### 数组：

招式

定义：

静态定义

|  |
| --- |
| int[] intArr;  intArr = new int[]{1, 2, 3, 4, 5, 6, 9, 22, 10, 89, -1, -99, 0};  //此外还可以简化写法  String[] objArr2 = {"we","are","learning","JAVA", "."}; |

动态定义

|  |
| --- |
| //声明数组变量是固定长度  int[] intArr2 = new int[5];  for(int i = 0; i < intArr2.length; i++) {  intArr2[i] = i + 1;  } |

数组常用的遍历方法

|  |
| --- |
| for(int i = 0; i < intArr2.length; i++)  System.out.println(intArr2[i]);    for(int a : intArr2)  System.out.println(a); |

数组标识符实际就是指向真实对象的一个引用，那些对象存在于内存中的堆中。

心法：

java中不存在多维数组，在内存中都是以线性数组的形式存在。

数组与集合的区别：

1. 集合只能装载引用类型，数组可以装备基本类型
2. 正因为原因1，如果装载基本类型，集合只能使用对应的封装类型，在访问效率上，由于数组的元素在内存中是连续的，所以效率高高于使用集合。
3. 数组功能弱，定义时必须固定长度

练习：

<https://www.hackerrank.com/challenges/camelcase/problem>

<https://www.hackerrank.com/challenges/angry-professor/problem?h_r=profile>

<https://www.hackerrank.com/challenges/beautiful-days-at-the-movies/problem>

<https://www.hackerrank.com/challenges/arrays-ds/problem>

<https://www.hackerrank.com/challenges/sparse-arrays/problem>

<https://www.hackerrank.com/challenges/mars-exploration/problem>

<https://www.hackerrank.com/challenges/strong-password/problem>

<https://www.hackerrank.com/challenges/2d-array/problem>

### 类：

字符：人类可以读懂的语言

字节：机器可以识别的“语言”，专业术语-机器指令

编译：可以类似认为人类的语言翻译成机器指令

在eclipse/STS项目目录下，有src存放的是源文件，bin则放的是编译后的二进制文件

Java的程序编译后才能够运行

类的封装规则：

priavte/default/protected/public

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **private** | **default** | **protected** | **public** |
| 同一个类中 | ○ | ○ | ○ | ○ |
| 同一个包中 |  | ○ | ○ | ○ |
| 子类中 |  |  | ○ | ○ |
| 全局范围 |  |  |  | ○ |
|  | 类中 | 包中 | 子类中 | 全部 |

方法重写： 子类覆盖父类的方法

方法重载： 相同方法名，不同参数列表，列如

|  |
| --- |
| public void print() {  System.out.println("I'm a sub-class");    }    public void print(String abc) {  System.out.println(abc);  }    public void print(int abc) {  System.out.println(abc);  }    public void print(String abc, int bcd) {  System.out.println(abc + bcd);  } |

变量的作用域与生命周期：

变量的生命周期

|  |
| --- |
| {  int a = 5;//变量a的作用域仅限在花括号里  }//变量a的生命结束 |

子类构造器的执行顺序

|  |
| --- |
| /\*\*  \* 1. 调用父类的构造器HelloWorld()  \* 2.调用Object的构造器 Object()  \* 3. 执行完Step2  \* 4.执行完Step1  \* 5.执行子类的构造器 HelloWorld2()  \* 6.初始化对象变量 Integer abc = 1  \*  \* 简单一句话，生成子类对象前要先生成父类对象  \*  \*/  HelloWorld2 w2 = new HelloWorld2(); |
|  |

### final

final修饰的类不能被继承，同样父类中的方法如果被final修饰，子类不能够重写

心法：

final 修饰的变量必须在使用时已经赋值了。

所以final修饰的类的成员变量，可以在**1.声明时赋值2.类的构造器内赋值**。 以上两点都保证了类的对象生成时，final的成员变量也完成赋值了。

但是static final 修饰的类的成员变量，必须在**1.声明时赋值2.类中static代码块赋值**，因为static 的静态变量在类的加载时（对象生成前）就调用了。

final修饰的引用变量，例如Map, 在Map<> map = new HashMap<>(),后可以对map随意put()/remove()等，因为map的对象没变，只是内部引用数据改变。但是不能之后再 map = new HashMap<>(); 因为这样map的对象就变化了。

所以private final Map<String, String> map = new HashMap<>();

之后，还可以进行map的put/remove,如何禁止呢？可以使用Collections的内部类UnmodifiableMap

final Map<> map = Collections.unmodifiableMap(values)

其实就是继承Map，并重写put/remove方法，使其抛出异常。

抽象类： 可以含有抽象方法，及只有方法签名没有方法的具体实现,抽象类要用abstract关键字

接口：类似抽象类，可以含有抽象方法，但是可以不用abstract修饰符

一个类只能继承一个父类，但是可以实现多个接口

归并排序：

<https://blog.csdn.net/qq_28081081/article/details/80603251>

Enum枚举类

1.枚举类默认使用final修饰符，即不能够诞生子类

2.成员变量使用final修饰符，即一般在内部调用构造器时进行赋值

3. 构造器使用private修饰符，即只能在类的内部通过构造器生成枚举对象

HashSet

hashSet集合判断两个**元素相等**的标准是通过两个对象的equals()和hashCode()方法比较，只有全部相等才视为同一元素。所以对自定义类设计时应注意保持equals()和hashCode()一致。

即，equals返回true，并且hashCode相等。

Set实际上就是通过将Map的value设为null,保留key.