**4.1**

Exercise 4.1.1. Suppose we want to write a class WordUtilsthat includes functions we can run on lists of words, including a method that calculates the longest string in an SLList. Try writing this method by yourself. The method should take in an SLList of strings and return the longest string in the list.

public static String longest(SLList list) {

        int maxDex = 0;

        for (int i = 0; i < list.size(); i += 1) {

            String longestString = list.get(maxDex);

            String thisString = list.get(i);

            if (thisString.length() > longestString.length()) {

                maxDex = i;

            }

        }

        return list.get(maxDex);

}

**Hypernyms, Hyponyms, and Interface Inheritance**

上义词，下义词，接口继承

public class AList<Item> implements List61B<Item> {

// Implements 是关键字

}

implements List61B<Item> 实际上是一种承诺。AList 类声明：“我承诺我将拥有并定义 List61B 接口中指定的所有属性和行为。”

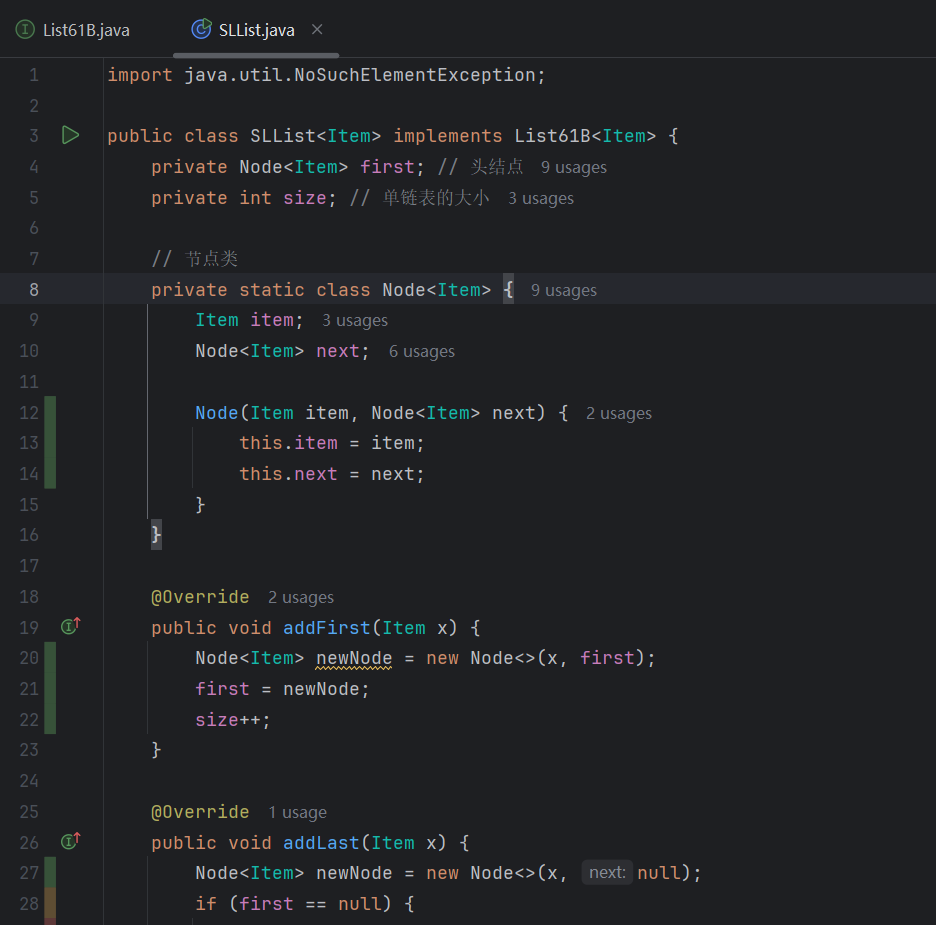
然后：

@Override要写在方法的上面

也就是说，我在父类里有个函数，叫sizeOf，然后我在子类里，我写了@Override，我仍然用sizeOf的函数名，但是我可以更改函数的内容  
**方法名：** 子类方法的方法名必须与父类方法相同，即 sizeOf。  
**参数列表：** 子类方法的参数列表必须与父类方法相同。  
**返回类型：** 子类方法的返回类型必须与父类方法相同，或者是其子类型（对于协变返回类型的情况）

实例已经放在Week4文件夹，父类和子类需要在同一目录、两个文件；@Override不需要引用任何包。





List61B<String> lst = new SLList<String>()

**静态类型**：指的是变量声明时的类型。例如，List61B<String> lst 中的 lst 的静态类型是 List61B<String>。

**动态类型**：指的是变量实际引用的对象的类型。例如，SLList<String> SP = new SLList<String>(); 中的 SP 的动态类型是 SLList<String>，因为它实际上指向了一个 SLList 的实例。

在非重载的情况下，List61B<String> lst = new SLList<String>(); 中的 lst 在编译时静态类型是 List61B<String>，但在运行时动态类型是 SLList<String>。这意味着，尽管你使用了 List61B 类型声明了 lst，但实际上 lst 指向的对象是 SLList 类型的。因此，你可以调用 lst 引用对象上 List61B 接口定义的方法，但实际运行时会调用 SLList 中的具体实现。

方法重载和动态分派：  
方法重载是指在同一个类中可以有多个方法，它们具有相同的名称但是参数类型或数量不同。Java 在调用重载方法时是根据静态类型来决定调用哪个方法的，这被称为静态分派。例如，考虑下面的两个方法：

public static void peek(List61B<String> list) {

System.out.println(list.getLast());

}

public static void peek(SLList<String> list) {

System.out.println(list.getFirst());

}

在调用 peek(SP); 和 peek(LP); 时，编译器会根据静态类型来选择调用哪个方法。因为 SP 的静态类型是 SLList<String>，所以会调用接受 SLList<String> 参数的 peek 方法。而 LP 的静态类型是 List61B<String>，会调用接受 List61B<String> 参数的 peek 方法。

**4.2**

**Extends**

**继承的定义**：你可以使用 extends 关键字来设置类之间的继承关系。例如，对于 RotatingSLList 类继承自 SLList 类，你的类头部会像这样：

public class RotatingSLList<Item> extends SLList<Item>{

}

这里，RotatingSLList<Item> 表示 RotatingSLList 是 SLList 的子类，并且拥有 SLList 类的所有属性和方法，同时可以在其基础上进行修改和扩展。

通过继承，你可以保留 SLList 的所有功能，包括 addFirst、size 等方法，并且可以在 RotatingSLList 中添加新的方法或修改现有方法来满足新的需求

例：

public void rotateRight() {

Item x = removeLast();

addFirst(x);

}

使用 extends 关键字，子类会继承父类的所有成员。"成员" 包括：

所有实例变量和静态变量  
所有方法  
所有嵌套类  
\*需要注意的是，构造函数不会被继承，而且私有成员不能被子类直接访问。