# Java09

## Task1

### 流API

```
List<String> strings = List.of("I", "am", "a", "list", "of", "Strings");
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);
//最后我们打印结果
System.out.println("limit = " + limit);
```

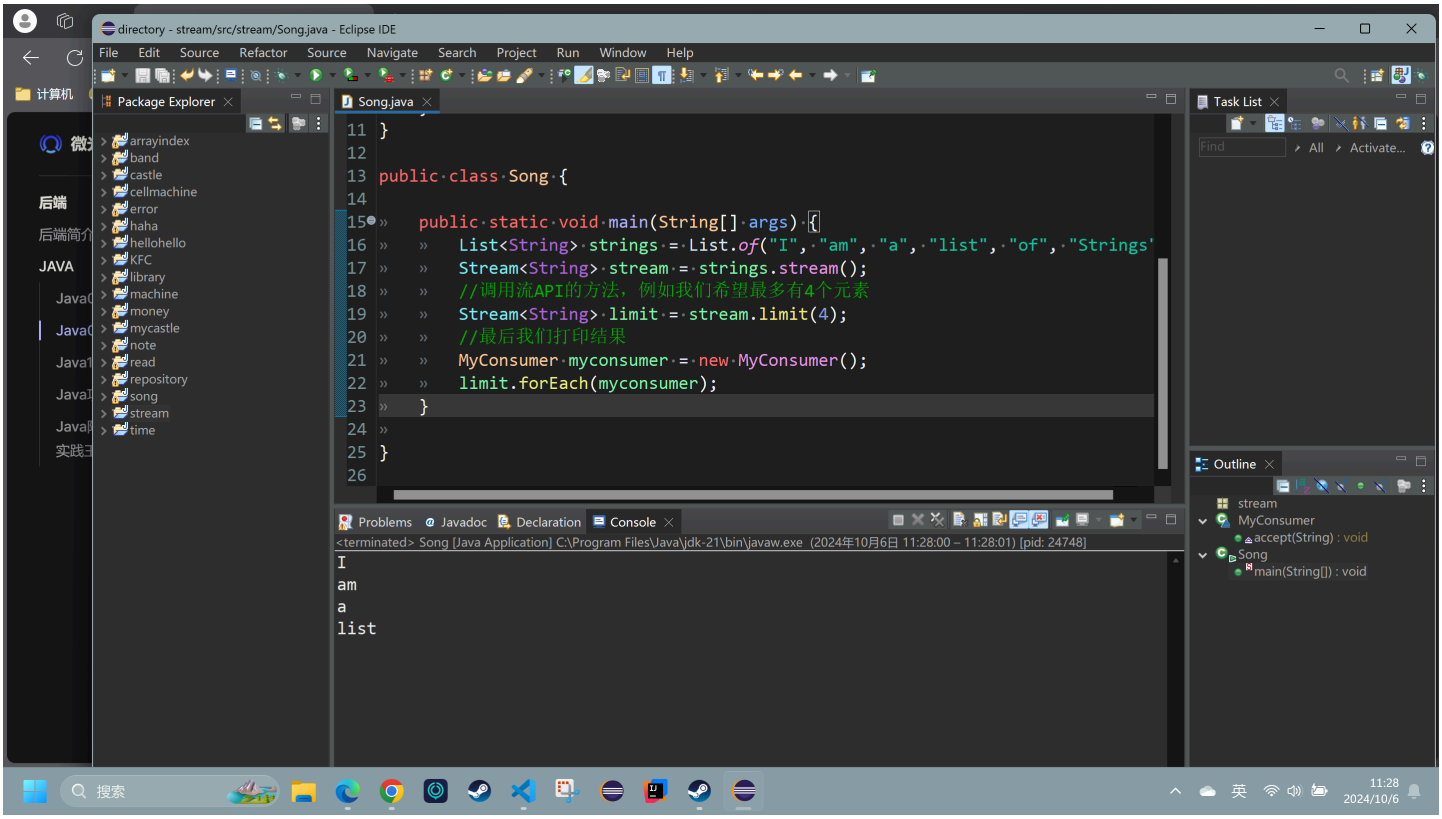
直接输出 会返回一串英文 和直接输出数组或者引用传递的对象一样 可以知道这应该是表示一个地址 原因就是流的执行分为中间操作和终端操作 中间操作是惰性执行的 每次中间操作会返回一个新流 只有当终端操作执行时 中间操作链才会被触发 依次执行 以这个为例 limit并不会立刻执行截取前四个元素的操作 而是返回一个新流 这个新流指向的是它对象的实例 直接输出会返回这个对象实例的地址 而这个对象实例也并不是执行了截取前四个元素的结果 而是它知道自己将来会执行 这里的limit相当于只是做了标记 当遇到终端操作时才会被执行
了解了原理之后 我们需要做的就是为它添加终端操作就行了 因为我的目的是要输出limit 那么有两个终端操作是可用的 一个是forEach 遍历流并输出 另一个是collect 把流的元素收集到一个集合中 collect下一题的例子有 我这里就用forEach吧

forEach
看编译器的参数表可以知道 这里需要放一个Consumer类 查了一下Consemer是一个接口 那么这里我只需要创建一个Consumer的实现类就可以了 把这个类的对象传给forEach forEach只起到遍历作用 而后面的参数表是类的对象 那谁来输出呢??? 查了一下Consumer的API

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

那么可以推知forEach应该会自动调用后面对象的accept函数 也就是我们在重写对象的accept函数时加上输出就可以了 试一试吧:

```java
package stream;

import java.util.List;
import java.util.stream.Stream;
import java.util.function.Consumer;

class MyConsumer implements Consumer<String> {
        public void accept(String s) {
                System.out.println(s);
        }
}

public class Song {

        public static void main(String[] args) {
                List<String> strings = List.of("I", "am", "a", "list", "of", "Strings");
                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);
                //最后我们打印结果
                MyConsumer myconsumer = new MyConsumer();
                limit.forEach(myconsumer);
        }

}
```



还有一种匿名内部类的方法:

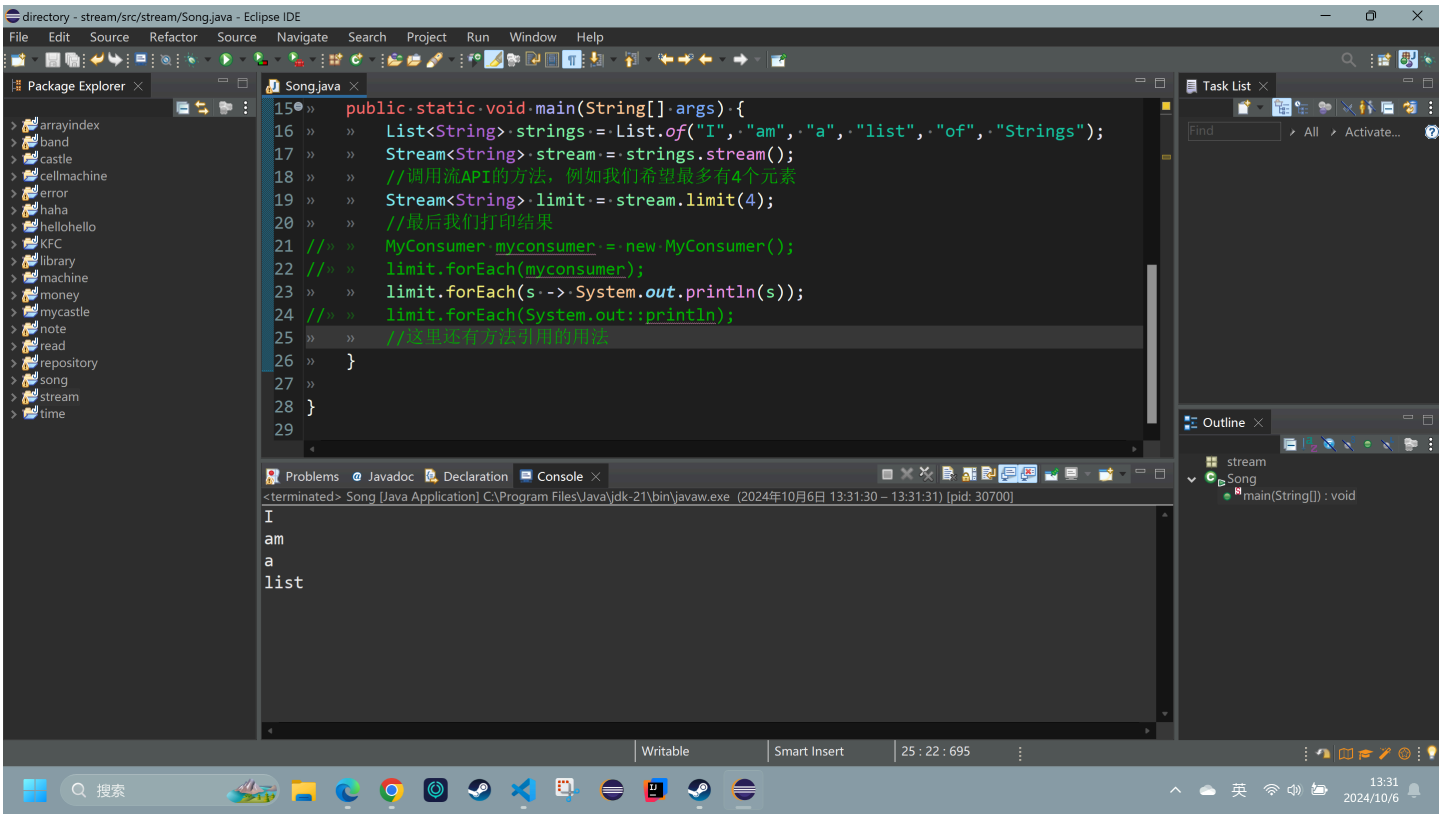
```
Consumer<String> consumer = new Consumer<String>() {
    @Override
    public void accept(String s) {
        System.out.println(s);
    }
};
```

这里应该是直接创建了一个实现了Consumer接口的类的对象实例 把这个对象实例交给了consumer对象 感觉有点像向上造型 虽然代码简化了 但是这个类的名字都没显示标注出来 后续如果再用到调用也不方便...

另外这里用lambda表达式似乎更方便 这里引入函数式接口的概念 就是内部只有一个方法的接口 可以看到上文的Consumer接口的API前面@FunctionalInterface就标注了这个一个函数式接口 而lambda方法 个人理解就是一种简化表达的方法 而又有实现函数式接口的功能:

```java
package stream;

import java.util.List;
import java.util.stream.Stream;
import java.util.function.Consumer;

//class MyConsumer implements Consumer<String> {
//	public void accept(String s) {
//		System.out.println(s);
//	}
//}

public class Song {

	public static void main(String[] args) {
		List<String> strings = List.of("I", "am", "a", "list", "of", "Strings");
		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);
		//最后我们打印结果
//		MyConsumer myconsumer = new MyConsumer();
//		limit.forEach(myconsumer);
		limit.forEach(s -> System.out.println(s));
//		limit.forEach(System.out::println);
		//这里还有方法引用的用法
	}

}
```

这里还意外发现流是一次性的 被输出过一次就不能再次被用了




下面就是流的堆叠了

```
List<Integer> squaresList = numbers.stream()
                                                    .map(i -> i * i)      //你可能会对这种表达式有
                                                    .sorted((x, y) -> y - x)      //你可能会对这种
                                                    .collect(Collectors.toList());
```

numbers可能是一个包含整数的集合吧 然后用stream转化它为流 下面

- map 中间操作 对每个元素都执行一个操作 这里是对元素平方
- sorted 中间操作 对元素排序(x, y) y - x 大于0 y在前;小于0 x在前 等于0 顺序不变 也就是把大的数放在前面 同理 (x, y) x - y 就是把小数放前面了
- collect 终端操作 把流终端元素收集到容器里

> 堆叠的原理上面已经说过了 这里就不再赘述

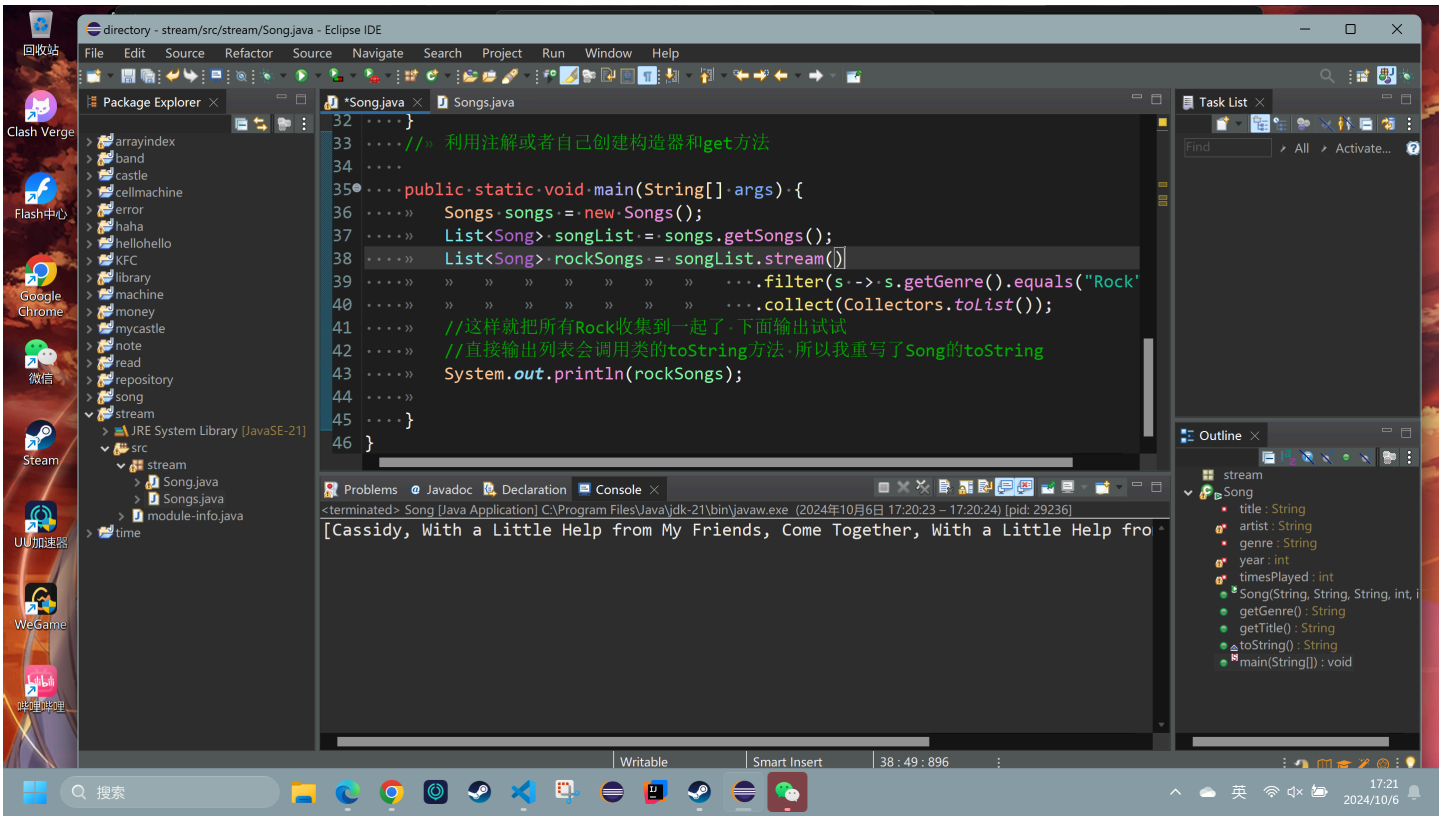
# 进阶挑战_应用流API

1. Songs类我没做修改 下面是Song类

```java
package stream;

import java.util.List;
import java.util.stream.Collectors;

public class Song{
    private String title;
    private String artist;
    private String genre;
    private int year;
    private int timesPlayed;

    public Song(String title,String artist,String genre,int year,int timesPlayed) {
        this.title = title;
        this.artist = artist;
        this.genre = genre;
        this.year = year;
        this.timesPlayed = timesPlayed;
    }

    public String getGenre() {
        return genre;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public String toString() {
        return getTitle();
    }
    //  利用注解或者自己创建构造器和get方法

    public static void main(String[] args) {
        Songs songs = new Songs();
        List<Song> songList = songs.getSongs();
        List<Song> rockSongs = songList.stream()
                                                                .filter(s -> s.getGenre().ec
                                                                .collect(Collectors.toList()
        //这样就把所有Rock收集到一起了 下面输出试试
        //直接输出列表会调用类的toString方法 所以我重写了Song的toString
        System.out.println(rockSongs);
```

```
    }
```

这里没换行 输出可能不是太好看

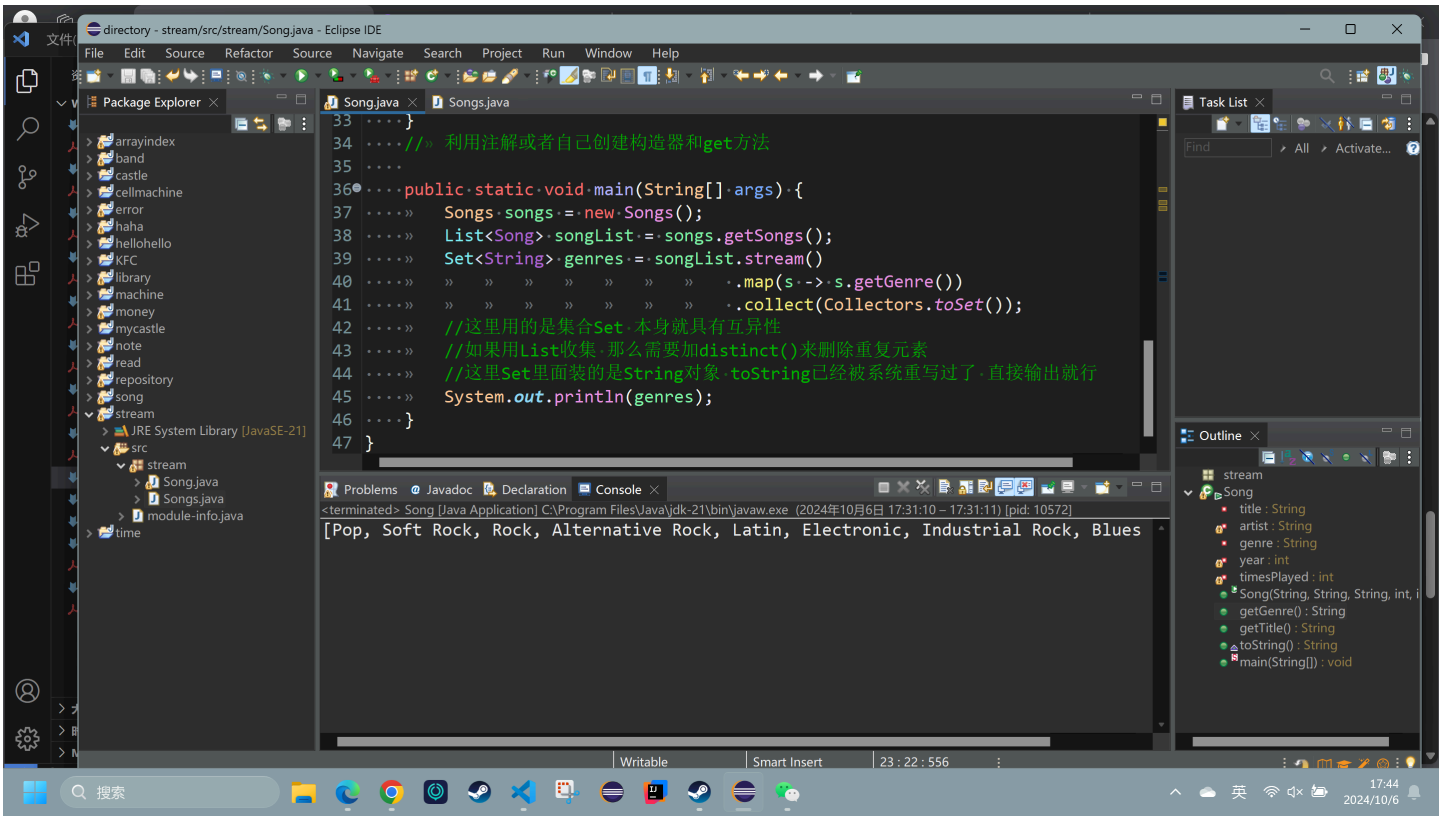



2. 列出所有流派

```java
package stream;

import java.util.List;
import java.util.stream.Collectors;
import java.util.Set;

public class Song{
    private String title;
    private String artist;
    private String genre;
    private int year;
    private int timesPlayed;

    public Song(String title,String artist,String genre,int year,int timesPlayed) {
        this.title = title;
        this.artist = artist;
        this.genre = genre;
        this.year = year;
        this.timesPlayed = timesPlayed;
    }

    public String getGenre() {
        return genre;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public String toString() {
        return getTitle();
    }
    //  利用注解或者自己创建构造器和get方法

    public static void main(String[] args) {
        Songs songs = new Songs();
        List<Song> songList = songs.getSongs();
        Set<String> genres = songList.stream()
                                                                .map(s -> s.getGenre())
                                                                .collect(Collectors.toSet());
        //这里用的是集合Set 本身就具有互异性
        //如果用List收集 那么需要加distinct()来删除重复元素
        //这里Set里面装的是String对象 toString已经被系统重写过了 直接输出就行
        System.out.println(genres);
```

```
    }
```

这里出现了新用法map 其实就是用来转换对象的类型吧




# Task2

- 下面的Songs类均是题目中给出的 没做修改 所以这里就不复制过来赘述了

# 串行化

```
package stream;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.List;
import java.util.stream.Collectors;
import java.util.Set;

public class Song implements Serializable{
    private String title;
    private String artist;
    private String genre;
    private int year;
    private int timesPlayed;

    public Song(String title,String artist,String genre,int year,int timesPlayed) {
        this.title = title;
        this.artist = artist;
        this.genre = genre;
        this.year = year;
        this.timesPlayed = timesPlayed;
    }

    public String getGenre() {
        return genre;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public String toString() {
        return getTitle();
    }
    //  利用注解或者自己创建构造器和get方法

    public static void main(String[] args) {
        Songs songList = new Songs();
```

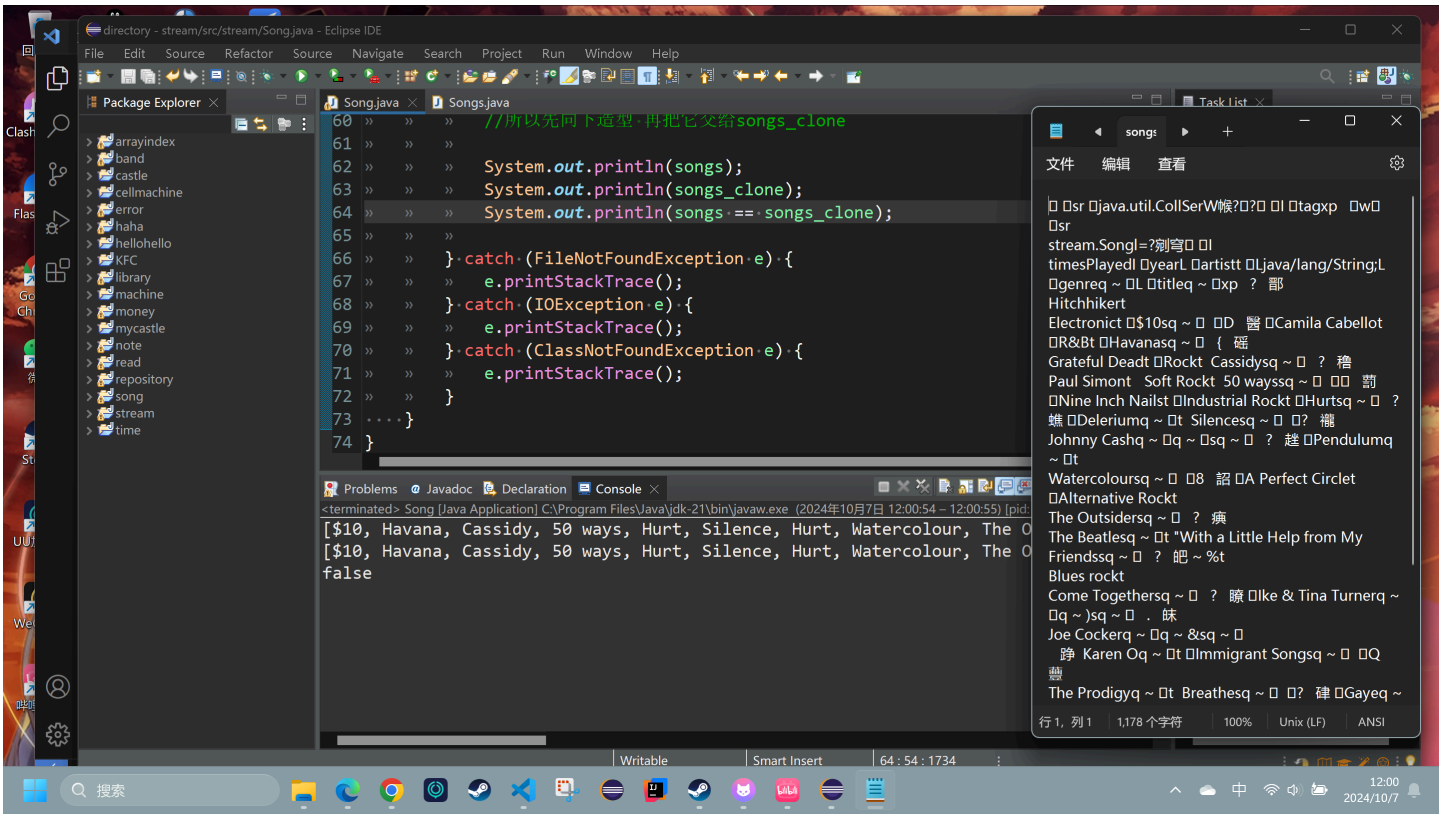
```
        List<Song> songs = songList.getSongs();

        try {
                        FileOutputStream FO = new FileOutputStream("songs.ser");
                        ObjectOutputStream out = new ObjectOutputStream(FO);
                        out.writeObject(songs);
                        out.close();
                        //把songs写进到文件里

                        FileInputStream FI = new FileInputStream("songs.ser");
                        ObjectInputStream in = new ObjectInputStream(FI);
                        List<Song> songs_clone = (List<Song>)in.readObject();
                        in.close();
                        //把文件里的songs读进来
                        //读入后的结果是Object 但我们知道他其实就是List
                        //所以先向下造型 再把它交给songs_clone

                        System.out.println(songs);
                        System.out.println(songs_clone);
                        System.out.println(songs == songs_clone);

                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
    }
}
```

先让Song实现Serializable 这个接口没有需要实现的方法 这里声明就表示这个对象的实例可以被串行化

> 其实这里也能看出来 一个容器如果它内部的对象实例都可以被串行化 那么这个容器也能被串行化

之后就是把文件输出到songs.ser中 再把它读入 交给songs_clone 最后另外判断了一下songs = songs_clone 结果为false 表明每次读入都会新建一个内容相同的对象

FileOutputStream只能将字节输出到文件里面 而ObjectOutputStream将对象转化为了字节




这个记事本直接打开文件是乱码 应该是要专门的二进制工具打开啥的

# 进阶挑战_文件I/O

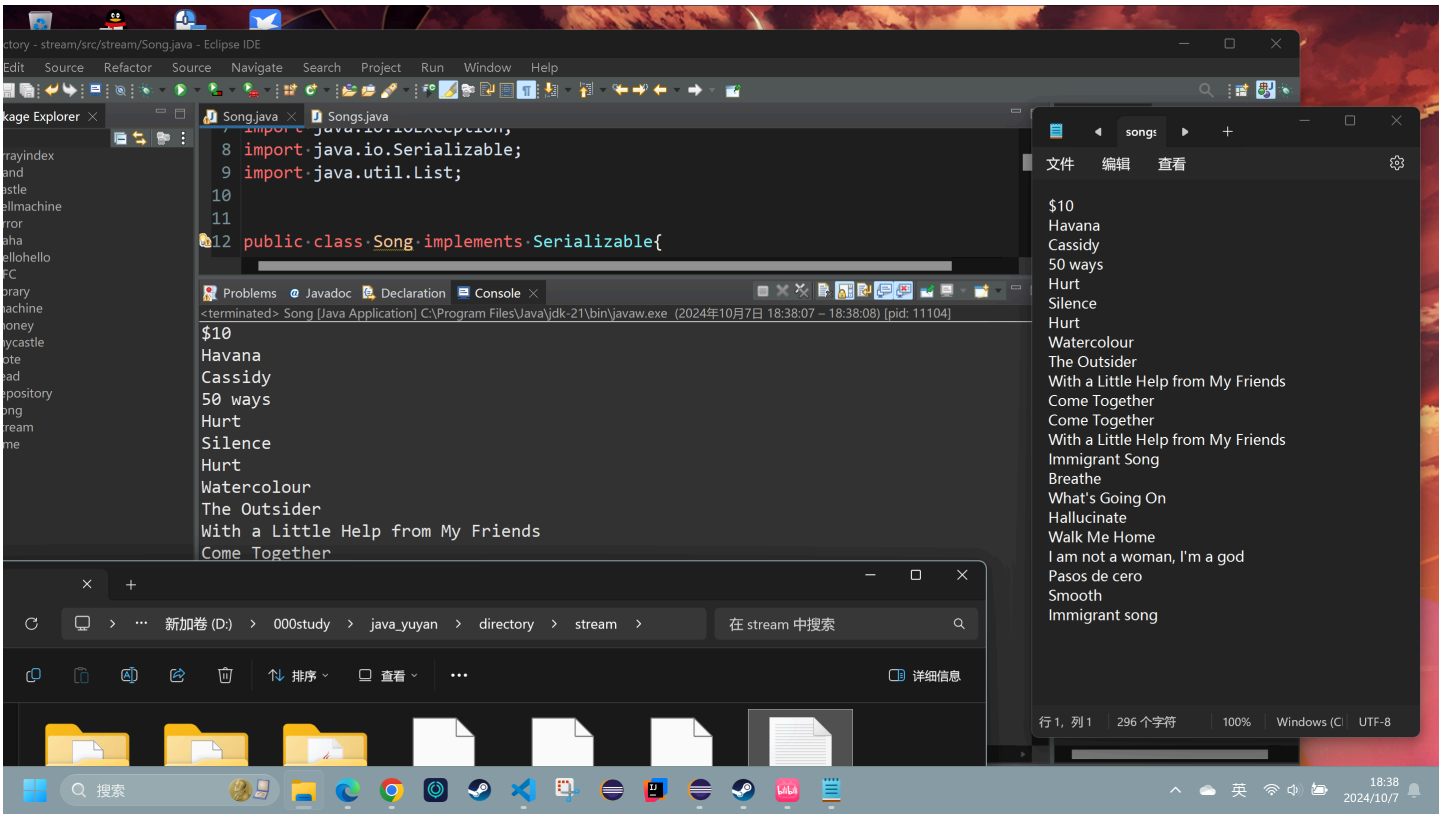
```
package stream;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Serializable;
import java.util.List;


public class Song implements Serializable{
    private String title;
    private String artist;
    private String genre;
    private int year;
    private int timesPlayed;

    public Song(String title,String artist,String genre,int year,int timesPlayed) {
        this.title = title;
        this.artist = artist;
        this.genre = genre;
        this.year = year;
        this.timesPlayed = timesPlayed;
    }

    public String getGenre() {
        return genre;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public String toString() {
        return getTitle();
    }
    //  利用注解或者自己创建构造器和get方法

    public static void main(String[] args) {
        Songs songList = new Songs();
        List<Song> songs = songList.getSongs();
                try(FileWriter FW = new FileWriter("songsTitle.txt");
```

```
                        BufferedWriter writer = new BufferedWriter(FW)  ){
                        for(Song i : songs) {
                                writer.write(i.getTitle());
                                writer.newLine();
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }

                try(FileReader FR = new FileReader("songsTitle.txt");
                        BufferedReader reader = new BufferedReader(FR)  ){
                        String line = null;
                        while((line = reader.readLine()) != null) {
                                //这里比较特殊 用这个方法给字符串赋值会有返回值
                                //返回line的内容
                                System.out.println(line);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }

    }
}
```

其实方法和上一题差不多 只是把输出的从对象换成了字符串 把FileOutputStream换成了FileWriter 后面读入的道理也一样 这里稍微做了一下改进 不用手动close了 而是选用了Java08中的try-with-resource方法

> 如果中途抛异常的话 程序会在那个地方中止 所以后面的close就不能正常执行了 但是如果用try-with-resource 文件资源总会自动关闭 会更安全一点

下面是运行结果:


Edit Source Refactor Source Navigate Search Project Run Window Help
Song.java
Songs.java
import java.io.Serializable;
import java.util.List;
public class Song implements Serializable{
Problems Javadoc Declaration Console
<terminated> Song [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (2024年10月7日 18:38:07 – 18:38:08) [pid: 11104]
$10
Havana
Cassidy
50 ways
Hurt
Silence
Hurt
Watercolour
The Outsider
With a Little Help from My Friends
Come Together
songs
文件 编辑 查看
$10
Havana
Cassidy
50 ways
Hurt
Silence
Hurt
Watercolour
The Outsider
With a Little Help from My Friends
Come Together
Come Together
With a Little Help from My Friends
Immigrant Song
Breathe
What's Going On
Hallucinate
Walk Me Home
I am not a woman, I'm a god
Pasos de cero
Smooth
Immigrant song
行 1，列 1 296 个字符 100% Windows (C UTF-8
新加卷 (D:) 000study java_yuyan directory stream
在 stream 中搜索
排序 查看 详细信息
搜索
英
18:38
2024/10/7