Java se

基本数据类型

要记得初始化

byte short int long

float double

char

boole

> *byte*

8位有符号

> *short*

16位有符号

> *int*

32位有符号

> *long*

64位有符号

> *float*

32位浮点数

1+8+23

> *double*

64位浮点数

1+11+52

> *char*

16位 字符

> *boolean*

一个字节 8位

一些问题

浮点数默认double float要加f 大小写均可

整数默认int 声明long 加上l

浮点数精度问题最好还是 用 **decimal.Decimal**

小字节数的数转大字节数的问题不大 类型自动提升 （隐式） 但可能会溢出（额外填充高位）

大转小会可能丢失（强转）（显式）（截取低位）

float转double 精度损失

Integer 包装类

Integer和int互转 会有自动装箱的问题 影响性能 编译器自动完成装箱

包装类适配功能多

包装类实现了 `Comparable` 接口

Java集合中只能存储对象

Integer包装类，我们可以直接使用stream()方法来计算所有元素的和。

有一个缓存池 在-128到127的范围内 不会都生成新的对象实例 而是复用缓存中的对象 直接从内存中取出。

面向对象

封装 继承 多态

多态包括重载重写 接口与实现 向上转型和向下转型

> 重载

不同的函数共享一个函数名 区别是参数不用

> 重写

子类覆盖父类的方法

> 接口与实现

多个类实现同一个接口的方法

> 向上下转型

可以使用父类类型的引用指向子类对象

向下指父类转会子类类型

> 重载

设计原则

> 单一职责原则

一个类应该只负责一项职责

> 开放封闭原则

对扩张开放 对修改封闭

> 里氏替换原则

子类对象应该能够替换掉所有父类对象

> 接口隔离原则

接口应该小而专 通过接口抽象层来实现底层和高层之间的解耦

> 最少知识原则

一个对象应当对其他对象有最少的了解

抽象类与接口

> 抽象类

抽象类只能被单继承（工厂方法？） 不能被实例化

可以包含抽象方法和普通方法：

抽象方法：没有具体实现的方法，必须由子类实现。

普通方法：可以有具体实现，子类可以直接使用或重写。

可以有成员变量：可以包含实例变量，也可以包含具体实现的属性。

构造方法：可以定义构造方法，但不能直接用于创建对象。

> 接口

实现 一个类可以实现多个类

只能有常量和抽象方法 （接口默认都是抽象的（**abstract**），因此可以省略 `abstract` 关键字）

变量默认是 `public`、`static` 和 `final` 的 所以是常量

jdk8以后有静态方法和默认方法

> 接口的默认方法和静态方法区别？

默认方法可以在实现类中被 重写（**override**），以提供不同的实现。如果实现类不重写，接口中的默认实现将被使用。

静态方法不能被实现类 重写。它属于接口本身，不能被继承或重写。接口名 调用

私有方法：私有方法是在 Java 9 中引入的,用于在接口中为默认方法或其他私有方法提供辅助功能。这些方法不能被实现类访问,只能在接口内部使用。

> *key*

抽象类可以包含实例变量和静态变量

实例变量指每个实例都有自己独立的变量

静态变量 是属于类本身的，所有实例共享相同的静态变量。

> 抽象类不能被*final*修饰

final禁止类被继承和重写

static

> 静态变量

所有实例共享该变量 且只初始化一次

> 静态方法

静态方法属于类本身，而实例变量属于对象（实例）。

无实例依赖（例如直接被main调用）

可以直接调用其他静态变量和静态方法

不能直接调用 非静态变量 （因为非静态变量属于实例变量，没实例就访问不了）

静态方法属于类，访问实例的变量和方法时会出错，因为它不知道应该操作哪个实例。

但是实例可以用静态方法（因为方法属于类），只要不涉及到访问非静态变量和方法就行了。

> 静态内部类?

静态内部类可以直接访问外部类的静态成员，但不能直接访问外部类的实例成员。

静态内部类的实例可以通过外部类名来创建（如 `OuterClass.StaticInnerClass`），而不需要先创建外部类的实例。

静态内部类是外部类的一个静态成员，因此可以在没有外部类实例的情况下创建它。

非静态内部类在外部类实例化后才能实例化,而静态内部类可以独立实例化。

非静态内部类不能定义静态成员，而静态内部类可以定义静态成员。

规则

当非抽象类实现接口时,确实必须实现该接口中声明的所有抽象方法。

在实现接口方法时:
- 返回值类型必须完全一致
- 方法名必须完全一致
- 参数列表必须完全一致
- 可以增加但不能减少方法的访问权限(比如接口中默认是public,实现类中也必须是public)

但是其他修饰符是可以改变的,比如:
- 可以添加synchronized修饰符
- 可以添加final修饰符
- 可以添加strictfp修饰符

深拷贝和浅拷贝？

浅拷贝只复制引用

深拷贝新开对象

Arrays.asList是浅拷贝 list.toArray是深拷贝

> 深拷贝的三种方法？

实习Cloneable接口和重写clone方法

使用序列化和反序列化

手动递归复制对象和字段

> 其他八股

具体性能排序:
System.arraycopy > Arrays.copyOf > clone > for循环

泛型

使用一个或多个类型参数 复用方法是、

泛型中的类型在使用时指定，不需要强制类型转换

例如常见的集合 new时需要指定存的类型 不然都是Object

对象

> 创建对象方法

new

Class类的newInstance方法

Constructor类的newInstance方法 （先getConstructor获取构造器方法）

使用Clone方法 克隆另外一个对象

使用反序列化

枚举类

在枚举类中，每一个枚举常量（如 `INSTANCE`）都是该枚举类的一个实例。在 Java 中，枚举类型可以有多个枚举常量，而每个枚举常量都会被视为该枚举类的一个实例。

枚举类不仅可以存储枚举常量，还可以存储其他成员，例如字段、方法、构造函数等。枚举类其实是一种特殊的类，它可以有实例字段、方法、甚至构造函数。

唯一性：每个枚举常量在整个程序中只有一个实例（即枚举常量实例是唯一的），而且是由 JVM 自动创建和管理的。

PS:枚举类的枚举常量是单例 但是想用在自己的类 就意义不大了

强引用、软引用、弱引用和虚引用

强引用 new的时候都是 直到设置为Null或没有引用指向 才会GC

软引用：内存不够的时候 JVM会回收这些对象， SoftReference类实现

弱引用：垃圾回收时就会回收 WeakReference类实现

虚引用：Phantom Reference 仅跟踪生命周期和通知 对象被回收时会被加入 `ReferenceQueue`

与ThreadLocal问题一起考

`ThreadLocalMap` 使用的是弱引用来存储 `ThreadLocal` 对象，这意味着，如果没有强引用指向 `ThreadLocal` 对象，那么垃圾回收器会回收它。

然而，`ThreadLocalMap` 中的值（即线程局部变量）是强引用。问题在于，如果 `ThreadLocal` 被垃圾回收了，但 `ThreadLocalMap` 仍然持有它的强引用，那么这些线程局部变量就不会被及时回收，可能导致内存泄漏。

要手动清理ThreadLocal　remove函数

其他

finally的return会覆盖try的return

== 比较变量本身的值，即两个对象在内存中的首地址，属于数值比较

equals比较字符串包含内容是否相同

对于非字符串变量来说 如果没有对equals方法重写，那 两者作用相同 都是比较对象在堆内存的首地址

String StringBuilder StringBuffer

String不可变 （速度最慢）

StringBuilder 线程不安全 可以变 （速度最快）

StringBuffer 线程安全，可变

> 为什么线程安全

方法用了Synchronized修饰

Stream API

流式编程

用在集合上

filter过滤

collect收集到新集合

sum总和

> 并行流

和ForkJoin相关 不是重点吧只能说 太细了

杂

import java.util.*这个语句的作用是导入java.util包下的所有类，但不包括其子包中的类。C选项准确描述了这一特性。

在Java I/O体系中,流分为字节流和字符流两大类。

字节流:以字节为单位处理数据,包括InputStream和OutputStream体系
字符流:以字符为单位处理数据,包括Reader和Writer体系

Java 传递参数时，基本数据类型按值传递，对象类型传递引用的副本。

|| 左边为true的话 不会运行右边的 &&同理 左边false就不会运行了

|”在*java*中为按位或运算符 ：当两边操作数的位有一边为1时，结果为1，否则为0