作业问题收集：

HW2:

1.在Java中，一个.java文件可以包含多个类定义，但有一些规则需要遵守：

**公共类的规则**：每个.java文件最多只能有一个公共类（public class）。此外，文件的名称必须与公共类的名称相匹配。例如，如果你有一个名为PublicClass.java的文件，那么它必须包含一个名为PublicClass的公共类。

**非公共类**：你可以在同一个文件中拥有任意数量的非公共类（没有public修饰符的类）。这些类可以被文件中的其他类访问，但它们对于文件外的其他类是不可见的。

**包的作用域**：即使是非公共类，它们也是属于声明文件顶部指定的包的。如果没有指定包，则它们属于默认包。

**访问修饰符**：由于非公共类不是公开的，因此它们在定义时不使用public修饰符。它们可以被同一包内的其他类访问。



**2.printIn和print**

println方法会自动将参数转换为字符串并进行打印。如果想要输出多个参数，可以使用加号"+"来拼接多个参数。

　　print和printf,它们的使用方式与println类似，但不会自动换行。如果不希望输出结果换行，可以使用print方法。

总之，println是Java中常用的输出语句，可以用于输出字符串、变量、数字等内容，并且会自动在结尾换行。

3. `Scanner` 类是 Java 的 `java.util` 包中的一部分，它用来解析原始类型和字符串的简单文本扫描器。`nextLine()` 和 `next()` 是 `Scanner` 类中用于读取输入的两个常用方法。

nextLine() 方法：

- 使用方法：`nextLine()` 方法读取输入上的下一行。

- 返回值：它返回从当前位置到行尾的字符串，包括空白符，但不包括行尾符。

- 注意事项：

- 在读取之前输入的 `nextInt()`, `nextDouble()` 等其他 `nextXXX()` 方法后使用 `nextLine()` 时，通常需要再调用一次 `nextLine()` 来消耗掉该行剩余的行尾符。

- 如果不先消耗行尾符，`nextLine()` 会认为它已经读到了一个整行，并返回一个空字符串。

### next() 方法：

- \*\*使用方法\*\*：`next()` 方法读取输入上的下一个单词。

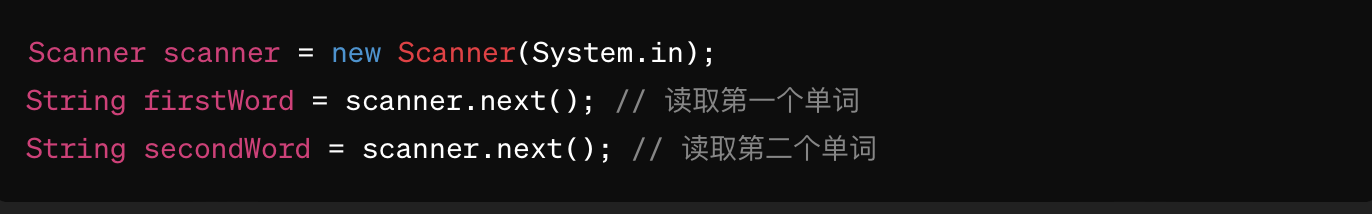
- \*\*返回值\*\*：它返回当前位置到下一个有效字符之前的字符串，它不包含任何空白符。

- \*\*注意事项\*\*：

- `next()` 会自动读取并丢弃任何前导空白字符（空格，换行等）。

- 它在遇到下一个空白符时停止读取。

- `next()` 不会读取两个单词之间的空格或者换行符等分隔符。



### 总结：

- 当你需要读取整行输入时，使用 `nextLine()`。

- 当你只需要读取输入中的下一个单词时，使用 `next()`。

- 在混合使用 `nextXXX()` 和 `nextLine()` 时，特别注意 `nextLine()` 的使用，它可能会立即返回，因为 `nextXXX()` 方法通常不会读取行尾的换行符。

记得始终关闭 `Scanner` 对象以释放资源，尤其是在输入结束时或者不再需要扫描器时：scanner.close();

使用 `Scanner` 类时，应该注意异常处理。通常，输入操作可能会导致 `InputMismatchException`，如果输入的数据类型与期望的类型不匹配，那么程序可能会抛出这个异常。因此，如果你的程序在对输入进行假设时，应该捕获这些异常或提前检查输入的有效性。

4. 对于代码：scanner.close();一般会放在Main函数的什么位置？

在Java中，调用 scanner.close(); 通常是为了关闭 Scanner 对象并释放它所占用的资源。选择在何处关闭 Scanner 取决于你不再需要使用 Scanner 对象进行输入操作的时间点。这里是一些常见的情况：

在程序末尾：如果 Scanner 只在 main 函数中使用，并且在程序的最后不再需要它时，你应该在 main 函数的最后关闭它，就在所有输入操作完成之后。



在输入操作之后：如果你在 main 方法中的一个特定点之后不再需要输入，可以在那个点关闭 Scanner。



**注意**：如果 Scanner 是使用 System.in 创建的，通常在程序结束前关闭它。但是，如果你在一个大型应用程序中，特别是那些有多个输入源的应用程序中关闭了 System.in，那么再次尝试从 System.in 读取输入可能会导致 “NoSuchElementException” 或 “ILLegalStateException”。在这种情况下，确保在应用程序生命周期的适当时间点关闭 Scanner。

HW4:

Java 类“Vector”提供了在数值上下文中执行基本向量运算的功能，例如加法、减法、标量乘法和范数计算。 下面是对每一行代码的解释：

### 进口

``java

导入java.util.Scanner；

````

- \*\*导入语句\*\*：从`java.util`包中导入`Scanner`类，该类用于从标准输入流（键盘）读取输入。

### 类声明

``java

公共类向量{

````

- \*\*类声明\*\*：声明一个名为“Vector”的公共类。

### 字段声明

``java

私有最终 int[] 元素；

````

- \*\*字段声明\*\*：声明一个私有最终整数数组“elements”，用于存储向量的组成部分。 “final”关键字意味着对数组的引用在初始化后就不能更改。

### 构造函数

``java

公共向量（int []元素）{

this.elements = 元素；

}

````

- \*\*构造函数\*\*：使用提供的整数数组初始化“Vector”的新实例。 传递给构造函数的数组成为向量分量的内部表示。

###向量相加法

``java

公共静态向量加（向量v1，向量v2）{

````

- \*\*方法声明\*\*：采用两个向量并返回它们的和的静态方法。

``java

if (v1.elements.length != v2.elements.length) {

throw new IllegalArgumentException("向量具有不同的维度。");

}

````

- \*\*维度检查\*\*：确保两个向量具有相同的维度。 如果不是，它会抛出一个“IllegalArgumentException”。

``java

int[] 结果 = new int[v1.elements.length];

for (int i = 0; i < 结果.length; i++) {

结果[i] = v1.elements[i] + v2.elements[i];

}

返回新向量（结果）；

}

````

- \*\*加法逻辑\*\*：计算两个向量的分量总和，并返回一个新向量和结果。

###向量减法

``java

公共静态向量减法（向量v1，向量v2）{

````

- 与加法类似，但用于减法。

``java

if (v1.elements.length != v2.elements.length) {

throw new IllegalArgumentException("向量具有不同的维度。");

}

````

- 确保两个向量具有相同的维度。

``java

int[] 结果 = new int[v1.elements.length];

for (int i = 0; i < 结果.length; i++) {

结果[i] = v1.elements[i] - v2.elements[i];

}

返回新向量（结果）；

}

````

- 计算分量差异并返回结果的新向量。

### 标量乘法

``java

公共向量乘法（int a）{

````

- 将向量的每个分量乘以标量“a”。

``java

int[] 结果 = new int[elements.length];

for (int i = 0; i < elements.length; i++) {

结果[i] = 元素[i] \* a;

}

返回新向量（结果）；

}

````

- 计算乘积并返回一个新向量。

### 范数计算方法

``java

公共双范数() {

````

- 计算无穷范数（分量之间的最大绝对值）。

``java

整数最大值=0；

for (int 元素: 元素) {

max = Math.max(max, Math.abs(元素));

}

返回最大值；

}

````

- 找到最大绝对值。

``java

公共双范数(int p) {

````

- 计算“p”范数，即向量范数的一般形式。

``java

如果（p <= 0）{

返回范数(); // 默认为无穷大范数

}

双倍总和 = 0；

for (int 元素: 元素) {

sum += Math.pow(Math.abs(元素), p);

}

返回 Math.pow(sum, 1.0 / p);

}

````

- 计算元素绝对值的“p”次方之和，然后求和的“p”次方根。

### 点积法

``java

公共静态 int 点（矢量 v1，矢量 v2）{

````

- 计算两个向量的点积。

``java

if (v1.elements.length != v2.elements.length) {

throw new IllegalArgumentException("向量具有不同的维度。");

}

整数结果 = 0；

for (int i = 0; i < v1.elements.length; i++) {

结果 += v1.elements[i] \* v2.elements[i];

}

返回结果；

}

````

- 将相应元素相乘并对结果求和。

### 角度计算方法

``java

公共静态双角（矢量v1，矢量v2）{

````

- 使用点积计算两个向量之间的角度