStackTrace();
20      }
21    }
22  }
```

对上述代码的几点说明

1. 写文件时如果目标文件不存在，程序运行不会出错，而是自动创建该文件，但如果目标路径不存在，则会出错。
2. 写文件操作结束后一定要关闭输出流，即关闭文件，否则被操作文件仍处于打开状态，不安全。

文件过滤

文件过滤，即只检索和处理符合特定条件的文件。最常见的为按照文件类型（后缀）进行划分，如查找.class 或.xml 文件。
文件过滤可以使用 java.io.FileFilter 接口，该接口只定义了一个抽象方法 accept。

```
boolean accept(File pathname)
```

测试参数指定的 File 对象对应的文件（目录）是否应该保留在文件列表中，即不被过滤。
在实际应用中，可以定义该接口的一个实现类，重写其中的 accept() 方法，在方法中添加文件过滤逻辑，然后创建一个该实现类的对象作为参数传递给 File 对象的文件列表方法 list()，在 list() 方法执行过程中会自动调用前者的 accept() 方法来过滤文件。

使用 FileFilter 实现文件过滤

课程配套代码 ➧ sample.commandline.filefilter

接下来…

模拟扑克牌游戏的洗牌
- 枚举类型
- Comparable 接口

Java 命令行文件操作程序设计
- 文件操作
- Java 执行外部程序
- 配置文件：属性信息导入/导出

Java 执行外部程序两种方式

Java 有两种方式来调用其它程序：Runtime 和 ProcessBulider。

- Runtime.getRuntime().exec()
- new ProcessBuilder().start()
- Runtime 和 ProcessBulider 提供不同的方式来启动程序，设置启动参数、环境变量和工作目录。两种方法都会返回一个用于管理操作系统进程的 Process 对象。

Runtime 方式

Runtime.getRuntime().exec() 用于调用外部可执行程序或系统命令，并重定向外部程序的标准输入、标准输出和标准错误到缓冲池。

课程配套代码 ➧ sample.outerprogram.RuntimeExecSample

```
1     Process process = Runtime.getRuntime()
2       .exec(new String[] { "ls", "-l", "/Users/xiaodong" });
3     BufferedReader reader = new BufferedReader(
4       new InputStreamReader(process.getInputStream(), "UTF-8"));

6     process.destroy();
7     process.waitFor();
```

ProcessBuilder 方式

每个 ProcessBuilder 实例管理一个进程属性集合。start() 方法使用这些属性创建一个新的流程实例。可以从同一个实例多次调用 start() 方法，以创建具有相同或相关属性的新子进程。

课程配套代码 ➧ sample.outerprogram.ProcessBuilderSample

```
1     List<String> params = new ArrayList<String>();
2     params.add("ls");
3     params.add("-la");
4     params.add("/Users/xiaodong");

6     ProcessBuilder processBuilder = new ProcessBuilder(params);
7     processBuilder.redirectErrorStream(true);
8     Process process = processBuilder.start();
9     BufferedReader br = new BufferedReader(
10      new InputStreamReader(process.getInputStream()));

12    int exitCode = process.waitFor();
13    System.out.println("exitCode = " + exitCode);
```

接下来…

属性信息的导入/导出

如果要永久记录用户自定义的属性，可以采用 Properties 类的 load()/store() 方法进行属性的导入/导出操作，即将属性信息写出到文件中和从文件中读取属性信息到程序。

CODE ◗ SaveProperties.java

```
1  import java.io.FileWriter;
2  import java.util.Properties;
3  public class SaveProperties {
4    public static void main(String[] args) {
5      try {
6        Properties ps = new Properties();
7        ps.setProperty("name", "Kevin");
8        ps.setProperty("password", "12345");
9        FileWriter fw = new FileWriter("props.txt");
10       ps.store(fw, "loginfo");
11       fw.close();
12     } catch (Exception e) {
13       e.printStackTrace();
14     }
15   }
16 }
```

属性信息的导入/导出

CODE ▸ LoadProperties.java

```
1  import java.io.FileWriter;
2  import java.util.Properties;
3  public class LoadProperties {
4    public static void main(String[] args) {
5      try {
6        Properties ps = new Properties();
7        FileReader fr = new FileReader("props.txt");
8        ps.load(fr);
9        fr.close();
10       ps.list(System.out);
11     } catch (Exception e) {
12       e.printStackTrace();
13     }
14   }
15 }
```

File: props.txt

```
#loginfo
#Sun Nov 04 21:20:17 CST 2012
password=12345
name=Kevin
```

Stdout

```
-- listing properties --
password=12345
name=Kevin
```

THE END

wangxiaodong@ouc.edu.cn