Java09

Task1

Q1

代码：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class java09_Q1 {
    public static void main(String[] args){
        File src=new File("doro.jpg");
        File tgt=new File("doro_copy.jpg");
        //吐槽一下：IDEA的相对路径的当前工作目录竟然就是项目根目录，
        //而Markdown的是.md文件所在的文件夹。
        FileInputStream fis=null;
        FileOutputStream fos=null;
        try{
            fis=new FileInputStream(src);
            fos=new FileOutputStream(tgt);
            byte[] bytes=new byte[1024];
            int lengthOfRead;
            while((lengthOfRead= fis.read(bytes))!=-1){
                fos.write(bytes,0,lengthOfRead);
            }

        }catch(IOException e){
            System.out.println(e.getMessage());
        }finally{
            try{
                if(fos!=null){
                    fos.close();
                }
            }catch(IOException e){
                System.out.println(e.getMessage());
            }
            try{
                if(fis!=null) {
                    fis.close();
                }
            }catch(IOException e){
                System.out.println(e.getMessage());
            }
        }
```

```
        }
    }
```

关于读写效率

查阅资料得知：字节数组读写比单字节读写效率高得多，主要原因是大幅减少了系统调用次数和改善了内存访问模式。

效率差异原因：

因素	单字节读写	字节数组读写
系统调用次数	每个字节1次调用	每缓冲区1次调用
上下文切换	频繁切换，开销大	大幅减少切换
缓存效率	缓存局部性差	缓存局部性好
磁盘I/O	随机小数据块	连续大数据块
CPU利用率	大量时间在等待	批量处理效率高

性能差异典型值：

- **小文件(100KB)**：缓冲读写快 **5-10倍**
- **中等文件(10MB)**：缓冲读写快 **50-100倍**
- **大文件(100MB+)**：缓冲读写快 **100-500倍**

输入输出流的开关顺序

查阅资料得知：通常需遵守“后开先关”的原则以尽可能防止出现`close()`无法被正常执行而导致的一些不良影响（然而真正的解决方案似乎是使用`try-with-resources`或者确保每个`close()`调用都有独立且安全的异常

处理)

Task2

Q2

代码:

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class java09_Q2 {
    public static void main(String[] args) {
        File src=new File("name.txt");
        File tgt=new File("name_sorted.txt");
        List<String> nameList=new ArrayList<>();
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(new
FileInputStream(src), StandardCharsets.UTF_8));
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(tgt), StandardCharsets.UTF_8));

            String s;
            while ((s = br.readLine()) != null) {
                if(s.trim().isEmpty()){
                    //过滤空行
                    continue;
                }
                nameList.add(s.trim());
            }

            nameList.sort(new Comparator<String>() {
                @Override
                public int compare(String o1, String o2) {
                    return o1.compareTo(o2);//自然排序
                }
            });
//该段代码会在name_sorted.txt的末尾留下一个空行
//             for (String name : nameList) {
//                 bw.write(name, 0, name.length());
//                 bw.newLine();
//                 bw.flush();
//             }
            for(int i=0;i<nameList.size();i++){
                bw.write(nameList.get(i));
                if(i< nameList.size()-1){
                    //最后一行不换行
                    bw.newLine();
```

```
                }
                bw.flush();//刷新缓冲区
            }

            //后开先关
            bw.close();
            br.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

运行程序后name_sorted.txt文件中的内容：

```
冯十三
卫十六
吴十
周八
孙七
张三
李四
杨二十
沈十八
王五
王十二
蒋十七
褚十五
赵六
郑十一
钱九
陈十四
韩十九
```

为什么不建议手写换行符

因为不同操作系统中的换行符不同，而newLine()能自动根据操作系统输出合适的换行符。

- Unix/Linux/macOS(10.0之后)：\n
- macOS(10.0之前)：\r
- Windows/DOS：\r\n

Task3

Q3

代码：

```java
//补充完善后的Student类
package java09_Q3;

import java.io.Serializable;

public class Student implements Serializable {
    private static final long serialVersionUID = 1L;

    private Integer id;
    private String name;
    private Integer gender; //1表示男性 2表示女性
    private String phone;

    public Student(Integer id, String name, Integer gender, String phone) {
        this.id = id;
        this.name = name;
        this.gender = gender;
        this.phone = phone;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getGender() {
        return gender;
    }

    public void setGender(Integer gender) {
        this.gender = gender;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
```

```
    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", gender=" + gender +
                ", phone='" + phone + '\'' +
                '}';
    }
}
```

```
package java09_Q3;

import java.io.*;

public class Main {
    public static void main(String[] args){
        Student student = new Student(1, "doro", 2, "114514");
        File file = new File("student.dat");
        try {
            ObjectOutputStream oos=new ObjectOutputStream(new
FileOutputStream(file));
            oos.writeObject(student);
            oos.flush();//刷新缓冲区，确保数据写入文件
            oos.close();
            //用完一个关一个
            //分离文件读写操作
            //某些操作系统不允许同时读写同一个文件
            ObjectInputStream ois=new ObjectInputStream(new
FileInputStream(file));
            Student doro=(Student) ois.readObject();
            ois.close();
            //关
            System.out.println(doro.toString());
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

为什么最好显式声明serialVersionUID

如果**不显式声明**，系统会自动根据类的结构（类名、方法、字段等）计算一个哈希值作为UID，之后一旦类的结构发生任何改变（比如增加一个字段），这个自动生成的UID就会改变，导致之前序列化的对象无法被反序列化。**显式声明**一个serialVersionUID之后，即使类结构发生变化， 仍可以通过保持相同的UID来正常反序列化旧对象。