目录

1. 基础语法

1.1 面向对象和面向过程的区别

面向过程
优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源;比如单片机、嵌入式开发、Linux/Unix 等一般采用面向过程开发，性能是最重要的因素。
缺点：没有面向对象易维护、易复用、易扩展
面向对象
优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护
缺点：性能比面向过程低

1.2 Java 面向对象编程三大特性:封装、继承、多态

封装
封装把一个对象的属性私有化，同时提供一些可以被外界访问的属性的方法，如果属性不想被外界访问，我们大可不必提供方法给外界访问。但是如果一个类没有提供给外界访问的方法，那么这个类也没有什么意义了。

继承
继承是使用已存在的类的定义作为基础建立新类的技术,新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。通过使用继承我们能够非常方便地复用以前的代码。
关于继承如下 3 点请记住：
1.子类拥有父类非 private 的属性和方法。
2.子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
3.子类可以用自己的方式实现父类的方法。（重写）。

多态
所谓多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。
在 Java 中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口（实现接口并覆盖接口中同一方法）

1.3 接口和抽象类有什么区别

你选择使用接口和抽象类的依据是什么？
接口和抽象类的概念不一样。接口是对动作的抽象，抽象类是对根源的抽象。

抽象类表示的是，这个对象是什么。接口表示的是，这个对象能做什么。比如，男人，女人，这两个类（如果是类的话……），他们的抽象类是人。说明，他们都是人。
人可以吃东西，狗也可以吃东西，你可以把“吃东西”定义成一个接口，然后让这些类去实现它.
所以,在高级语言上,一个类只能继承一个类(抽象类)正如人不可能同时是生物和非生物),但是可以实现多个接口(吃饭接口、走路接口)。
总结几句话来说:
1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。
2、抽象类要被子类继承，接口要被类实现。
3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现
4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。
5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。
6、抽象方法只能申明，不能实现，接口是设计的结果 ，抽象类是重构的结果
7、抽象类里可以没有抽象方法
8、如果一个类里有抽象方法，那么这个类只能是抽象类
9、抽象方法要被实现，所以不能是静态的，也不能是私有的。
10、接口可继承接口，并可多继承接口，但类只能单根继承。
总结:
1、抽象类和接口都是用来抽象具体对象的. 但是接口的抽象级别最高
2、抽象类可以有具体的方法 和属性，接口只能有抽象方法和不可变常量
3、抽象类主要用来抽象类别,接口主要用来抽象功能.
4、抽象类中，且不包含任何实现，派生类必须覆盖它们。接口中所有方法都必须是未实现的。
当你关注一个事物的本质的时候，用抽象类；当你关注一个操作的时候，用接口。 抽象类的功能要远超过接口，但是，定义抽象类的代价高。因为高级语言来说（从实际设计上来说也是）每个类只能继承一个类。在这个类中，你必须继承或编写出其所有子类的所有共性。虽然接口在功能上会弱化许多，但是它只是针对一个动作的描述。而且你可以在一个类中同时实现多个接口。在设计阶段会降低难度的。

1.4 字符型常量和字符串常量的区别

1.形式上: 字符常量是单引号引起的一个字符 字符串常量是双引号引起的若干个字符
2.含义上: 字符常量相当于一个整形值(ASCII 值),可以参加表达式运算,字符串常量代表一个地址值(该字符串在内存中存放位置)
3.占内存大小:字符常量只占 2 个字节,字符串常量占若干个字节(至少一个字符结束标志)
(注意： char 在 Java 中占两个字节)

Java要确定每种基本类型所占存储空间的大小。它们的大小并不像其他大多数语言那样随机器硬件架构的变化而变化。这种所占存储空间大小的不变性是Java程序比用其他大多数语言编写的程序更具可移植性的原因之一。

基本类型	大小	最小值	最大值	包装器类型
boolean	—	—	—	Boolean
char	16-bit	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	—	—	—	Void

1.5 重载和重写的区别

重载：发生在同一个类中，方法名必须相同，参数类型不同、个数不同、顺序不同，方法返回值和访问修饰符可以不同，发生在编译时。

重写：发生在父子类中，方法名、参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类，访问修饰符范围大于等于父类；如果父类方法访问修饰符为 private 则子类就不能重写该方法。

注意：父类的私有属性和构造方法并不能被继承，所以Constructor 构造方法不能被 override(重写)，但是可以 overload(重载)，所以一个类里可以有多个构造函数，无参构造，有参数构造等等。

1.6 String类

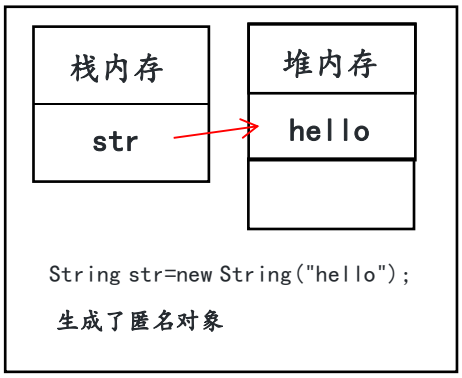
String 类代表字符串，所有的字符串都是这个类的实例实现，字符串是常量，在创建之后值不可变。
String 类中使用 final 关键字字符数组保存字符串，private final char value[]，所以 String 对象是不可变的。

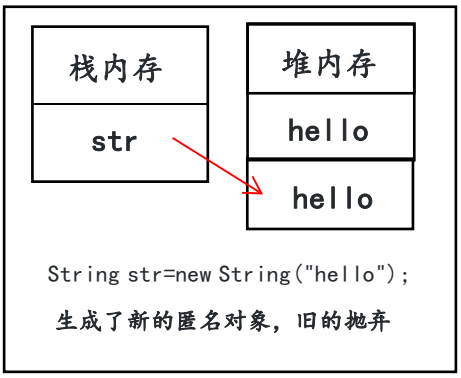
1.6.1 两种创建方式：

1. String str="hello";//直接赋值的方式
2. String str=new String("hello");//new 关键字实例化的方式

在字符串中，如果采用***直接赋值***的方式进行对象的实例化，直接赋值会先检查常量池内有没有对应的字符串，有的话就直接指向，则会将“hello”放入常量，每当下一次对不同的对象进行直接赋值的时候会直接利用池中原有的匿名对象。

采用**构造实例化**的方式的时候，会像堆申请内存，来存储字符串，str 在栈内存中指向堆内存中的匿名对象“hello”，当我们再一次的 new 一个同样内容的 String 对象时，原来的匿名对象将被，会产生新的匿名对象“hello”，原来的将被抛弃变成垃圾回收，字符串对象存在于堆空间中，与字符串常量池无关。





两种实例化的总结：

1）直接赋值（String str = "hello"）：只开辟一块堆内存空间，并且会自动入池，不会产生垃圾。

2）构造方法（String str= new String("hello");）:会开辟两块堆内存空间，其中一块堆内存会变成垃圾被系统回收，而且不能够自动入池，需要通过 public String intern();方法进行手工入池。在开发的过程中不会采用构造方法进行字符串的实例化。

例题：

String s1="hello"+"world";

String s2="helloworld";

其中 s1==s2 返回的是 true。字符串相加的时候，如果相加的是静态字符串，则会去常量池中寻找对应的存在直接指向。

1.6.2 字符串的比较：

== 和 public boolean equals()比较字符串的区别：

==在对字符串比较的时候，对比的是**内存地址**，而 equals 比较的是**字符串内容**在开发的过程中，equals()通过接受参数，可以避免空指向。

举例:

```
String str = null;
if(str.equals("hello")){//此时会出现空指向异常
    ...
}
if("hello".equals(str)){//此时 equals 会处理 null 值，可以避免空指向异常
    ...
}
```

1.6.3 字符串的不可变：

字符串从创建开始就不可以去变了，变化的的最多只是地址，内容是不会变的。

在设计的时候，被设计成**享元模式**，每当生成一个新的字符串的是，就会被放进共享池中，第二次生成同样的内容字符串的时候，则会共享这个对象，这些操作只限于**直接赋值==**。

在 object 中，equals()是用来比较内存地址的，但是 String **重写了 equals()方法，用来比较内容的，即使是不同地址，只要内容一致，也会返回 true。**

不可变的好处：
避免多次创建的开销
2. 出于安全性考虑，字符串在程序用应用太多了。
3. 例如 HashMap 中可以用 String 做 key，可变的话就会有问题
4. 在传参的时候，使用不可变类不需要去考虑谁可能会修改其内部的值，如果使用可变类的话，可能需要每次记得重新拷贝出里面的值，性能会有一定的损失

举例：
使用 String s = new String("hello");会创建几个对象
会创建 2 个对象
首先，出现了字面量"hello"，那么去 String Pool 中查找是否有相同字符串存在，因为程序就这一行代码所以肯定没有，那么就在 Java Heap 中用字面量"hello"首先创建 1 个 String 对象。接着，new String("hello")，关键字 new 又在 Java Heap 中创建了 1 个对象，然后调用接收 String 参数的构造器进行了初始化。最终 s 的引用是这个 String 对象.

1.7 String,StringBuffer 与 StringBuilder 的区别

1.7.1 可变与不可变

String 类中用字符数组保存字符，采用 final 修饰，所以 **String 对象不可变**。
StringBuffer 与 StringBuilder 继承于 AbstractStringBuilder，AbstractStringBuilder 里面也是采用字符数组，但是没有 final 修饰，所以是**可变的**。

1.7.2 运行速度

由于 String 类是不可变的，每次重新赋值也就是创建新的对象，原来的被回收，所以比较慢。
而 StringBuilder 和 StringBuffer 的对象是变量，对变量进行操作就是直接对该对象进行更改，而不进行创建和回收的操作，所以速度要比 String 快很多。

1.7.3 线程安全

String 是不可变的，显然是线程安全的。
StringBuilder 和 StringBuffer，StringBuffer 内部对字符串的操作都采用了 synchronized 关键字，所以是线程安全的；StringBuilder 是非线程安全的。
由于，StringBuilder 非线程安全，所以加锁释放锁的操作就没有，就相对于 StringBuffer 更快。

1.7.4 总结

String：适用于少量的字符串操作的情况

StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况

StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况