第八题

简单挑战

```
import java.util.Collections;
import java.util.List;

public class Paixu {
public static List<String> getSongStrings(){
    List<String> songs = new ArrayList<>();
    //模拟将要处理的列表
    songs.add("sunrise");
    songs.add("noprice");
    songs.add("thanks");
    songs.add("$100");
    songs.add("havana");
    songs.add("114514");
    return songs;
}
public static void main(String[] args) {
    List<String> songs = getSongStrings();
    Collections.sort(songs);
    System.out.println(songs);
}
}
```

```
[$100, 114514, havana, noprice, sunrise, thanks]

进程已结束，退出代码为 0
```

好神奇，直接就排好了。

Comparator

```
Arrays.sort(words, new Comparator() {
@Override
public int compare(String o1, String o2) {
return Integer.compare(o1.length(), o2.length());
}
});
将这一段程序分解学习
```

Arrays和Collections

Arrays（数组）Collections（集合）
观察到有的程序用的arrays，有的用的collections。了解了一下，发现两者都可以用，但是collections可能会灵活一点

sort

sort（排序）
Arrays.sort（）是Java标准库中的一个方法，用于对数组进行排序。

Comparator

这一段程序包含了定义比较方法和使用方法的过程
return Integer.compare(o1.length(), o2.length());
如果第一个参数小于第二个参数，返回负数。 如果第一个参数等于第二个参数，返回0。 如果第一个参数大于第二个参数，返回正数。我的理解就是这个返回值会传到Arrays.sort（），然后Arrays.sort（）会利用返回值进行排序，只是排序过程的程序没有写出来。
所以说，Comparator是起比较的作用，而Arrays.sort()起排序作用

进阶挑战

```
package Paixu;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Paixu {
public static void main(String[] args) {
    List<text> songs = getSongStrings();
    Collections.sort(songs,Comparator.comparing(text::getTitle));
/*想了好久都不知道该怎么引用对象中的参数，songs.get(0).getTitle(),这个可以引用这个参数，但好像不能用在这里，上面用的这个又不怎么看得懂，（text::getTitle
*/
    System.out.println(songs.toString());

}
public static List<text> getSongStrings(){
    List<text> songs = new ArrayList<>();
    text song1=new text("Million Years Ago ","Adele",1);
    //节拍随便写的
  songs.add(song1);
    songs.add(new text("$100", "abc", 2));
    //这么写还要简单一点
    return songs;
}

}
```

。

```
package Paixu;

public class text {

    private String title;
    private String artist;
    private int bpm;

    public text(String title, String artist, int bpm) {
        this.title = title;
        this.artist = artist;
        this.bpm = bpm;
    }


public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getArtist() {
        return artist;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    public int getBpm() {
        return bpm;
    }

    public void setBpm(int bpm) {
        this.bpm = bpm;
    }
@Override
public String toString() {
    return "Song{" +
            "title='" + title + '\'' +
            ", artist='" + artist + '\'' +
            ", bpm=" + bpm +
            '}';
}

}
```

```
[Song{title='$100', artist='abc', bpm=2}, Song{title='Million Years Ago ',
 artist='Adele', bpm=1}]

进程已结束，退出代码为 0
```

写完了才发现还没有去了解泛型

toString

目的是返回一个代表对象值的 String。
需要重写，以求其返回更有用的信息。否则默认会返回一个字符串，该字符串由对象的类名、@ 符号和对象的哈希码的无符号十六进制表示组成。如图

```
Paixu.text@34a245ab, Paixu.text@7cc355be]

程已结束，退出代码为 0
```

这个也是最开始我很疑惑为什么输出不对劲的原因

@

注解作用很多：提供元数据，代码标记，编译时处理等等
@Override使用在toString（）之前，表示我会重写toString（）这个方法，如果不满足，编译器就会报错。
其实就是起了一个检查的作用。
其他注解：
@Deprecated：表示某个程序元素（类、方法、字段等）已经过时，不推荐使用。
@SuppressWarnings：告诉编译器忽略特定的编译警告。
@SafeVarargs：用于方法上，指示该方法不会对传入的参数产生任何副作用。
@FunctionalInterface：用于接口上，指示该接口是一个函数式接口，即只有一个抽象方法的接口。
还可以自定义注解

```
public @interface CustomAnnotation {
// 注解的属性
String value() default "default value";
}
```

我觉得注解和方法有一点像，感觉他们都是被设计出来执行什么功能，要用的时候就调用一下

泛型

泛型的作用

List这里的 List 是一个泛型类，尖括号 <> 内指定了 List 将要存储的元素类型是 Text 类型。作用是当你尝试向这个列表中添加非 Text 类型的对象时，编译器会报错，避免了运行错误。
除了避免添加时，加入错误的类型，还可以在需要添加不同类型时，减少代码重复，用一个泛型方法来处理多种类型的数据，而不需要为每种类型编写特定的方法。

> public T testMethod1(T t){}

比如这个方法可以被用来执行任何类型的操作

类型擦除

泛型信息只在编译时存在，在编译成.class文件后，这些泛型信息会被擦除，变成原始类型Object或者指定的边界类型。

泛型通配符

泛型通配符有 3 种形式：
<?> ：被称作无限定的通配符。
<? extends T> ：被称作有上界的通配符。表示类型参数的范围是 T 和 T 的子类
<? super T> ：被称作有下界的通配符。表示类型参数的范围是 T 和 T 的父类
泛型通配符可以实现多态和泛型类型的结合

> 做到这里发现之前学的继承，多态都忘记了，又回去听了一遍，感觉之前好多听不懂的都听懂了