TASK1

1.

```
List<String> strings = List.of("I", "am", "a", "list", "of", "Strings");
//使用List.of()方法来初始化一个包含指定元素的不可变列表。

Stream<String> stream = strings.stream();
//调用流API的方法，例如我们希望最多有4个元素
Stream<String> limit = stream.limit(4);
//最后我们打印
System.out.println("limit = " + limit);
//结果:limit = java.util.stream.SliceOps$1@6d311334
//limit 是Stream对象的一个引用，不能直接打印引用本身，需要通过某种方法获取到里面的数据
limit.forEach(System.out::println);
//通过终端方法forEach输出，::表示引用了System.out的println方法，接收一个参数并打印，调用该
//方法时由底层的迭代器负责提供流中的每个元素给println方法。
```

2. 代码解释

```
List<Integer> numbers = Arrays.asList(1,2,5,4,8);
List<Integer> squaresList = numbers.stream()//创建一个流
        .map(i -> i * i)     //将原来列表中每个元素映射成它的平方，保存到一个新的流里面
        .sorted((x, y) -> y - x)     //降序排序，保存到一个新的流里
        .collect(Collectors.toList());//终端操作，先触发所有的中间操作处理这个流，最
后将流收集到一个新的列表squaresList中并返回一个列表，终端操作之后，流就被消耗掉了，流无法
再被使用
for(Integer i : squaresList) {
    System.out.print(i+" ");
}
```

Lamba表达式

基本语法：实现的这个接口中的抽象方法中的形参列表 -> 抽象方法的处理

当只有一个抽象方法时，可以用Lambda表达式来代替传统的匿名类实现。

对于上面代码

```
.map(i -> i*i)
//括号里需要传入一个M接口的实例化对象，该接口只有一个抽象方法
//i -> i*i 实际上是创建了一个匿名内部类实现这个接口，实现该接口的抽象方法的时候传入i,并返
回i*i；编译器会自动识别传入的类型

//上一题的歌曲名排序也可以用Lamba表达式化简
    Comparator<String> firstcharacter = new Comparator<String>(){
            public int compare(String s1, String s2){
                char first = s1.charAt(0);
                char seconed = s2.charAt(0);
                return first-seconed;
                //定义一个匿名内部类来实现Comparator接口，在类中重写了compare方法
                //通过返回正数，0，和负数表示两个首字符的大小关系，符合compare返回的参
数的要求
```

```
            }
        };
        Collections.sort(songs, firstcharacter);
//化简后

Collections.sort(songs,(s1,s2)->s1.charAt(0)-s2.charAt(0))//一行解决问题
```

TASK2

1. 串行化：也叫序列化，将对象转化成可以存储和传输的形式，这个形式通常是字节流。

```
public class Song implements Serializable
//首先要在Song类实现Serializable接口，这个接口没有定义任何方法，只是提供标记作用，表示该对象可以被序列化和反序列化
//
for(int i=0;i<n;i++){
    Song s = song.get(i);//将列表中的对象提取出来
    ObjectOutputStream oos = null;//实例化对象，方便finally语句中能调用oos.close()
    try {
        System.out.println("正在写入："+i+".ser");
        oos = new ObjectOutputStream(new FileOutputStream("./src/KTV/SongFile/"+i+".ser"));
        //ObjectOutputStream  将对象序列化并写入字节输出流中。
        //ObjectOutputStream 的构造函数需要一个 OutputStream 类型的参数，用于指定序列化后的字节流应该写入到哪里。
        //FileOutputStream是OutputStream的子类，所以可以传入它的实例化对象指定写入到的位置。
        //FileOutputStream   将字节输出流中的数据写入文件。
        oos.writeObject(s);//完成写入工作
    } catch (IOException e) {
        //ObjectOutputStream的构造函数会抛出IOException，FileOutputStream会抛出FileNotFoundException,由于
        //FileNotFoundException的父类就是IOException，所以只要捕获父类就行了
        e.printStackTrace();
        System.out.println("写入文件失败");
    }finally{
        try {
            oos.close();//在finally代码块中的程序一定会执行。需要关闭关闭流释放系统资源。但是这个关闭又会抛出一个异常，又要再捕获一次
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }


//读取文件
for(int i=0;i<n;i++){
    ObjectInputStream ois = null;
    try {
        ois = new ObjectInputStream(new FileInputStream("./src/KTV/SongFile/"+i+".ser"));
        Song s2 = (Song) ois.readObject();
        //readObject方法返回的是Object对象，为了调用Song类的方法，需要将Object对象向下转型
```

```
        //该方法会抛出ClassNotFoundException，不和IOE构成继承关系，需要用他们呢共同的父类Exception来捕获
        System.out.println(s2.getInfo());
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("读取文件失败");
    }finally{
        try {
            ois.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

进阶挑战，文件IO

```
//写入文件
```

```
for (int i = 0; i < n; i++) {
    Song s = song.get(i);
    FileWriter fw = null;
    //FileWriter是字符输入流，将字符数据写入到文本文件中，它也有异常需要捕获
    try {
        fw = new FileWriter("./src/KTV/SongFile/"+i+".txt");
        fw.write(s.WriteIn());//调用歌曲对象s的WriteIn方法将歌曲信息按一定格式写入
        System.out.println("....正在写入："+i+".txt");
    } catch (IOException e) {
        throw new RuntimeException(e);
    }finally {
        try {
            fw.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

}
```

```

```

```
//读取文件
String c1[] = new String[5];
c1[0] = "歌曲";
c1[1] = "创作者";
c1[2] = "流派";
c1[3] = "发行时间";
c1[4] = "时长";
int k=0;
for(int i=0;i<n;i++){
    System.out.println("--------------------------------");
    BufferedReader br = null;
    String line = null;
    k=0;
```

```
    try {
        br = new BufferedReader(new FileReader("./src/KTV/SongFile/"+i+".txt"));
        // BufferedReader是具有缓冲功能的输入流，能提高读取效率，它是一个包装流，包装了基础
的字符输入流，并在此基础上实现了
        //缓冲功能，用它是因为能读一行，与写入的格式相匹配
        while((line=br.readLine())!=null){
            System.out.println(c1[k]+":"+line);
            k++;
            //c[k]存储了第k行信息的名称
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```