CS109-计算机程序设计基础笔记

By 树礼2023级 阿黄

这是**2024年春季学期**本人上CS109计算机程序设计基础（也就是大家会习惯性称为JavaA的笔记）的笔记，最后的总评是A，**受了计系大二学长hlh的很多帮助，十分感谢他。**

于是我也分享出笔记，希望能帮到大家，也祝大家能取得满意的成绩~

要是有需要交流/勘误等，欢迎+QQ: 2672859764

一点小小分享：感觉JavaA的期末考试占比最高，但是又考很多的上课提到的细节，大家要好好听课！Proj很难做，但是分数占比没有特别高，对于编程基础一般的同学（比如我这种）可以优先保证基础分，再锦上添花！最后想说的是**重要的是学到知识、培养能力**，绩点多少努力争取，但不尽人意也不用挂在心上，满怀期待冲冲冲！！！

Content (后续可以从Part1开始食用)

0. **Mine：**是一些IDE对于初学者来说的使用小技巧、快捷键等，几乎能覆盖学编程前期所需要的！还有一些好用的库和函数，有时间可以看看~
1. **Parts0:** 是一些其他在学习过程中遇到的感觉有点旁逸斜出的、但是需要了解的一些知识！
2. **Parts1:** 是一些编程基本知识，也是本门课程最开始会要学的内容（每个语言都大差不差，但是仍然有许多细微之处要注意辨别），循环输入输出等
3. **Part2:** 是一些面向对象编程的知识，这是Java很有特点的地方，也是难点，但是大家放轻松学

建议先看Part1，是正式开始的内容！！！！

Mine

IDE快捷输入

psvm ->

```
public static void main(String[] args) {

}
```

sout ->

```
System.out.println();
```

souf ->

```
System.out.printf("");
```

fori ->

```
for (int i = 0; i < ; i++) {

            }
```

IDE快捷键

`Ctrl` + `Alt` + `L` 调整格式

`Ctrl` + `D` 复制该行

`Ctrl` + `Shift` + `F10` 编译并运行

`Ctrl` + `Shift` + `Enter` 补全代码括号结构

`Alt` + `Insert` 类中生成

`Alt` + `Enter` 导入头文件/生成函数（即补全缺失的）

`Ctrl` + `Shift` + `<-` backspace跳回前一个调用的地方

`Ctrl` + `W` 扩展选区

库和函数

1.Math类

函数	含义
Math.min(a,b)	
Math.max(a,b)	
Math.sqrt(a)	开方

2.Arrays类

方法	含义
Arrays.toString(hlh)	以字符串形式打印数组(只能一维数组)
Arrays.sort(hlh)	排序，可以对整型数组、字符数组排序；可以对部分位置排序，字符数组可以逆序，字典序，ASCII序
Arrays.equals(lhh,hlh)	比较数组元素是否相等
Arrays.binarySearch(hlh, val)	二分查找特定元素，可以指定区间，没找到会返回负值，否则返回index
Arrays.copeOf()/Arrays.copeOfRange()	截取数组
Arrays.fill(hlh,1)	填充数组
Arrays.deepToString(a)	以字符串形式打印数组(只能大于一维的数组)

补丁

1.等价计数器

```
for(int i=1;i<=3;i++)
```

```
int t=3;
while(t-->0)
```

2.

OJ的输入输出流分开

3.需要输入的函数记得传入Scanner对象

4.nextLine读入会读入换行符前面的，尽量在前面先用一个nextLine换行

5.块作用域 Block Scope：注意变量的定义领域

6.`String.format("%d",n)`返回一段格式化的字符串

7.复印ArrayList：

```
ArrayList<Integer> c2 = new ArrayList<Integer>(c1);
ArrayList<Integer> c2 = (ArrayList<Integer>)c1.clone();
```

8.用arraycopy函数可以进行一维数组的快速赋值：
`arraycopy(model,start1,panel,start2,length)`

Parts0

运行时间的确定：

```
long current1=System.currentTimeMillis();
/* your algorithm */
long current2=System.currentTimeMillis();
System.out.printf("your program using %.3f second",(current2-current1)/1000.0d);
```

生成随机数

调包：

```
import java.util.Scanner;
```

定义对象：

```
Random r = new Random();
```

输入：

```
n = r.nextInt(10);//随机生成0-9整型
n = r.nextInt(10)+1;//随机生成1-10整型
```

包装类Wrapper Classes：

- **.valueOf()**：将基本类型或字符串转换为包装类对象。
- **.parseXXX()**：例如`Integer.parseInt("123")`，将字符串转换为基本数据类型。
- **.toString()**：返回表示对象值的字符串。
- **.compareTo()**：比较两个包装类对象。

简单来说，我的第一层理解：其实就是方便了基本数据类型和字符串之间的转换。

示例

自动装箱：

```
Integer myInt = 5;  // 自动装箱，将int转换为Integer
```

自动拆箱：

```
int myPrimitiveInt = myInt;  // 自动拆箱，将Integer转换为int
```

使用`.valueOf()`和`.parseXXX()`方法：

```
Integer myInteger = Integer.valueOf("123");  // 将字符串转换为Integer对象
int myInt = Integer.parseInt("123");         // 将字符串转换为int基本类型
```

设置主程序参数：

IDEA：

在 IntelliJ IDEA 中设置主方法参数

1. 打开您的项目，并找到您想要运行的主类。
2. 点击菜单栏中的 `Run` -> `Edit Configurations...`。
3. 在打开的 `Run/Debug Configurations` 对话框中，找到您的应用程序，并点击它。
4. 在右侧，找到 `Program arguments` 字段。
5. 在 `Program arguments` 字段中输入您想要传递给主方法的参数，参数之间用空格分开。
6. 点击 `OK` 保存您的配置。
7. 现在，您可以使用这些参数来运行您的应用程序了。

命令行：

在命令行中设置主方法参数

在命令行中运行 Java 应用程序并传递参数相对简单。您需要编译您的 Java 源文件（如果还没有编译的话），然后使用 `java` 命令运行编译后的类，并在命令后面附加您的参数。以下是基本步骤：

1. 打开命令行工具。
2. 更改目录（使用 `cd` 命令）到包含您的 `.java` 文件的目录。
3. 使用 `javac` 命令编译您的 Java 文件。例如，如果您的文件名为 `Main.java`，则命令为：

```css
javac Main.java
```

这将生成一个 `Main.class` 文件。

4. 使用 `java` 命令运行您的程序，并传递参数。例如：

```css
java Main arg1 arg2 arg3
```

在这里，`arg1`，`arg2`，`arg3` 是传递给 `Main` 类 `main` 方法的参数。

请注意，您在命令行中传递的参数会作为字符串数组传递给 `main` 方法。您可以在程序内部通过索引访问这些参数。

Junit

JUnit 是一个流行的 Java 测试框架，用于编写和运行可重复的测试。它是一个开源框架，专门用于测试小块代码，通常称为单元测试。JUnit 提供的工具和注解帮助开发人员快速编写测试，并确保代码按预期工作。以下是 JUnit 的一些关键特点和组件：

1. **测试用例（Test Cases）**：测试用例是一组包含测试方法的类，用于测试特定的程序行为。在 JUnit 中，测试用例通常是用 `@Test` 注解的方法。

2. **断言（Assertions）**：断言是用于验证测试中的条件的方法。如果条件为真，则测试通过；如果条件为假，则测试失败。例如，`assertEquals()` 用于检查两个值是否相等。
3. **测试运行器（Test Runners）**：这是 JUnit 的一部分，负责测试执行的流程。测试运行器决定哪些测试会运行以及它们的运行顺序。
4. **测试装置（Test Fixtures）**：测试装置是指为执行测试所需的固定的环境或状态。例如，你可能需要在每个测试方法运行之前或之后，创建特定的对象或设置特定的数据。JUnit 提供了 `@Before`、`@After`、`@BeforeClass` 和 `@AfterClass` 注解来帮助设置和清理测试环境。
5. **注解（Annotations）**：JUnit 使用 Java 注解来提供元数据。这些注解如 `@Test`（声明方法为测试方法）、`@Ignore`（忽略某个测试方法）、`@Before` 和 `@After`（分别在每个测试方法前后执行）等，都是简化测试代码的有效工具。
6. **测试套件（Test Suites）**：测试套件允许你将多个测试类组合在一起并一起执行。通过使用 `@RunWith` 和 `@Suite` 注解，你可以创建一个测试套件，其中包括一组需要同时运行的测试类。

JUnit 提供了一个清晰的框架来帮助开发者编写和维护他们的单元测试，这在持续集成和测试驱动开发的过程中尤为重要。使用 JUnit 可以帮助确保你的 Java 应用程序的质量和稳定性，同时也使得回归测试变得容易和高效。

StdDraw画图

Color类：

```
Color.GRAY//灰色
Color color = Color.GRAY;
```

StdDraw类：静态方法

方法	含义
StdDraw.setPenColor(color.getColor());	
StdDraw.filledCircle(x, y, radius);	
StdDraw.filledRectangle(x, y, this.width / 2, this.height / 2);	

Parts1

示例代码

```
public class Lhh {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

输入与输出

输入 Scanner

调包：

```
import java.util.Scanner;
```

定义对象：

```
Scanner input = new Scanner(System.in);
input.close();//关闭输入流，如果没有关闭，会随着程序关闭而关闭
```

输入：

```
a = input.next();//读入字符串
b = input.nextInt();//读入并转化为整型
c = input.nextDouble();//读入并转化成浮点型
d = input.next().charAt(0);//读入字符串的第一个字符
```

输出：System.out Object

可以用sout快速输出

三个方法：

Println

print

printf

```
printf("%d + %d = %d", 1, 1 , 2);
```

格式化输出：

Types	占位符
byte, short, int, long	%d
float, double	%f
char	%c
String	%s
boolean	%b

占位符	含义
%.1f	保留一位小数(四舍五入)
%-20f	左对齐20字符
%20f	右对齐20字符

占位符	含义
%-20.1f	左对齐20字符并保留一位小数

初始数据类型 Primitive Data Types

整型：

Type	Size	Range
byte	8 bits	-128 (10000000*) to +127 (01111111*)
short	16 bits	-32,768 to +32,767
int	32 bits	(about) -2.147 billion to +2.147 billion
long	64 bits	(about) -10E18 to +10E18 (-9.22 quadrillions to +9.22 quadrillions)

注：如果long定义的时候，赋的常数值大于了int类型，则必须加L，其他情况可加可不加。

浮点型：

Type	Size	Range
float (单精度)	32 bits	-3.4E+38 to +3.4E+38
double (双精度)	64 bits	-1.7E+308 to 1.7E+308

注意：（float要以f结尾，不然默认为double）

```
double d = 3.14;
float f = 3.14f;
```

布尔型：

关键词 true 和 false

```
boolean isChecked = false
```

字符型：

单一的16-bit Unicode character，可以储存几乎任意字符，

```
char c = 'a'

char c = '\u5927'
```

取值范围：

```
public class PrimitiveTypeTest {
    public static void main(String[] args) {
        System.out.println(Integer.SIZE);//二进制位数
        System.out.println(Integer.MAX_VALUE);//上界
        System.out.println(Integer.MIN_VALUE);//下界
    }
}
```

Byte	Short	Integer	Long	Float	Double

变量命名

Java 变量命名规则 | 菜鸟教程 (runoob.com)

数据类型转换

Type Cast:

```
int a,b;
average = (double) a / b;
```

注意：强制数据类型转换只会创造一个a的浮点型临时副本，不改变a本身的数据类型

注意：强制数据类型转换的优先级高于二元运算符"/"，所以实际上是double除以一个int

Type Promotion:

在上述实例中，b会被临时提升成double，使结果也为double，所以b本身的数据类型不会发生改变

Type	Valid promotions
double	None
float	double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double (but not char)
byte	short, int, long, float or double (but not char)
boolean	None (boolean values are not considered to be numbers in Java)

只能低级向高级自动promote，高级向低级可能会有数据损失，只能强制转换。

算数运算 Arithmetic Computation

常规运算符：

运算符	含义
+	
-	
*	
/	整除（与C++一样，取决于参与运算的数据类型）
%	求余
++	
--	
==,!=	
>,<,>=,<=	
&&	
||	
!	
+=,-=,*=,%=,/=	
?:	三目运算符

注意：逻辑与的优先级高于逻辑或，当优先级一样时，从左到右执行

当从左执行时，逻辑结果已经确定，则右侧的不会执行了（短路了），右侧的赋值语句也不会执行

可以用|和&来代替||和&&执行，则逻辑两侧的语句都会被检测

^也可以是异或

（理解方式*：相同0，不同1）

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

位运算符：

位运算符	含义
&	按位与
|	按位或
^	按位异或（相同0，不同1）
~	按位非（一元）
<<	按位左移，低位补0
>>	按位右移，正数高位补0，负数高位补0
<<</>>>	去掉符号左移/右移

下表中具有最高优先级的运算符在的表的最上面，最低优先级的在表的底部。

类别	操作符	关联性
后缀	() [] . (点操作符)	左到右
一元	expr++ expr--	从左到右
一元	++expr --expr + - ~ ！	从右到左
乘性	* /%	左到右
加性	+ -	左到右
移位	>> >>> <<	左到右
关系	> >= < <=	左到右
相等	== !=	左到右
按位与	&	左到右
按位异或	^	左到右
按位或	|	左到右
逻辑与	&&	左到右
逻辑或	||	左到右
条件	？：	从右到左
赋值	= + = - = * = / =%= >> = << = &= ^ = | =	从右到左
逗号	,	左到右

流程控制

循环结构

for

注：据说++i会相对i++优化一些，因为不用临时内存空间，而二者表型一样

执行顺序：

1. 初始化+判断条件语句
2. 执行循环体
3. 结束后，对变量进行操作，操作后，进行条件语句判断

while

do-while

注：一定要注意最后的分号！！！！

for(int x : 数组名)

continue

break

条件语句

if-else

switch-case

注：先进行匹配，如果有匹配则转入对应的case，如果没有匹配，则转入default。一旦进入case/default，则会一直进行后续语句，直至遇到break。

只能是整数类型、字符，**绝对不能是实数。**

```
import java.util.Scanner;

public class SwitchCaseTest {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        char grade = in.next().charAt(0);

        switch (grade) {
            case 'A':
                System.out.println("优秀");
                break;
            case 'B':
            case 'C':
                System.out.println("良好");
                break;
            case 'D':
                System.out.println("及格");
                break;
            case 'F':
                System.out.println("你需要再努力努力");
                //break;
            default:
                System.out.println("未知等级");
            case 'G':
                System.out.println("你有点差劲了");
        }
        System.out.println("你的等级是 " + grade);

    }
}
```

数组 Array

声明与创建

```
int[] c = new int[12];
```

等价于

```
int[] c;//声明一个int类型数组变量
c = new int[12];//创建一个长度为12的数组空间，并赋值给数组变量
```

初始值：数组创建时候有初始值，整型数组为0，字符型为\u0000，布尔型为false。

初始化：

```
int[] c = new int[]{1,5,6,8,9};
int[] c = {1,5,6,8,9};
```

`c[0]`表示找到c对应的数组空间，利用index0来找到第一个储存空间的值。

数组的方法与函数：

方法或者函数	作用
c.length	返回数组的长度
c.clone()	复印件，地址不同（直接赋值则地址相同）
c.equals(hlh)	比较地址是否一样

缩写	作用
c.iter()/c.for()	for (int i : c)
c.fori()	for (int i = 0; i < c.length; i++)
c.sout()	System.out.println(c);
c.nn()/c.notnull()/c.null()	

注：若直接输出数组名不能直接输出整个数组，而会输出它代表的地址。**字符数组例外。**

```
char[] Array4 = {'a', 'b', 'c'};
System.out.println(Array4);
//运行结果：abc
```

增强循环Enhanced For:

```
for (int e : array2){
    //每一轮循环，将array2中的一个元素赋值给e，所以改变e不会改变array2中的元素
}
```

空数组的建立与地址调用：

```
int[] b = null;
b = c;//让b和c共享地址
```

二维数组：

每个元素都是一维数组的地址

注：二维数组定义时（）一定要定义行数，但不一定需要列数。

数组ragged arrays：

长短不一的一维数组的数组，超出初始化的值会被视为超过index）

二维数组示例程序：

```
import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        int[][] arr = new int[3][];
        arr[0] = new int[]{1, 2, 3};
        arr[1] = arr[0];
        arr[2] = new int[]{3, 4, 5, 6};
        System.out.println(Arrays.toString(arr[1]));
        System.out.println(Arrays.toString(arr));
        System.out.println(Arrays.deepToString(arr));
        for (int e : arr[1]) {
            System.out.println(e);
            e = 0;
            System.out.println(e);
        }
        for (int[] a : arr) {
            System.out.println(a);
            a = new int[2];
        }
        System.out.println(Arrays.deepToString(arr));
        for (int[] a : arr) {
            for (int i = 0; i < a.length; i++) {
                a[i]++;
            }
        }
        System.out.println(Arrays.deepToString(arr));
        }
}
```

矩阵输入：

```
int[][] a = inputMatrix(in);
```

字符与字符串

字符

数据类型转换

java中unicode是包含了Ascii码表的。

所以把字符强制数据类型转换为int，会变成ASCII码值（十进制）。（c++同理）

若要把Unicode变成字符，则是 `\u + a` a是**十六进制码位**

String类

String是reference type，指向一段储存了字符串的内存

String具有不可变性，任何的modification都会创建一个新的String对象。

实例化/创建

```
String s1 = new String("Hello world!");
String s2 = new String();//空字符串
String s3 = new String(s1);//会建立一个新的
String s4 = new String(charArray);
String s5 = new String(charArray,3/*出发点offset*/,2/*数几个count*/);
```

也可以用字面常量(string literals)创建

```
String s1="Hello world!";
String s2="Hello world!";//这里与语句String s2=s1;等价
```

```
String s1 = "Java";
String s2 = "Java";
String s3 = new String("Java");
String s4 = new String("Java");
System.out.println(s1 == s2); // true
System.out.println(s3 == s4); // false
```

函数

函数	含义
s.length()	返回字符串长度
s.split(String regex)	用regex分割字符串，返回字符串数组
s.charAt()	返回对应index处的值
s.getChars(int start, int end, char[] dst, int start_in_dst)	把String类型中从start到end的
s.equals(s2)	比较内容是否相等
s.indexOf(s2`,fromindex=0`)	
s.isEmpty()	
s.toCharArray	
s.trim()	
s.subSequence(int start, int end)	
s.startsWith(String prefix)	是否以给定前缀开始

- `length()`：返回字符串的长度。
- `charAt(int index)`：返回指定位置的字符。
- `substring(int beginIndex, int endIndex)`：返回字符串的一个子字符串。
- `equals(Object anObject)`：比较字符串的内容是否相等。
- `indexOf(String str)`：返回指定子字符串在此字符串中第一次出现处的索引。
- `toLowerCase()` 和 `toUpperCase()`：将字符串转换为全部小写或大写。
- `concat(String str)`：将指定字符串连接到此字符串的结尾。
- `replace(CharSequence target, CharSequence replacement)`：替换字符串中的某部分。

操作

字符串拼接

用 `+` 可以拼接两个string或者string和char

缺点：效率很低，空间占用率较大

字符串相等比较

对于reference type，如果使用 `==` 来比较两个字符串，则当且仅当指向的内存块相同的时候，才能相同。

如果使用 `s.equals(ss)` 则是比较内容是否匹配

字符串大小比较

从index从小到大，挨个比较ASCII码值/unicode编码

由于空字符是 `\u0000` 所以它比其他的字符都要小

StringBuilder类：

创建：

```
StringBuilder sb = new StringBuilder();//初始长度为16字符
```

函数：

- `append(xxx)`：将参数添加到当前 `StringBuilder` 对象的末尾。`xxx` 可以是任何类型，如 `String`、`int`、`char` 等。
- `setlength(int len)`：将长度设置到 `len` 位。
- `length()`：返回当前容量（字符数）。
- `charAt(int index)`：返回指定位置的字符。
- `delete(int start, int end)`：删除此序列的子字符串中的字符。
- `insert(int offset, xxx)`：将指定数据插入此序列中的指定位置。`xxx` 可以是任何类型。
- `reverse()`：将此字符序列用其反转形式取代。
- `toString()`：将当前 `StringBuilder` 对象转换为 `String` 对象。实则冗余。

ArrayList类

可以动态变化长度，可以用for-each语句遍历。

定义： `ArrayList` 是 `List` 接口的一个实现类，使用动态数组来存储元素。它提供了可变大小的数组，并且支持所有 `List` 接口的方法。

特点：

- 动态数组实现，随着元素的增加，数组会自动扩展。
- 允许快速随机访问元素（通过索引）。
- 插入和删除操作相对较慢（特别是在中间位置），因为需要移动元素。

创建：

```
ArrayList<Integer> intArrayList = new ArratList<Integer>();//尖括号内填写类命，而非数据类型
```

注：BigInteger、BigDecimal 用于高精度的运算，BigInteger 支持任意精度的整数，也是引用类型，但它们没有相对应的基本类型。

如果原括号内有其他的arraylist，则会生成这个arraylist的复印件。

成员方法：

方法	含义
al.add()	添加元素
al.get(index)	访问元素
al.size()	返回长度
al.set(index,)	修改元素
al.remove(index)	删除元素
al.addAll(al2)	把al2全都加入到al中来
al.isEmpty()	是否为空
al.clone()	返回复印件
al.clear()	
al.contains()	判断是否含有
al.indexOf()	返回匹配元素的索引值（第一次出现）
al.lastIndexOf()	返回匹配元素的索引值（最后一次出现）
al.removeAll	
al.subList(int start, int end)	截取子列表

方法	含义
al.sort(Comparator.naturalOrder())	正向排序
al.sort(Comparator.reverseOrder())	反向排序
al.toArray()	
al.toString()	

ensureCapacity()	设置指定容量大小的 arraylist
lastIndexOf()	返回指定元素在 arraylist 中最后一次出现的位置
retainAll()	保留 arraylist 中在指定集合中也存在的那些元素
containsAll()	查看 arraylist 是否包含指定集合中的所有元素
trimToSize()	将 arraylist 中的容量调整为数组中的元素个数
removeRange()	删除 arraylist 中指定索引之间存在的元素
replaceAll()	将给定的操作内容替换掉数组中每一个元素
removeIf()	删除所有满足特定条件的 arraylist 元素
forEach()	遍历 arraylist 中每一个元素并执行特定操作

Java ArrayList | 菜鸟教程 (runoob.com)

静态方法：

方法	含义
Collections.sort(al)	排序

List接口

`List` 是 Java 集合框架中的一个接口，它定义了一组有序的元素，可以包含重复的元素。`List` 接口提供了一些方法用于操纵元素，例如插入、删除、访问和查找元素。`ArrayList` 是 `List` 接口的一个常用实现。

定义： `List` 是一个接口，它继承了 `Collection` 接口。`List` 中的元素是有序的，可以通过索引访问，允许有重复的元素。

常用方法：

- `add(E e)`：在列表末尾添加元素。
- `add(int index, E element)`：在指定位置添加元素。
- `remove(int index)`：移除指定位置的元素。
- `get(int index)`：获取指定位置的元素。
- `set(int index, E element)`：替换指定位置的元素。
- `size()`：返回列表中的元素数量。

- `isEmpty()`：判断列表是否为空。
- `contains(Object o)`：判断列表是否包含某个元素。
- `indexOf(Object o)`：返回元素在列表中第一次出现的位置。

LinkedList 类

定义： `LinkedList` 也是 `List` 接口的一个实现类，它使用链表数据结构来存储元素。`LinkedList` 还实现了 `Deque` 接口，因此可以作为队列或双端队列使用。

特点：

- 链表实现，插入和删除操作更快（特别是在中间位置），不需要移动元素。
- 随机访问速度较慢，因为需要从头部或尾部开始遍历。
- 适合频繁的插入和删除操作。

示例代码：

```
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample {
    public static void main(String[] args) {
        // 使用 LinkedList 实现 List 接口
        List<String> myList = new LinkedList<>();

        // 添加元素
        myList.add("Dog");
        myList.add("Cat");
        myList.add("Cow");

        // 在指定位置添加元素
        myList.add(1, "Horse");

        // 获取元素
        System.out.println("Element at index 1: " + myList.get(1));

        // 设置元素
        myList.set(1, "Elephant");
        System.out.println("Element at index 1 after setting: " + myList.get(1));

        // 删除元素
        myList.remove(2);
        System.out.println("List after removing element at index 2: " + myList);

        // 列表的大小
        System.out.println("Size of list: " + myList.size());

        // 判断列表是否包含某个元素
        System.out.println("List contains 'Dog': " + myList.contains("Dog"));

        // 遍历列表
        System.out.println("Elements in list:");
```

```
        for (String element : myList) {
            System.out.println(element);
        }
    }
}
```

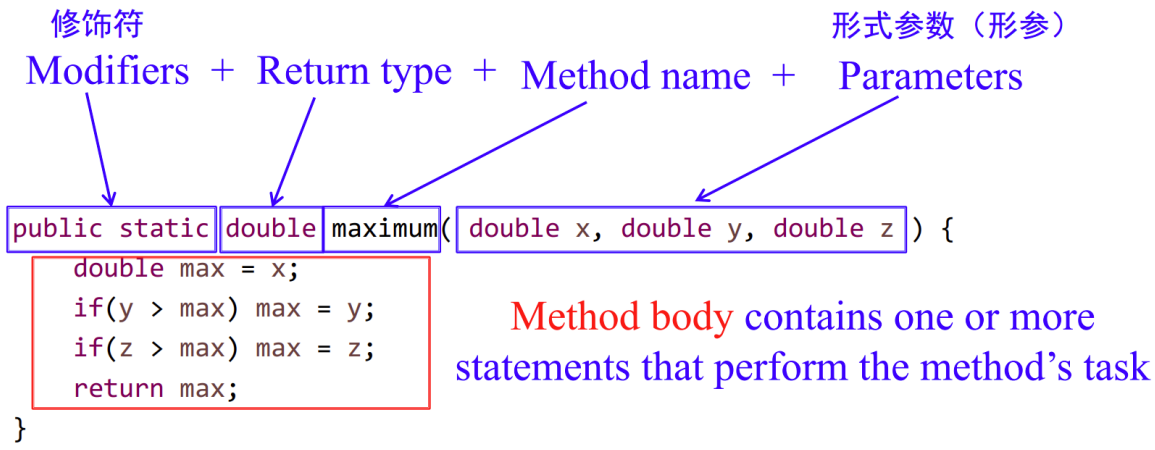
总结

- `List` **接口**：定义了一组有序的元素，可以包含重复的元素，提供了许多方法来操纵列表中的元素。
- `ArrayList` **类**：`List` 接口的一个实现，使用动态数组存储元素，适合频繁访问元素的场景。
- `LinkedList` **类**：`List` 接口的另一个实现，使用链表存储元素，适合频繁插入和删除元素的场景。

Parts2

Methods 方法

声明 declaring:



注：java中定义的顺序无关紧要，只能在类里定义方法，无法在方法中定义方法；

调用 call or invoke:

可以跨类调用ooo

参数传递 passing argument:

按值传递

可以是primitive types 也可以是 references types

如果是reference types会把地址传过去，对此数组的操作会影响原数组的值

注：若函数中需要输入数据，传同一个Scanner进去，若OJ，只会给一个in输入；若命令行，则回车一次算一次；

？主函数是什么意思：

Using Command-Line Arguments

- We can pass arguments from the command line to a Java application by including a parameter of type `String[]` in the parameter list of `main`.

```
public static void main(String[] args)
```

- By convention, this parameter is named `args`.
- When an application is executed using the `java` command, Java passes the command-line arguments that appear after the class name in the `java` command to the `main` method as `String`s in the array `args`.

传递数量不定的变量列表：

```
public static double average(double... numbers){
    double total=0.0;
    for(double d : numbers){
        total+=d;
    }
    return total/numbers.length;
}
```

方法调用栈：

方法重载：Method overloading

方法签名：method signature

name of the method, numbers, types and orders of the parameters

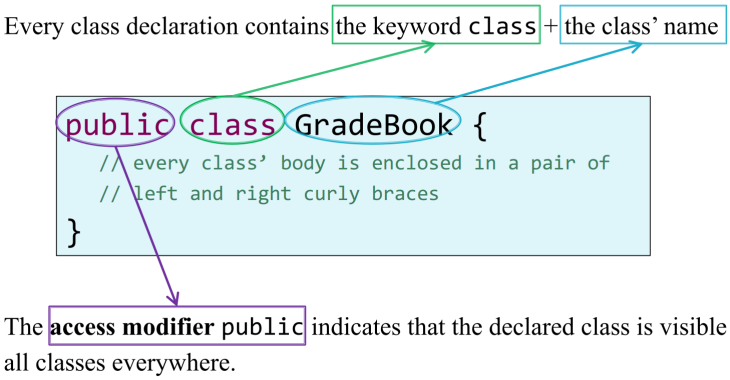
方法签名与函数返回值类型无关

面向对象编程： （dead）

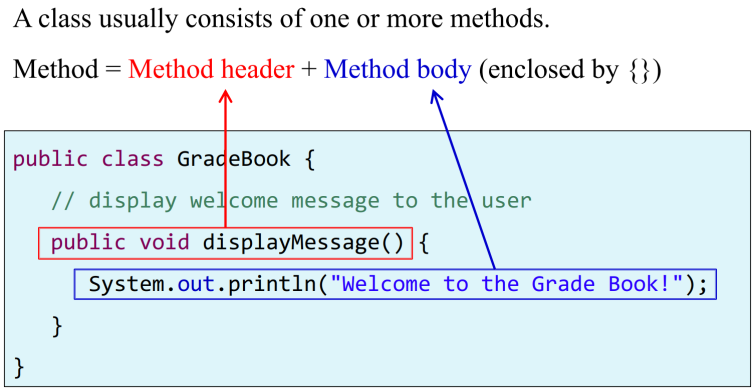
The Whole Picture

- Class – a car's engineering drawings (a blueprint/template)
  - Variable – to specify the attributes (e.g., color, speed)
  - Method – designed to perform tasks (e.g., making a car move)
- Instance / Object – the real car that we drive
- Method call – perform the task (pressing the accelerator pedal)

类的声明：declaring a class



方法的声明：



实例化（创建类变量）：Instantiation

```
GradeBook myGradeBook = new GradeBook();
```

Argument and Parameters:

argument 是实际参数，即传给方法的值

parameter 是形式参数，即函数中声明的参数名

类的属性：Class attributes/fields

即在类中创建的变量

instance variable：每个实例都有fields的copy

public: 其他类可以访问

private: 其他类不能访问

习题课：

成员方法：由对象调用

可以调用静态变量

，根据实例的参数得值

静态方法：由类名调用

，根据传递参数得值

在成员函数内，可以调用成员变量

成员变量和局部变量：

成员变量随着对象的创建而存在，局部变量随着方法的调用而存在

成员变量有默认初始值，局部变量没有默认初始值

局部变量可以和成员变量名称相同，就近原则

类：

实例化：

```
Circle c = new Circle();//第一个Circle相当于声明，后一个其实是构造函数
```

类的属性：

(attribute, field)

在类中定义的变量是全局变量，能被类中的方法使用（会有初始化，0，false，null）。

private: 只能被这个类内部访问，外部无法赋值，也无法访问其值

public: 可以直接被类的外部访问，也可以写入

default(no modifier): 包内，如果一个类、成员变量、方法或构造器没有明确指定访问修饰符，那么它就会有default访问级别，有时也称为包访问级别 (Package-Private)。default访问级别意味着该元素只能被同一个包内的类访问。它对于同一包外的类是不可见的。

protected: protected修饰的元素不仅可以被同一个包内的类访问，还可以被不同包中的子类访问。这意味着如果一个类继承了使用protected修饰的另一个类，即使这两个类不在同一个包中，子类也可以访问父类中的protected成员。

成员变量(aka实例变量instance variables)：储存在对象的内存空间里，只能被成员方法访问（因为只能依靠对象访问），无法被静态方法访问。**在本类中可以直接用变量名访问，也可以用`this`访问（recommended，因为声明是成员变量）。**（会有初始化，0，false，null）

静态变量(static variables)：储存在类的内存区域里，不需要实例化就能直接访问。**在本类中可以直接用变量名访问，也可以用类名访问（recommended，因为声明是静态变量）。**（会有初始化，0，false，null）

注：如果在方法内的局部变量名字与全局变量冲突，可以用类命或者`this`区分。

方法：

静态方法(static)：被类命调用，无需实例就能调用，所以主要根据传入的参数和静态变量返回值

成员方法：由实例调用，结合属性、静态变量和传入的参数得返回值

构造方法：Constructor

（当类没有构造方法时，会自带一个缺省构造方法；如果有了，则会被覆盖掉）

没有返回类型**（没有void，因为void是返回值为空）**

如果要在外部调用，则可以直接public+类命

方法名与类名相同，可以重载，可以无参数。

感觉是依附于类

toString方法：

（每个object都缺省了一个toString方法，会直接返回reference地址；重写toString可能会更有意义）

不需要写出toString才会备用，直接输出reference就会用。

感觉像是依附于object

数据封装：Data Encapsulation

在其他方法里要输出数据的时候，尽量用get函数，因为这样无论要更改什么，只需要改变get函数就可以实现所有函数中的输出都被更改。（主要是程序模块化）

包 Packages：

声明包：

```
package home.hlh;

public class Length{
    public static love_lhh(){

    }
}
```

调用包：

```
import home.hlh.Length;
import home.hlh.*;//import multiple classes
```

or

```
home.hlh.Length.love_lhh();
```

（包内的成员变量若没有modifier，则默认包私有，比private高，比public低）

静态调用：

```
import static home.hlh.Length;

{
    love_lhh();
}
```

类的组合 Composition：

has-a relationship

一个类用另外一个类定义实例变量

类的继承 Inheritance：（specialization）

is-a relationship `Superclass <-> Subclass`

- 一个子类只能继承一个父类，一个父类可以有多个子类（但是类似于多继承的功能可以通过接口实现）
- 子类继承父类，即子类继承了父类中定义的所有属性、方法
- 子类继承父类，子类构造方法的第一行一定要记得调用父类的构造方法（除非super无参数，会自动补全）

- `this`：用来表示当前对象的构造方法this（），当前对象的方法this.methodName（）
- `super`:表示当前对象的父类

继承语句

```
public class BasePlusCommissionEmployee extends CommissionEmployee{

}
```

访问权限

	private	default	protected	public
Same package same class	ok	ok	ok	ok
Same package other classes		ok	ok	ok
Other packages Other classes. Inheritance			ok	ok
Other packages Other classes. No inheritance				ok

重写 Overiding:

在子类中重写一个signature相同的方法。（则对应的子类实例会用对应的方法）

事实上，只要参数不同的方法都可以认为是完全不同的方法，因为parameter也是signature的一部分。

枚举类 Enumerations:

与其它类的区别：在定义的时候就确定了它的对象（实例化）。

枚举类也可以有自己的属性、方法。

若要传入参数，则在声明枚举值的时候要传入参数。

外部调用：可以用`Direction.NORTH`这样调用，或者`Direction.values()`，也可以用`EnumSet.range(from_name,to_name)`来返回一定区间的数组（顺序和声明的顺序相同）

常用场景与实例：方向、星期数、行星等

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST; //objects of Direction
    //若只有这一行，则引号不必须
}
```

```
public enum Planet {
    MERCURY(3.303e+23, 2.4397e6),
    VENUS(4.869e+24, 6.0518e6),
    EARTH(5.976e+24, 6.37814e6),
    MARS(6.421e+23, 3.3972e6),
    JUPITER(1.9e+27, 7.1492e7),
```

```
    SATURN(5.688e+26, 6.0268e7),
    URANUS(8.686e+25, 2.5559e7),
    NEPTUNE(1.024e+26, 2.4746e7);

    // Universal gravitational constant in m^3 kg^-1 s^-2
    private static final double G = 6.6730OE-11;

    private final double mass;   // in kilograms
    private final double radius; // in meters

    // Constructor
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }

    public double getMass() {
        return mass;
    }

    public double getRadius() {
        return radius;
    }

    // Method to calculate gravitational force
    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    // Method to calculate weight on this planet
    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}

public class TestPlanets {
    public static void main(String[] args) {
        double earthWeight = 75;  // 75 kg
        double mass = earthWeight / Planet.EARTH.surfaceGravity();
        for (Planet p : Planet.values()) {
            System.out.printf("Your weight on %s is %f%n", p,
p.surfaceWeight(mass));
        }
    }
}
```

方法

方法	含义
Direction.values()	返回一个数组，枚举类所有object的值
Direction.ordinal()	返回枚举常量的index
Direction.valueOf()	返回指定字符串值的枚举常量（方便遍历）

默认的toString方法是返回枚举值/实例的名字。重写后，若还要返回这个，则是`this.name`。

构造方法会执行与常量个数相同的次数

多态 Polymorphism：

（多种形态，一个大概念可以有很多种不同的具体实现，所以它的子类可以重写他的方法。与抽象类搭配使用，则要求关键方法）

（依靠继承实现；与继承的区别在于，继承强调同，而多态则是不同（override的方式重写））

tips:

可以将不同子类的实例都放入父类的ArrayList里

父类的实例想用子类的方法，则只需要在父类中添加子类的方法名：（实际上是子类重写了父类的方法）

父类的元素可以用子类的实例化。

多态字面意思是“多种形态”。在Java中，多态是一种机制，允许你可以引用父类的对象来表示一个子类的实例。通过多态，我们可以写出更加通用的代码，而这些代码在运行时可以根据具体的对象类型来表现出不同的行为。

多态的实现主要有两种方式：

1. **方法重写（Override）**：子类重写继承自父类的方法。
2. **接口实现（Implement）**：类通过实现接口来提供接口方法的具体实现。

多态的好处：

- **提高代码的可复用性**：可以使用相同的接口或父类引用来调用不同子类的实现。
- **提高代码的可扩展性**：新增子类对多态方法的不同实现时，不需要修改调用这些方法的代码。
- **接口化编程**：允许开发者编写灵活的代码，对“做什么”和“怎么做”进行分离。

抽象类：

- 定义：

  抽象类是不能被实例化的类，通常用作其他类的基类（父类）。抽象类可以包含抽象方法和具体方法。抽象方法是没有实现的方法，它们只有声明而没有具体的执行体。

- 特质：

  抽象类无法实例化自身

  只有抽象类才能定义抽象方法

  提供一个适当的基类，该基类定义了子类必须实现的方法。

封装了一些可以由多个子类共享的代码。

```
abstract class Animal {
    abstract void sound(); // 抽象方法，没有方法体

    void sleep() { // 具体方法，有方法体
        System.out.println("This animal is sleeping.");
    }
}
```

接口 Interface：

接口就像是一个契约或协议，它规定了某个类必须具备哪些功能（方法），但不关心具体的实现细节。任何实现这个接口的类都必须实现这些方法。

接口是一个完全抽象的类，它包含了一组抽象方法（只有方法签名，没有方法体）和常量（静态常量）。一个类可以实现多个接口，从而解决 Java 中单继承的局限。

接口的定义与实现

定义接口

接口使用 `interface` 关键字来定义。接口中的方法默认是 `public` 和 `abstract` 的，即使不显式声明。接口中的属性默认是 `public static final` 的。

```
// 定义一个接口
interface Animal {
    void eat();    // 抽象方法，没有方法体
    void sleep();  // 抽象方法，没有方法体
}
```

实现接口

类使用 `implements` 关键字来实现接口。实现接口的类必须提供接口中所有方法的具体实现。

```
class Dog implements Animal {
    @Override
    public void eat() {
        System.out.println("Dog eats.");
    }

    @Override
    public void sleep() {
        System.out.println("Dog sleeps.");
    }
}

class Cat implements Animal {
    @Override
    public void eat() {
        System.out.println("Cat eats.");
    }

    @Override
    public void sleep() {
```

```
        System.out.println("Cat sleeps.");
    }
}
```

接口可以弥补多继承，implement多个接口可以让一个类也能有多个类的功能。

排序器 Comparable interface：

（让特定类可以比较大小）

返回negative（-1）表示当前对象（this）排在前面；

```
public class Circle extends Shape implements Comparable<Circle>
```

```
@Override
public int compareTo(Circle o) {
 if(this.radius < o.radius){
 return 1;
 }else if(this.radius > o.radius){
 return -1;
 }
 return 0;
}
```

给一个不可比较的类接入一个比较器的接口，就相当于给予这个类某种可以比较的规则；

```
Collection.sort(circleList);
```

排序器 Comparator interface：

创建一个排序器，

返回negative（-1）表示左侧对象（o1）排在前面；

```
Class NComparator implements Comparator<Circle>{
    //重写方法，定义规则
}
```

只需要切换排序器，就能使一个类有不同的排序方式

```
NComparator n= new NComparator();
circleList.sort(n);
```

泛型Generic：

泛型方法：

```
public static <T> void printArray(T[] array){
    //可以换字母
}
```

```
public static <T extends Number> void printArray(T[] array){
    //上限是Number，T要是Number及其子类
}
```

Parts3

Final语句

用于定义常量，或者说值不再改变的量。

final变量一定要在类的构造方法结束之前被初始化。

即声明时候初始化，或者在构造方法中初始化。

栈内存（stack memory）和堆内存（heap memory）：

	Stack	Heap
Pros	访问速度快 不需要手动控制内存	可以全局访问 没有空间大小限制
Cons	只能存放局部变量 Stack空间有限	访问速度较慢 必须手动管理内存

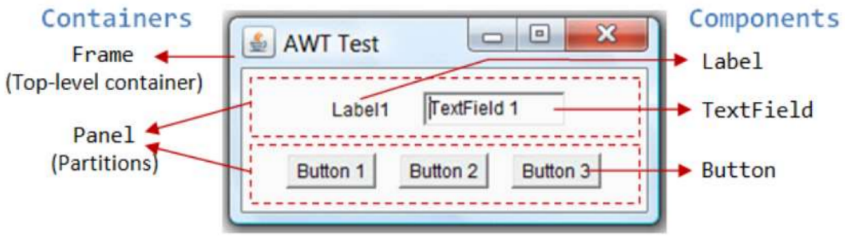
图形用户界面 GUI：

Graphic User Interface & Command-Line Interface

常用GUI开发工具APIs：AWT，Swing，JavaFX

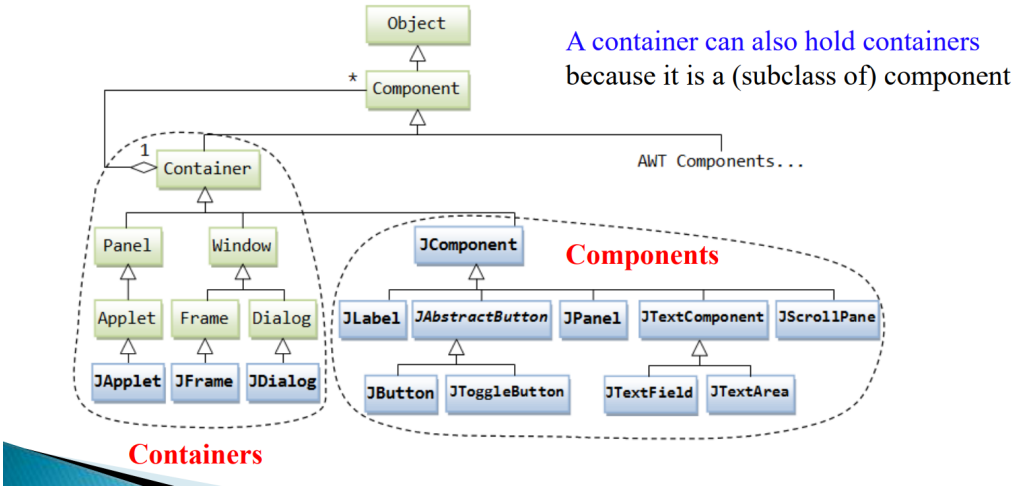
Java GUI Core Concepts

- **Component** (组件): Components are elementary GUI entities, such as Button, Label, and TextField.
- **Container** (容器): used to hold components in a specific layout
- **Event handling** (事件处理): decides what should happen if an event occurs (e.g., a button is clicked)



Java GUI Class Hierarchy

- There are two groups of classes (in package `javax.swing`): **containers** and **components**. A container is used to hold components.



详情：CS109_Ch12_GUI.pdf

Swing API：

JFrame & JComponent

注意x,y坐标

最后大家如果觉得有用可以打赏一下哦~（可能以后的某天我失意的时候恰好看到...）

推荐使用微信支付

presumably(**恒)

微信支付