Java面向对象(中级)

Java面向对象(初级)

#一、知识补充

包package

包的本质就是创建不同的文件夹/目录来保存类文件
包的命名规范：包命名的时候只能包含数字，字母，下划线，英文句号，不能是数字开头，不能包含关键字和保留字
如何导入包

```
import 包名.方法;//导入包中特定方法
import 包名.*;//导入全部
```

package的作用是声明当前类所在的包，需要放在类的最上面，一个类中最多只有一句package
import 指令位置放在package的下面，在类定义的前面，可以有多句并且没有顺序

访问修饰符

公开级别public 对外公开
受保护级别protected 向子类和同一个包公开
默认级别(无修饰符) 向同一个包的类公开
私有级别private 只有类本身可以访问，不对外公开

访问级别	访问控制修饰符	同类	同包	子类	不同包
公开	public	yes	yes	yes	yes
受保护	protected	yes	yes	yes	no
默认	没有修饰符	yes	yes	no	no
私有	private	yes	no	no	no

1. 同一个包下可以访问公开、受保护以及默认的属性和方法，不能访问私有的属性和方法。
2. 在不同包下，只能访问公开级别的属性和方法
3. 访问修饰符可以用来修饰类中的属性，成员方法以及类
4. 只有默认和public才可以修饰类，并且遵循权限特点
5. 成员方法的访问规则和属性一样

#二、面向对象基本特征

封装encapsulation

把抽象出的数据的属性和对数据的操作方法封装在一起，数据被保护在内部，程序的其他部分只有通过被授权的方法才能对数据进行操作
封装三部曲：

1. 将属性进行私有化private
2. 提供一个公共的(public) set方法，用于对属性进行判断并且赋值
3. 提供一个公共的(public)get方法，用于获取属性的值

继承extends

当不同的类中有相似的属性和方法的时候可以从这些类中抽象出来父类，在父类中定义这些相同的属性和方法，所有子类不需要重复定义这些重复的属性和方法，只需要使用extends来声明继承即可

```
class 子类 extends 父类{
子类独有的属性和方法
}
```

子类会自动继承父类的属性和方法

1. 子类继承了所有的属性和方法，非私有的属性和方法可以直接在子类中访问，但是私有的属性和方法不能在子类中直接进行访问，需要使用公共的方法去访问
2. 子类必须调用父类的构造器完成父类的初始化。子类与父类均有无参构造器则初始化的时候均会被调用。如果父类有多个构造器，则必须在子类的构造器中使用super去指定使用父类的哪一个构造器来完成父类的初始化的工作，否则编译不会通过。
3. super使用的时候必须放在子类构造器语句的第一行
4. 如果没有this，子类构造器第一行默认含有super()
5. super和this都只能在构造器的第一行，所以使用super或者使用this
6. Object是所有类的子类
7. 子类只能继承一个父类
8. 不可滥用继承，子类和父类必须满足is-a的逻辑关系，即子类必须在逻辑上属于父类的一种
9. 子类属性(方法)与父类重名时候，使用该属性的时候如果子类有这个属性并可以访问则访问，如果父类有这个属性并可以访问则访问，如果父类没有则继续寻找，直到查找到顶部父类Object，遇到目标属性(方法)但是私有则直接报错，不会继续向上查找

super关键字

```
this.属性/方法();//从本类开始查找
super.属性/方法();//从父类开始查找
```

super代表父类的引用，用于访问父类的属性，方法，构造器

1. super可以访问父类的属性和方法，但是不能访问父类的私有private属性和方法
2. 当子类中有和父类的成员重名的时候，为了访问父类的成员，必须通过super，如果没有重名则可以直接调用，调用的时候向上查找
3. super的访问可以越级访问，如果爷爷类中有与本类同名的成员，也可以使用super去访问爷爷类的成员

方法重写/覆盖override

方法重写/覆盖就是子类有一个方法与父类的某个方法的名称、返回类型与参数**完全一样**，那么说子类这个方法覆盖了父类的方法

1. 如果方法返回类型与父类一样，或者父类的返回类型与子类的返回类型构成父子关系则可以实现重写
2. 子类重写的时候不能缩小父类的访问权限

方法重写与重载的区别

名称	发生范围	方法名	参数列表	返回类型	修饰符
重载	本类	必须一样	类型个数顺序至少一个不同	无要求	无要求
重写	父子类	必须一样	相同	子类重写方法返回的类型必须与子类对应方法的返回类型相同，或者是父子关系	子类方法不能缩小父类的访问范围

多态

多态顾名思义即多种形态，同一个方法或操作在不同对象上表现出的不同的行为。
多态的体现

1. 方法多态
   (1)重载体现多态(2)重写体现多态
2. 对象多态
   (1) 对象的编译类型与运行类型可以不一致，编译类型在定义的时候就已经确定
   (2)对象的运行类型是可以变化的，可以通过getClass()来查看运行类型
   (3)定义对象的时候编译类型在等号的左边，运行类型在等号的右边
   父类的引用指向了子类的对象

```
父类类型 引用名=new 子类类型()
```

1. 一个对象的编译类型和运行类型可以不一致
2. 编译类型在定义对象的时候就确定了不能改变
3. 运行类型是可以变化的
4. 编译类型看定义时候=的左边，运行类型看=的右边
   可以调用父类的所有成员，但是需要遵循访问权限
   最终运行效果看子类的具体表现
5. 使用方法的时候参数可以接受编译类型子类对象
   向上转型：父类的引用指向了子类的对象，可以调用父类的所有成员，但是不能调用子类的所有成员
   编译的时候看编译类型，不能调用子类特有的
   运行的时候看运行类型，仍然是从运行类型向上查找
   总之只能调用父类中的方法或者子类中与父类重名的方法
   向下转型：

```
子类类型 引用名=(子类类型)父类引用
```

只能强转父类的引用，不能强转父类的对象
要求父类引用必须指向当前目标类型的对象
可以调用子类类型中的所有成员
instanceof：比较操作符，用于判断对象的运行类型是否为XX类型或者XX类型的子类
属性并没有重写之分，重写的值看的是编译类型
动态绑定机制：调用方法的时候，该方法会与该对象的内存地址/运行类型绑定，当调用属性的时候没有动态绑定机制，在哪里声明就在哪里使用
多态数组：数组的定义类型是父类类型，里面保存的类型是子类类型，编译类型是父类运行类型根据实际情况，可能是父类也可能是其中某一子类
使用数组名[i]调用子类的方法时候无法调用，这里可以使用instanceof判断并且向下转型，将其编译类型强转为对应的子类的类型
多态参数，形参类型为父类类型，实际调用的时候允许为子类类型

根类Object

Object是任何一个类的父类，任何一个类都可以使用Object类的方法

方法	介绍
clone	创建并返回该对象的一个副本
equals	判断两个对象是否对等，只能判断引用类型；==既可以判断值也可以判断引用是否相同
getClass	返回运行类型
hashCode	返回对象的哈希码值
toString	返回该对象的字符串表示

方法	介绍
finalize	当垃圾处理器确定不存在对该对象的更多引用的时候，由对象的垃圾回收调用此方法

hashCode

1. 提高具有哈希结构的容器的效率
2. 两个引用如果指向同一个对象，则哈希值一定一样
3. 两个引用如果指向不用对象，则哈希值不一样
4. 哈希值是根据地址号来的，但是并不完全等价于地址
   toString
   默认返回全类名+@+哈希值的十六进制

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

一般来说要想输出对象的详细信息会对这个方法进行重写
直接输出一个对象的时候toString会被默认调用
finalize
当对象被回收的时候，系统自动调用该对象的finalize方法，子类可以重写该方法，做一些释放资源的操作
当某个对象没有任何引用的时候，jvm就会认为该对象是一个垃圾对象，就会使用垃圾回收机制来销毁该对象，在销毁该对象之前会调用finalize方法
垃圾回收机制的调用是由系统决定的(即有自己的GC算法)，也可通过System.gc()主动触发回收机制