

1、java 中 == 和 equals 和 hashCode 的区别

1) == 若是基本数据类型比较，是比较值，若是引用类型，则比较的是他们在内存中的存放地址。对象是存放在堆中，栈中存放的对象的引用，所以 == 是对栈中的值进行比较，若返回 true 代表变量的内存地址相等；

2) equals 是 Object 类中的方法，Object 类的 equals 方法用于判断对象的内存地址引用是不是同一个地址（是不是同一个对象）。若是类中覆盖了 equals 方法，就要根据具体代码来确定，一般覆盖后都是通过对象的内容是否相等来判断对象是否相等。

3) hashCode() 计算出对象实例的哈希码，在对象进行散列时作为 key 存入。之所以有 hashCode 方法，因为在批量的对象比较中，hashCode 比较要比 equals 快。在添加新元素时，先调用这个元素的 hashCode 方法，一下子能定位到它应该旋转的物理位置，若该位置没有元素，可直接存储；若该位置有元素，就调用它的 equals 方法与新元素进行比较，若相同则不存，不相同，就放到该位置的链表末端。

4) equals 与 hashCode 方法关系：

hashCode() 是一个本地方法，实现是根据本地机器上关的。equals() 相等的对象，hashCode() 也一定相等，hashCode() 不等，equals() 一定也不等，hashCode() 相等，equals() 可能相等，也可能不等。

所以在重写 equals(Object obj) 方法，有必要重写 hashCode() 方法，确保通过

`equals(Object obj)`方法判断结果为 `true` 的两个对象具备相等的 `hashCode()`返回值。

5) `equals` 与 `==` 的关系：

`Integer b1 = 127;`在 java 编译时被编译成 `Integer b1 = Integer.valueOf(127);`对于 -128 到 127 之间的 `Integer` 值，用的是原生数据类型 `int`，会在内存里供重用，也就是这之间的 `Integer` 值进行 `==` 比较时，只是进行 `int` 原生数据类型的数值进行比较。而超出 -128~127 的范围，进行 `==` 比较时是进行地址及数值比较。

2、`int`、`char`、`long` 各占多少字节数

`int`\float 占用 4 个字节，`short`\`char` 占用 2 个字节，`long` 占用 8 个字节，`byte`/`boolean` 占用 1 个字节

基本数据类型存放在栈里，包装类栈里存放的是对象的引用，即值的地址，而值存放在堆里。

3、`int` 与 `integer` 的区别

`Integer` 是 `int` 的包装类，`int` 则是 java 的一种基本数据类型，`Integer` 变量必须实例化才能使用，当 `new` 一个 `Integer` 时，实际是生成一个指向此对象的引用，而 `int` 是直接存储数据的值，`Integer` 默认值是 `null`，而 `int` 默认值是 0

4、谈谈对 java 多态的理解

同一个消息可以根据发送对象的不同而采用多种不同的行为方式，在执行期间判断所引用的对象的实际类型，根据其实际的类型调用其相应的方法。

作用：消除类型之间的耦合关系。实现多态的必要条件：继承、重写（因为必须调用父类中存在的方法）、父类引用指向子类对象

5、String、StringBuffer、StringBuilder 区别

都是字符串类，String 类中使用字符数组保存字符串，因有 final 修饰符，String 对象是不可变的，每次对 String 操作都会生成新的 String 对象，这样效率低，且浪费内存空间。但线程安全。

StringBuilder 和 StringBuffer 也是使用字符数组保存字符，但这两种对象都是可变的，即对字符串进行 append 操作，不会产生新的对象。它们的区别是：StringBuffer 对方法加了同步锁，是线程安全的，StringBuilder 非线程安全。

6、什么是内部类？内部类的作用

内部类指在类的内部再定义另一个类。

内部类的作用：1) 实现多重继承，因为 java 中类的继承只能单继承，使用内部类可达到多重继承；2) 内部类可以很好的实现隐藏，一般非内部类，不允许有 private 或 protected 权限的，但内部类可以；3) 减少了类文件编译后产生的字节码文件大小；

内部类在编译完后也会产生.class 文件，但文件名称是：外部类名称\$内部类名称.class。分为以下几种：

1) 成员内部类，作为外部类的一个成员存在，与外部类的属性、方法并列，成员内部类持有外部类的引用，成员内部类不能定义 static 变量和方法。应用场合：每一个外部类都需要一个内部类实例，内部类离不开外部类存在。

2) 静态内部类，内部类以 static 声明，其他类可通过外部类.内部类来访问。特点：不会持有外部类的引用，可以访问外部类的静态变量，若要访问成员变量须通过外部类的实例访问。应用场合：内部类不需要外部类的实例，仅为外部类提供或逻辑上属于外部类，逻辑上可单独存在。设计的意义：加强了类的封装性（静态内部类是外部类的子行为或子属性，两者保持着一定关系），提高了代码的可读性（相关联的代码放在一起）。

3) 匿名内部类，在整个操作中只使用一次，没有名字，使用 new 创建，没有具体位置。

4) 局部内部类，在方法内或是代码块中定义类，

7、抽象类和接口区别

抽象类在类前面须用 abstract 关键字修饰，一般至少包含一个抽象方法，抽象方法指只有声明，用关键字 abstract 修饰，没有具体的实现的方法。因抽象类中含有无具体实现的方法，固不能用抽象类创建对象。当然如果只是用 abstract 修饰类而无具体实现，也是抽象

类。抽象类也可以有成员变量和普通的成员方法。抽象方法必须为 `public` 或 `protected` (若为 `private` , 不能被子类继承, 子类无法实现该方法)。若一个类继承一个抽象类, 则必须实现父类中所有的抽象方法, 若子类没有实现父类的抽象方法, 则也应该定义为抽象类。

接口用关键字 `interface` 修饰, 接口也可以含有变量和方法, 接口中的变量会被隐式指定为 `public static final` 变量。方法会被隐式的指定为 `public abstract`, 接口中的所有方法均不能有具体的实现, 即接口中的方法都必须为抽象方法。若一个非抽象类实现某个接口, 必须实现该接口中所有的方法。

区别：

- 1) 抽象类可以提供成员方法实现的细节, 而接口只能存在抽象方法;
- 2) 抽象类的成员变量可以是各种类型, 而接口中成员变量只能是 `public static final` 类型;
- 3) 接口中不能含有静态方法及静态代码块, 而抽象类可以有静态方法和静态代码块;
- 4) 一个类只能继承一个抽象类, 用 `extends` 来继承, 却可以实现多个接口, 用 `implements` 来实现接口。

7.1、抽象类的意义

抽象类是用来提供子类的通用性, 用来创建继承层级里子类的模板, 减少代码编写, 有利于代码规范化。

7.2、抽象类与接口中的应用场景

抽象类的应用场景：1) 规范了一组公共的方法，与状态无关，可以共享的，无需子类分别实现；而另一些方法却需要各个子类根据自己特定状态来实现特定功能；

2) 定义一组接口，但不强迫每个实现类都必须实现所有的方法，可用抽象类定义一组方法体可以是空方法体，由子类选择自己感兴趣的方法来覆盖；

7.3、抽象类是否可以没有方法和属性？

可以

7.4、接口的意义

1) 有利于代码的规范，对于大型项目，对一些接口进行定义，可以给开发人员一个清晰的指示，防止开发人员随意命名和代码混乱，影响开发效率。

2) 有利于代码维护和扩展，当前类不能满足要求时，不需要重新设计类，只需要重新写了一个类实现对应的方法。

3) 解耦作用，全局变量的定义，当发生需求变化时，只需改变接口中的值即可。

4) 直接看接口，就可以清楚知道具体实现类间的关系，代码交给别人看，别人也能立马明白。

8、泛型中 extends 和 super 的区别

<? extends T> 限定参数类型的上界，参数类型必须是 T 或 T 的子类型，但对于 List<? extends T>，不能通过 add() 来加入元素，因为不知道 <? extends T> 是 T 的哪一种子类；

<? super T> 限定参数类型的下界，参数类型必须是 T 或 T 的父类型，不能通过 get() 获取元素，因为不知道哪个超类；

9、父类的静态方法能否被子类重写？静态属性和静态方法是否可以被继承？

父类的静态方法和属性不能被子类重写，但子类可以继承父类静态方法和属性，如父类和子类都有同名同参同返回值的静态方法 show()，声明的实例 Father father = new Son(); (Son extends Father)，会调用 father 对象的静态方法。静态是指在编译时就会分配内存且一直存在，跟对象实例无关。

10、进程和线程的区别

进程：具有一定独立功能的程序，是系统进行资源分配和调度运行的基本单位。

线程：进程的一个实体，是 CPU 调度的基本单位，也是进程中执行运算的最小单位，即执行处理机调度的基本单位，如果把进程理解为逻辑上操作系统所完成的任务，线程则表示完成该任务的许多可能的子任务之一。

关系：一个进程可有多个线程，至少一个；一个线程只能属于一个进程。同一进程的所有线

程共享该进程的所有资源。不同进程的线程间要利用消息通信方式实现同步。

区别：进程有独立的地址空间，而多个线程共享内存；进程具有一个独立功能的程序，线程不能独立运行，必须依存于应用程序中；

11、final , finally , finalize 的区别

final：变量、类、方法的修饰符，被 final 修饰的类不能被继承，变量或方法被 final 修饰则不能被修改和重写。

finally：异常处理时提供 finally 块来执行清除操作，不管有没有异常抛出，此处代码都会被执行。如果 try 语句块中包含 return 语句，finally 语句块是在 return 之后运行；

finalize：Object 类中定义的方法，若子类覆盖了 finalize() 方法，在垃圾收集器将对象从内存中清除前，会执行该方法，确定对象是否会被回收。

12、序列化 Serializable 和 Parcelable 的区别

序列化：将一个对象转换成可存储或可传输的状态，序列化后的对象可以在网络上传输，也可以存储到本地，或实现跨进程传输；

为什么要进行序列化：开发过程中，我们需要将对象的引用传给其他 activity 或 fragment 使用时，需要将这些对象放到一个 Intent 或 Bundle 中，再进行传递，而 Intent 或 Bundle 只能识别基本数据类型和被序列化的类型。

Serializable：表示将一个对象转换成可存储或可传输的状态。

Parcelable：与 Serializable 实现的效果相同，也是将一个对象转换成可传输的状态，但它的实现原理是将一个完整的对象进行分解，分解后的每一部分都是 Intent 所支持的数据类型，这样实现传递对象的功能。

Parcelable 实现序列化的重要方法：序列化功能是由 writeToParcel 完成，通过 Parcel 中的 write 方法来完成；反序列化由 CREATOR 完成，内部标明了如何创建序列化对象及数级，通过 Parcel 的 read 方法完成；内容描述功能由 describeContents 方法完成，一般直接返回 0。

区别：Serializable 在序列化时会产生大量临时变量，引起频繁 GC。Serializable 本质上使用了反射，序列化过程慢。Parcelable 不能将数据存储在磁盘上，在外界变化时，它不能很好的保证数据的持续性。

选择原则：若仅在内存中使用，如 activity\service 间传递对象，优先使用 Parcelable，它性能高。若是持久化操作，优先使用 Serializable

注意：静态成员变量属于类，不属于对象，固不会参与序列化的过程；用 transient 关键字编辑的成员变量不会参与序列化过程；可以通过重写 writeObject()和 readObject()方法来重写系统默认的序列化和反序列化。

13、谈谈对 kotlin 的理解

特点：1) 代码量少且代码末尾没有分号；2) 空类型安全（编译期处理了各种 null 情况，避免运行时异常）；3) 函数式的，可使用 lambda 表达式；4) 可扩展方法（可扩展任意类的属性）；5) 互操作性强，可以在一个项目中使用 kotlin 和 java 两种语言混合开发；

14、string 转换成 integer 的方式及原理

- 1) parseInt(String s)内部调用 parseInt(s, 10)默认为 10 进制。
- 2) 正常判断 null\进制范围，length 等。
- 3) 判断第一个字符是否是符号位。
- 4) 循环遍历确定每个字符的十进制值。
- 5) 通过*=和-=进行计算拼接。
- 6) 判断是否为负值返回结果。