# cmd常用命令

盘符+:即可直接切换盘符

dir:功能类似ls,但是默认不加参数也会展示所有文件，包括隐藏

cd

cd /回到盘符目录

cls清屏

tree 【文件名】会以树的形式展示出此目录的所有目录

## Java语法

Java面向对象三大特征

封装，继承，多态

封装：对象代表什么就得封装对象的数据并提供数据所对应的行为

继承：当类与类之间存在相同内容时，过多的相同内容重复写太麻烦，可以把相同内容提到一个单独类中，让子类继承：public class子类 extends 父类{}

**单继承：Java 中一个子类只能直接继承一个父类，这是为了避免多继承带来的复杂性。​**

**传递性：如果类 A 继承类 B，类 B 继承类 C，那么类 A 也间接继承了类 C 的属性和方法。**

**访问权限：子类只能访问父类中 public、protected 修饰的成员，不能访问 private 修饰的成员。**

**(Java 继承中，构造方法无论权限子类均不能继承；成员变量无论私有或非私有子类都能继承（private 需通过父类方法访问 ）；成员方法子类能继承非私有的，不能继承私有的 。)**

**但是对于子类的构造方法第一行必须先用super()调用父类构造，父类有参无参按自己需求写,**

**同时this和super不能出现在同一构造中，若是因为父类有无参而省略super的情况可以写this**

**子类可以根据自身需求，对父类的方法进行重写（Override），重写时需要满足以下条件：**

**方法名、参数列表必须与父类相同。​**

**返回值类型与父类相同或是父类返回值类型的子类。​**

**访问权限不能低于父类的访问权限。**



//java的注释跟c一样（

值得一提的是Java的标识符还增加了$美元（？？？）

Java的关键字都是小写的但是不包括类似System这种

java的数据类型分两种，一种是基本数据类型，一种是引用数据类型，基本就是下面的八种



基本数据类型变量存储的是真实数据，而引用数据类型存储的是地址值

Java的地址值是十六进制

需要注意long和float在末尾要加l和f，大小写都行，如long a=10L;

字面量：就是变量类型

java数字默认为十进制，可以通过添加前缀改变0b为二进制，0为八进制，0x为十六进制

public class aaa {  
 public static void main(String[] arg) {  
 System.*out*.println("这是一个简单的Java程序");  
 }  
}

其中第一行public相当于代码入口，一个Java文件只能有一个public类，剩下为普通类，不加public，不同类之间一般没有关系，但也可选择调用，而第二行当于函数开头，在Java里称为方法,注意括号内不能为空，因为这是给命令行参数的预留位,String的s要大写;

Print默认不换行，加上ln换行

在 Java（和所有面向面向对象编程****语言）中，**** 对象（Object）\*\* 是类（Class）的****具体实例****，可以理解为 “根据模板创建出的具体事物”。比如Car myCar = new Car();然后可以通过mycar.xxxxx来进行下一步操作

* System.out.println就是输出到控制台，**System**：Java 自带的一个 “系统类”，包含与系统交互的核心功能（比如输入输出、获取系统信息等）。
* **out**：System 类中的一个****静态成员变量****，代表 “标准输出流”（默认就是控制台）。
* **println()**：out 对象的一个****方法****，作用是 “打印参数内容并自动换行”（print line 的缩写）。
* Static就是静态的意思，变量加上这个则这个变量作用域为这个类，这个类共用这个变量
* 方法是Java程序执行的最小单元

**Java中的方法可以同名，但是参数或参数个数或顺序要不同**

**形如**public int add(int a, int b) { ... }

public double add(int a, int b) { ... } /

**不算重载会报错**

// 1. 两个 int 相加

public int add(int a, int b) {

return a + b;

}

// 2. 三个 int 相加（参数个数不同 → 重载）

public int add(int a, int b, int c) {

return a + b + c;

}

// 3. 两个 double 相加（参数类型不同 → 重载）

public double add(double a, double b) {

return a + b;

}

// 4. int 和 double 相加（参数顺序不同 → 重载）

public double add(int a, double b) {

return a + b;

}

// 5. double 和 int 相加（参数顺序不同 → 重载）

public double add(double a, int b) {

return a + b;

}

****静态方法（**static**修饰）****：属于****类本身****，是类级别的功能，不依赖任何对象存在。  
可以理解为：“这个功能是类天生就有的，不需要创建具体对象就能用”。

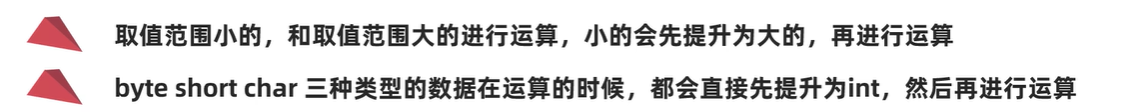
****非静态方法（无**static**）****：属于****类的对象****，是对象级别的功能，必须通过具体对象才能调用。  
可以理解为：“这个功能是每个对象独有的能力，只有创建了对象，才能使用它”。

使用静态方法可以直接类名.方法名

而使用非静态就需要先创建一个此类的对象再对象名.方法名



**隐式转换（自动类型提升）：自动把一个取值范围小的数值转成取值范围大的数据**



Char是变为字符对应的编码

Java支持字符串相加，如”123”+1=”1231”

数组建立支持两种格式

一：数组类型[] 数组名

二:数组类型 数组名[]

#### 构造

创建对象时由虚拟机调用的方法，需要与类名相同，不写的话默认虚拟机会自动添加一个无参构造，如果写了有参则无参不再默认创建

public class Student {

private String name;

// 无参构造

public Student() {

name = "默认名";

}

// 有参构造

public Student(String name) {

this.name = name;

}

// 构造方法调用其他构造（this()）

public Student(String name, int age) {

this(name); // 调用上面的有参构造

// 处理age...

}

// 使用

public static void main(String[] args) {

Student s1 = new Student(); // 调用无参构造

Student s2 = new Student("张三"); // 调用有参构造

}

}

**Javabean规范**

****核心规范****：

1. 类必须是公共的（public），且有公共无参构造方法
2. 成员变量通常为私有（private）
3. 通过 setXxx() 和 getXxx() 方法访问 / 修改成员变量（遵循命名规范）

（嫌麻烦有快捷键，alt+insert/alt+fn+insert）

##### ****字符串****

**对于字符串来说，==比较的是地址值**

所以对于String a=”abc”;String b=new String(“abc”);a和b不相等

故而可以使用equals(完全一致)或equalsIgnoreCase(忽略大小写)

返回的是bool值

byte[] a={12,22,33,44};  
String s=new String(a);

字符串会把a的内容按ASCII码转换

但是这种格式不支持char和byte以外的参数，以外的参下

**StringBuilder**

**一个内容可变的容器**

可以空参构造，也可有参public StringBuilder(String str);

常用方法有

append(任意类型)：可以往 StringBuilder 里添加各种类型的数据，比如字符串、数字等。添加之后会返回当前这个 StringBuilder 对象本身，所以能像链条一样连续调用。

reverse()：能把 StringBuilder 里存的内容反过来。比如说原来存的是 “abc”，调用这个方法后就变成 “cba” 了。

length()：用来获取 StringBuilder 里字符的数量，也就是里面内容的长度。

toString()：可以把 StringBuilder 对象转换成 String 类型，这样就能方便地以字符串的形式去使用或者输出了。

**StringJoiner**

也是一种内容可变容器，但是可以指定格式，需注意没有空参形式，拼接时比较方便

public StringJoiner (间隔符号)/public StringJoiner (间隔符号,开始符号，结束符号)

常用方法有

add(CharSequence newElement)：向 StringJoiner 中添加单个元素，添加后会自动用分隔符与之的元素分隔。

merge(StringJoiner other)：将另一个 StringJoiner 中的所有元素合并到当前对象中，自动处理分隔符，不包含对方的前缀和后缀。

toString()：返回最终拼接好的字符串（包含前缀、所有元素、分隔符和后缀）。

length()：返回最终字符串的长度（包含前缀、后缀、元素和分隔符的总长度）。

其中CharSequence很有用，这是一个接口，相当于一种规范常用的 String、StringBuilder、StringBuffer 都实现了这个接口，它提供了几个常用方法可以对符合这个接口的不同类型数据统一处理

charAt(int index)：返回序列中指定索引位置的字符（索引从 0 开始）。

length()：返回字符序列的长度（包含的字符个数）。

subSequence(int start, int end)：返回从 start 索引（包含）到 end 索引（不包含）的子序列。

toString()：返回该字符序列的字符串表示。

**StringBuffer**

StringBuffer 是线程安全的可变字符串类，适合在多线程场景中进行频繁的字符串修改操作。如果是单线程环境，推荐使用效率更高的 StringBuilder；如果字符串内容固定不变，则直接用 String 即可。

append(任意类型)：向字符串末尾添加数据（如字符串、数字、布尔值等），返回当前 StringBuffer 对象本身，支持链式调用。

insert(int offset, 内容)：在指定索引位置（offset）插入内容，原位置及后续字符自动后移。

delete(int start, int end)：删除从 start 索引（包含）到 end 索引（不包含）的字符。

deleteCharAt(int index)：删除指定索引位置的单个字符

replace(int start, int end, String str)：用字符串 str 替换从 start 到 end（左闭右开）的字符。

reverse()：将字符串内容反转。

length()：返回字符串的长度（字符个数）。

charAt(int index)：返回指定索引位置的字符。

toString()：将 StringBuffer 转换为 String 类型。

**值得一提的是StringBuffer与StringBuilder的方法名基本上都一样**

###### 集合

## 一、核心体系

Collection：存储单个元素

List：有序、可重复（有索引）

Set：无序、不可重复（无索引）

Queue：先进先出（队列）

Map：存储键值对（key 唯一，value 可重复）

## 二、常用类及用法

### 1. List

#### ArrayList

特点：查询快、增删慢（底层是动态数组）

用法：

List<String> list = new ArrayList<>();

list.add("a"); // 添加元素

String val = list.get(0); // 获取索引0的元素

适用场景：频繁查询，少量增删

#### LinkedList

特点：增删快、查询慢（底层是双向链表）

用法：

List<Integer> list = new LinkedList<>();

list.add(1);

list.remove(0); // 删除索引0的元素

适用场景：频繁增删，少量查询；也可作为队列使用

### 2. Set

#### HashSet

特点：无序、去重快（底层是哈希表）

用法：

Set<String> set = new HashSet<>();

set.add("x");

set.add("x"); // 重复元素，添加失败

适用场景：需要去重，不关心顺序

#### LinkedHashSet

特点：有序（按插入顺序）、去重

用法同 HashSet

适用场景：需要保留插入顺序的去重

#### TreeSet

特点：自动排序、去重（底层是红黑树）

用法：

Set<Integer> set = new TreeSet<>();

set.add(3);

set.add(1); // 最终集合会按1、3排序

适用场景：需要排序的去重

### 3. Queue

#### LinkedList

特点：支持队列基础操作

用法：

Queue<String> queue = new LinkedList<>();

queue.add("first"); // 入队

String first = queue.poll(); // 出队（获取并删除队首）

适用场景：简单队列场景

#### PriorityQueue

特点：按优先级排序出队（默认自然排序）

用法同 LinkedList

适用场景：需要优先级的队列

### 4. Map

#### HashMap

特点：无序、键值对存储快（底层是哈希表）

用法：

Map<String, Integer> map = new HashMap<>();

map.put("age", 20); // 存键值对

int age = map.get("age"); // 根据key取value

适用场景：一般键值对场景

#### LinkedHashMap

特点：有序（按插入顺序）

用法同 HashMap

适用场景：需要保留插入顺序的键值对

#### TreeMap

特点：key 自动排序（底层是红黑树）

用法同 HashMap

适用场景：需要按 key 排序的键值对

## 三、通用方法

添加：Collection 用 add ()，Map 用 put ()

获取：List 用 get (index)，Map 用 get (key)

删除：remove ()

大小：size ()

包含：contains ()

### 常用类

**导入**





Scanner：用于输入

