# 第四次上机报告

王志强 21200100001

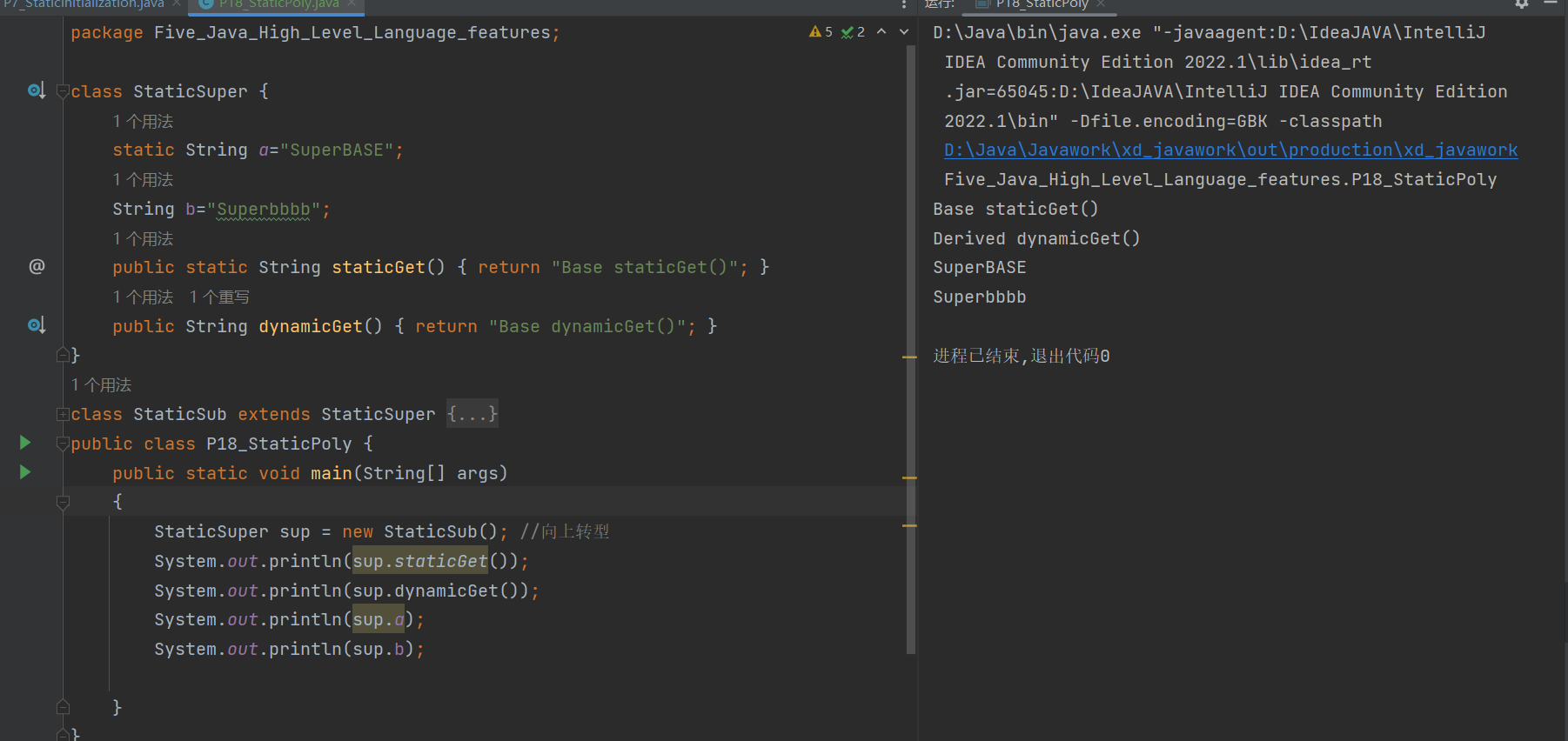
## PPT练习

## P7

1. 运行一个JAVA程序从Main中开始。可以分为两个部分：首先对类成员的加载、实例化再对普通成员的加载、实例化。基本流程是类似的
2. 类成员：先进行类的加载，对静态的成员进行初始化，如果静态成员也是一个实例化的过程的时间，依次类加载再实例化。并且对一个类的加载只能够进行一次（对对应的类成员初始化一次）；对类成员加载后进行类成员实例化的过程，对构造方法的参数绑定，然后对普通变量进行显示初始化，可进行多次。
3. 普通成员变量：类加载，也是如果有类成员先对类成员先进行上述的操作，然后才是实例化的过程，递归到最底层的父类，对父类的普通变量显示初始化完毕后执行构造方法里的内容，在对子类进行相同的操作。

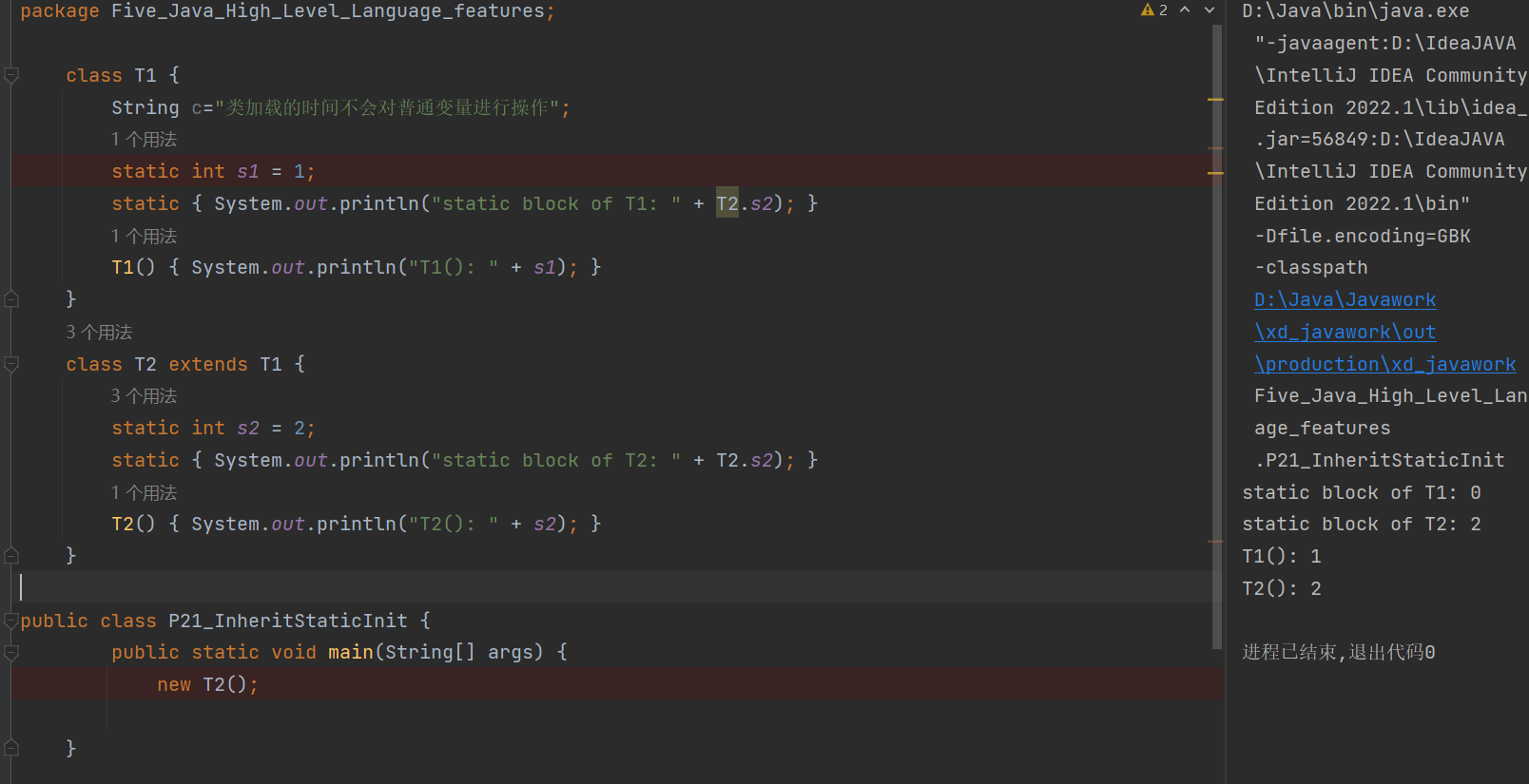
## P18

1. 在向上造型的时间使用父类引用类型的变量指向子类，如果有重写的方法或者是隐藏的成员，通过该引用访问的是普通成员是父类的变量，访问方法的时间是对自雷重写后的方法进行访问。
2. 在静态方法中不具有继承的关系，也就不存在静态方法的重写，在自雷中写出与父类有相同函数头的方法，是与父类完全不同的方法就不存在继承的关系。
3. 综上，在这个示例中重写的方法 dynamicGet，和子类中与父类独立的方法staticGetzai使用向上造型的引用访问的时间是不同的结果。重写后的gynameGet将父类中的方法覆盖，所以在访问的是子类重写的方法；而StaticGet是与父类独立的方法，在父类的引用中自然访问的是父类中的staticGet方法。在访问成员变量的时间对待普通的成员变量和静态变量，通过父类引用访问的都是父类的成员。



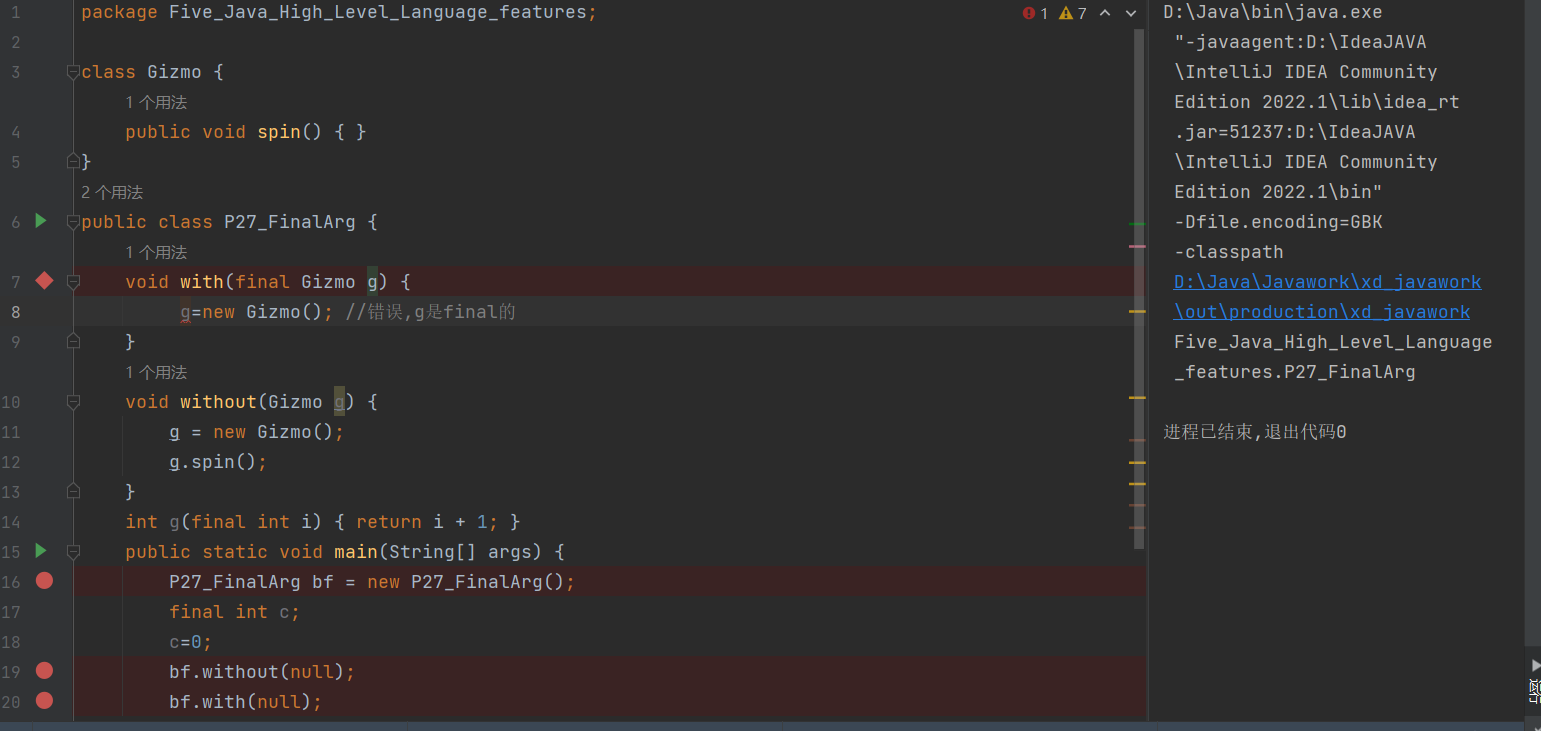
## P21

1. 在实例化一个具有静态成员的子类的而时间，首先进行类加载，对类中静态成员递归到根部，赋值为对应的‘0’，在向下显示初始化，文中在没有对子类中的成员显示初始化的时间就去调用他，所得到的的数值就是相应的‘0’，在类加载的过程中不会对普通的成员进行操作。并且类加载只进行一次。
2. 实例化一个拥有父类的子类的时间，和上述造作基本类似，但是如果实例化多个类的时间，会执行多次。



## P27

1. 类的成员变量被定义为final的时间，则变量一经赋值后就不能改变，可以使用来定义常量。
2. 如果是局部变量，对未赋初值的，在所属的方法体重的任意一个地方都可以赋初值，只能赋值一次。
3. 对于引用型的变量，引用是不能改变的，但是引用对象的数值是可以被修改的。

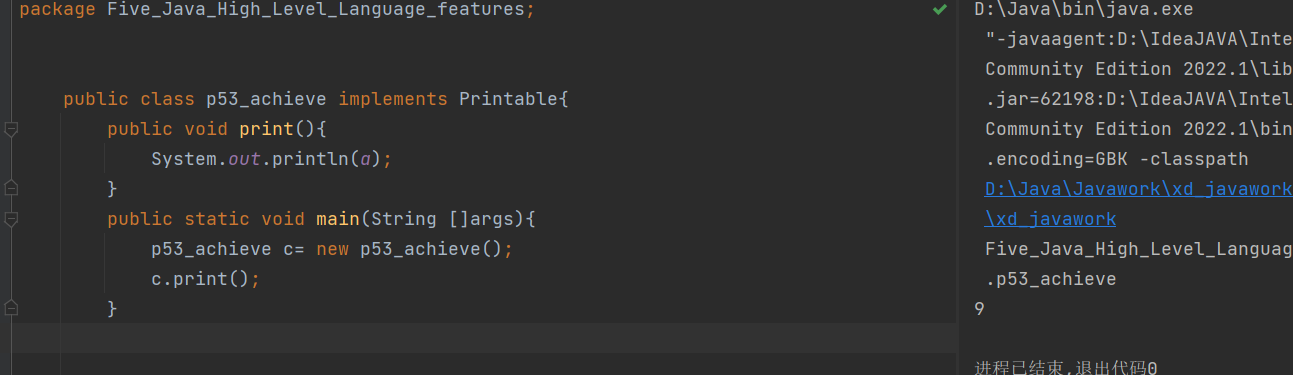
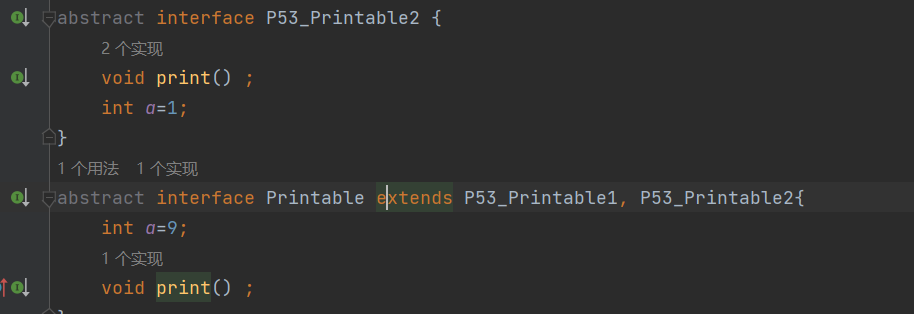


## P31

1. 在final成员变量定义后，必须对其赋初值。如果没有对其赋初值的时间，必须在其构造方法中赋值，并且只能赋值一次，赋值后不能再进行修改。

## P53

1. （1）和（3）错误：“不应该有方法体”，去除 “{}”；其他均正确。但是可以省略abstarct修饰。
2. 在接口的声明之中，对于接口默认是具有public、abstract、属性的。所以在声明一个接口的时间，写与不写都是可以的。
3. 在接口中定义的常量默认是具有public、abstract、static的属性。
4. 声明的方法是默认具有public和abstract的属性。
5. 子类接口中有同名的成员变量，和同名的成员方法的时间，子类成员方法会重写父类的成员方法，子类变量会隐藏父类的成员。



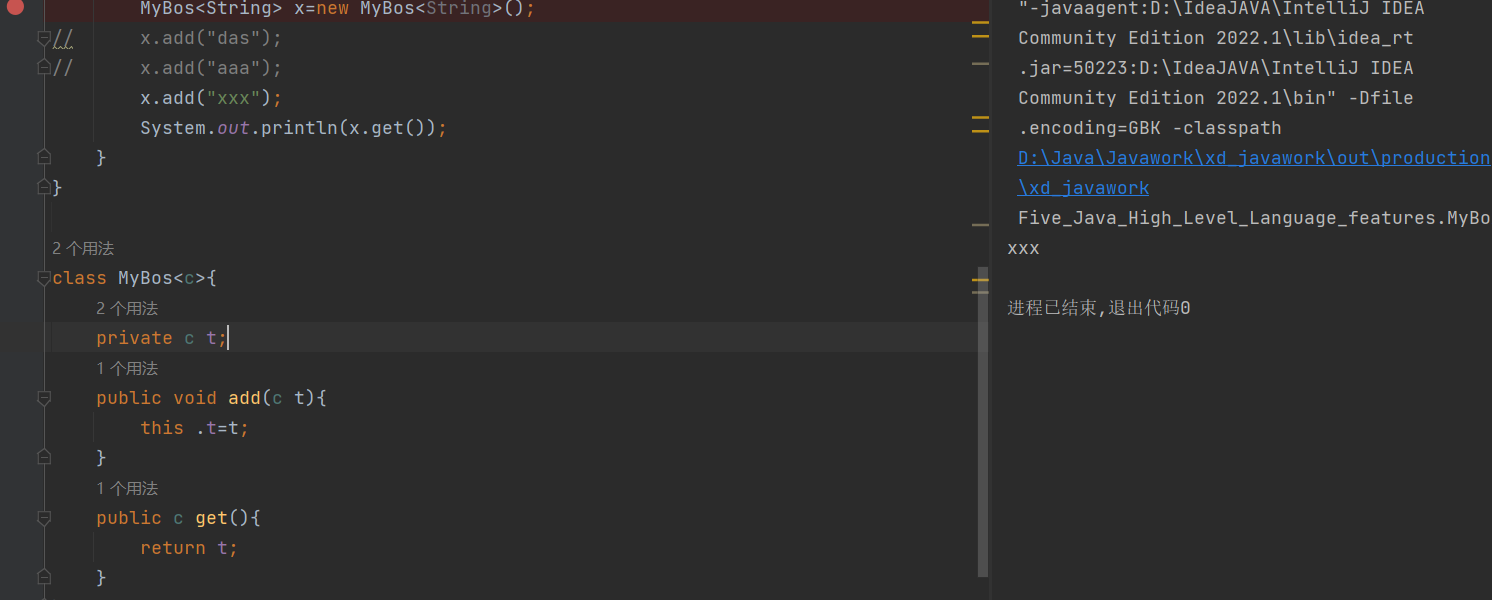
1. 在接口中的成员不能使用private、protected。



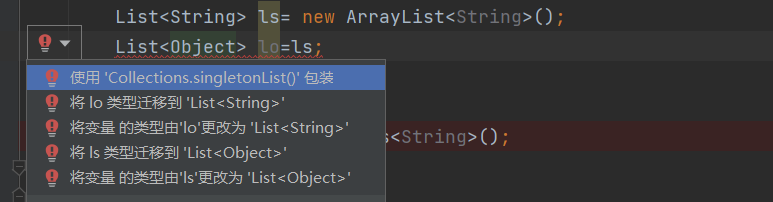
1. 接口在定义的时间，方法只写出方法头 就可以并且以**“ ；”**结尾。

## P55

1. 在泛型定义的时间，泛型中的类型变量是不存在的数值，即可以随便定义；但是最好使用已有的命名习惯。E-元素、K-键值、N-数字、T-类型、V-值、S、U、V代表二三四个类型参数。



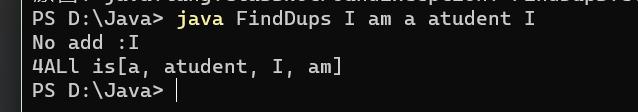
1. 在泛型定义的时间并没有所谓的父子类的继承关系，即使类型参数是有父子类关系的，对应的泛型也是不具备父子两类关系的。



1. 如果需要使用泛型的父子类关系，可以使用通配符***‘ ？’*** 来解决。例如：Animal是lion和butterfly的父类，就可以使用Link<? Extends Animal>来接收lion和butterfly类型的泛型变量。此时Animal泛型认定为lion和butterfly泛型的父类。
2. 泛化方法的定义在定义方法的时间增加了类型变量的声明，在修饰符和返回类型之间增加带‘<>’的类型参数的声明。可以对所有的方法进行泛化，需要注意的是，泛化方法传递的参数是和普通方法一样的，即传递的是实参。并不是和泛型调用时传递的实参是一个具体的类型。

## P64

1. 在集合HashSet中添加的元素是无序的，没有固定的顺序，并且不会添加相同的元素。



## P70

1. 在ArrayList中的相应操作，添加元素、插入元素、修改指定位置的元素、删除指定位置的元素、截取相应的元素、添加一个集合中的所有元素。
2. 在添加元素的时间使用到List.Add()如果没有指定元素的位置，默认会在末尾去添加，如果指定了位置的时间，会进行插入操作，将原本位置的元素整体向后移动



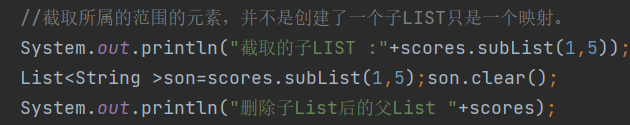
1. 修改指定位置的元素：List.set(index,element) 将指定位置的元素重新赋值

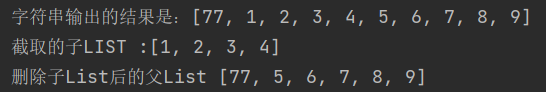


1. 删除指定位置的元素。Remove（index）。

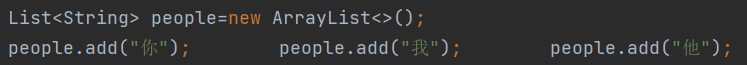


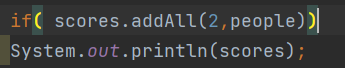
1. 截取相应的子List，注意到只是当前List视图，改变子List的值会反映到当前的List，相当于是同步的。





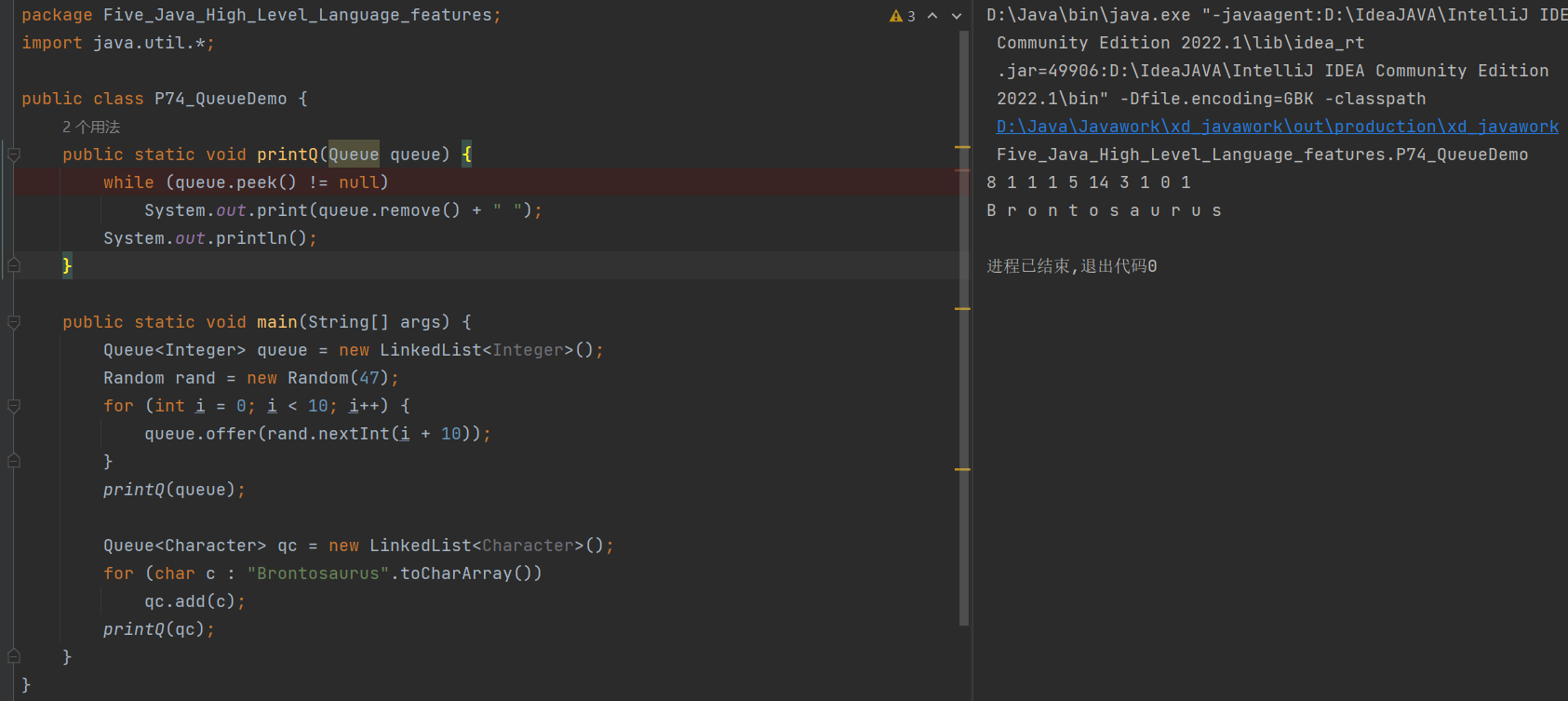
1. 将其他集合的值全部添加到当前的List, 成功返回true。





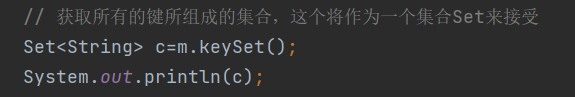
## P74

1. 队列的操作可以分为两种，一种是在失败的时间抛出异常，一种是在失败的时间返回NULL或者FALSE值。
2. 在调用offer（）插入对象的时间，对象在队列中排序：Integer等采用的是莫仍方法的排序



## P79

1. 在Map中是通键和值之间的对应关系来存储的数据的，HashMap中存储的数据是允许数据为空。
2. 获取Map中的所有键值将其作为一个Set类的集合将其输出。Map.KeySet()



1. 获取Map中所有键所对应的数值Map.Values（）,其类型类Collection



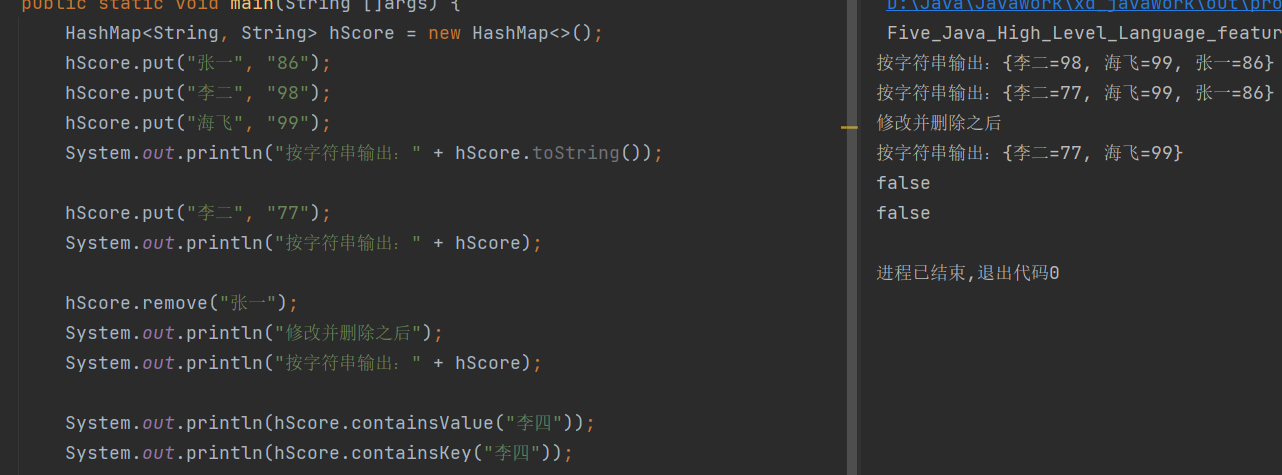
1. 获取键值对





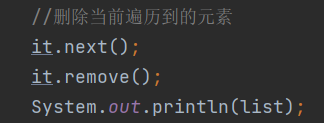
## P81

1. 对Map集合的基本操作，键值对的添加、通过建来删除对应的值，通过键来删除对应的值，判断知否有某个键或者某个值。

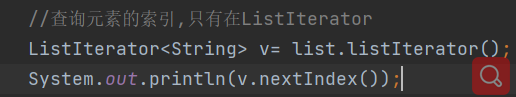


## P83

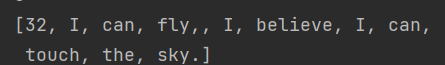
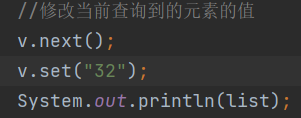
1. 在iterator中用来遍历集合中的所有元素，在Set对象中的遍历中，元素的遍历次序是不确定的；对于List对象中，对象的遍历是从前往后的，并且List还支持ListItertor,该接口支持元素的反响遍历。
2. Removew方法移除的是当前遍历到的元素，即删除最近一次的Next和prevoius（）调用返回的数值



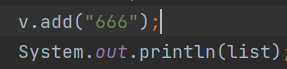
1. 在其子类查询下一个元素的索引NextIndex（）



1. 子类ListIterator中改变当前元素的值set（）



1. 子类ListIterator中在下一个将要取得的元素之前插入一个新的元素，add（）

## P94

1. Enum枚举类型中的常量实际上是一个枚举类型的实例化的对象。即在创建枚举常量的时间使用默认构造方法的时间可以省略（）。
2. 枚举类型实际上是一个 具有固定实例的一个特殊类，枚举常量是具有static、final属性的。

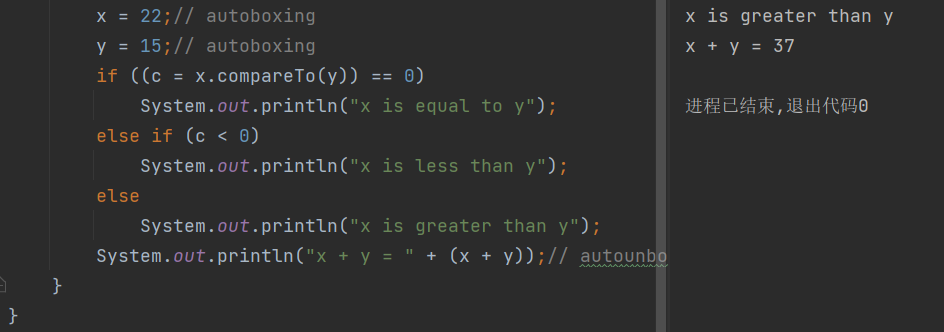
1. 枚举体的定义是和普通的类是一样的，并且枚举类型的构造方法的权限默认为private并且只能设置为private，保证不会有其他程序调用枚举类型构造方法来创建新的实例。

1. 枚举类型的equals（）方法，比较的是指向的对象是否等于次枚举常量

## P99

在Wapper中变量x、y赋予了整型值，但是由于他们是Integer类型的，编译器自动将他们封装为Interger对象，使得x.compareTo可以运行，但是在执行X+Yde 的时间，编译器又自动的将X和Y拆箱，这样两者才可以进行加法运算。



## Libiary

1. Book类用于存储一本书的属性：书名、作者、出版社、是否在馆的信息。他所能够提供的信息外界返回他的基本信息
2. Books类作为使用Book的一个集合类，可以用来存储一种书籍的不同本，具体是否在馆的状态工具*Inlibaray* 来判断，他向外界提供增加书，减少书籍，返回书籍的各种信息的功能。借阅功能来改变状态。
3. Libiary类向外界提供各种功能。

