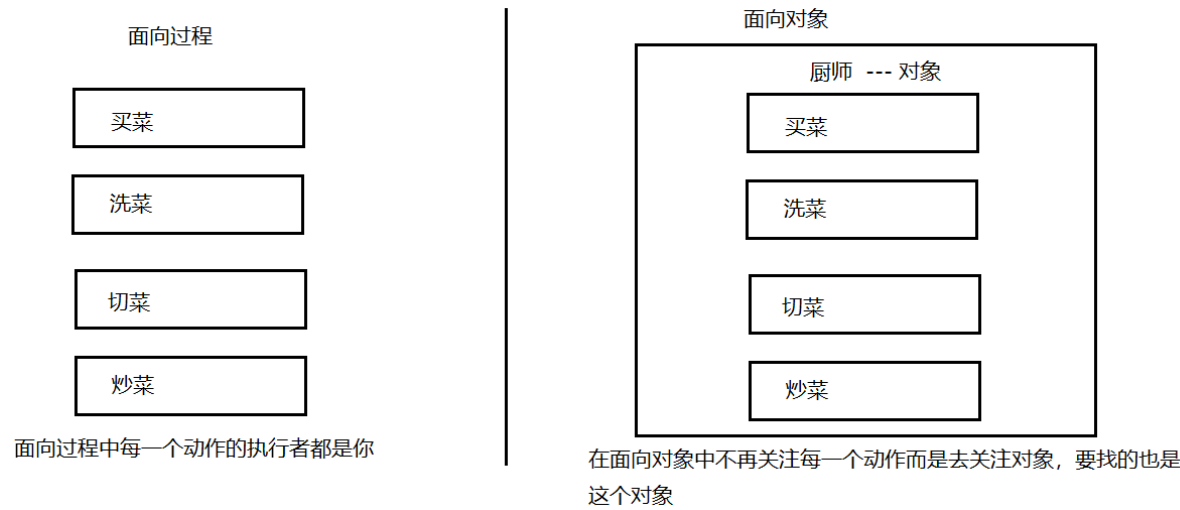


# 面向对象简介

## 一、面向对象与面向过程的比较

1. 面向对象是相对于面向过程而言的。面向对象和面向过程都是生活中的思维方式



- 2. 面向过程注重于过程，熟悉过程中的每一个细节 - 自己动手就是面向过程
- 3. 面向对象则是着重于对象，只要找到了对象，则自然的拥有了对象身上的一切功能 - 找别人代做就是面向对象
- 4. 面向对象是基于面向过程的
- 5. 面向对象不一定优于面向过程
  - a. 在一些相对复杂的场景下，面向对象能够更好的做到分工合作从而提高效率，所以此时建议使用面向对象
  - b. 在一些相对简单的场景下，面向过程反而比面向对象能够更快的完成，所以此时建议使用面向过程

## 二、类与对象的关系

1. 对象是现实生活中存在的事物。而对某一类事物进行抽象总结，将这些事物共有的特征抽取成变量，称之为是属性或者是成员变量，将这些事物共有的行为定义为方法，称之为成员方法，而定义一个类表示这一类对象，那么就抽取出一个代表这些对象的类。所以简而言之，类是对一类对象的概括抽取

练习：定义一个类表示学生，这个学生包含姓名、年龄、性别、班级、学号属性，包含学习、吃饭、玩耍行为

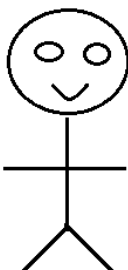
人

特征：姓名、年龄、性别、身高、体重...

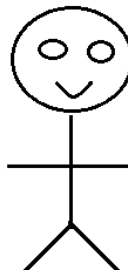
行为：吃饭、喝水、睡觉...

```
class Person {  
    String name;  
    int age;  
    byte gender;  
    double height;  
    double weight;  
    public void eat(String food){}  
    public void drink(){}  
    public void sleep(){}  
}
```

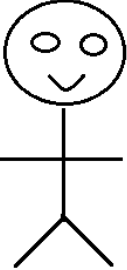
属性/成员变量



姓名：翠花  
年龄：30  
性别：女  
身高：150  
体重：150  
...  
吃饭、喝水、睡觉...



姓名：如花  
年龄：18  
性别：男  
身高：180  
体重：90  
...  
吃饭、喝水、睡觉...



姓名：桃花  
年龄：70  
性别：女  
身高：170  
体重：100  
...  
吃饭、喝水、睡觉...

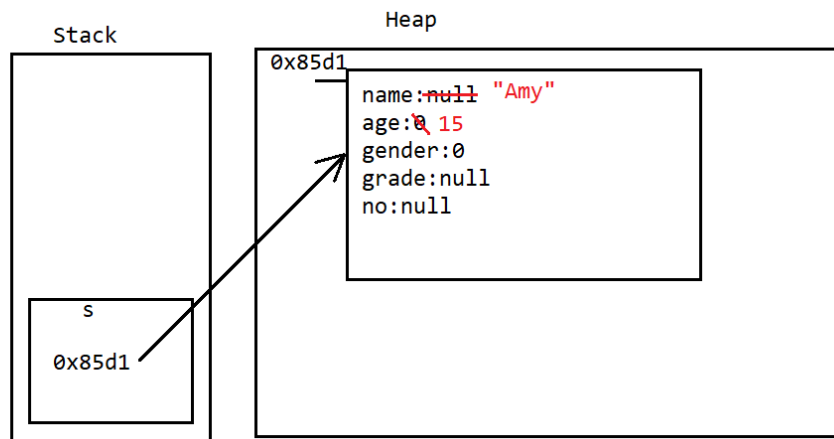
2. 产生了类之后，可以利用new关键字来产生一个实例对象。new在java中是一个用于创建实例对象的关键字

练习：定义一个类表示车，然后创建一个车对象，这个车有5个座位，车的颜色是红色的

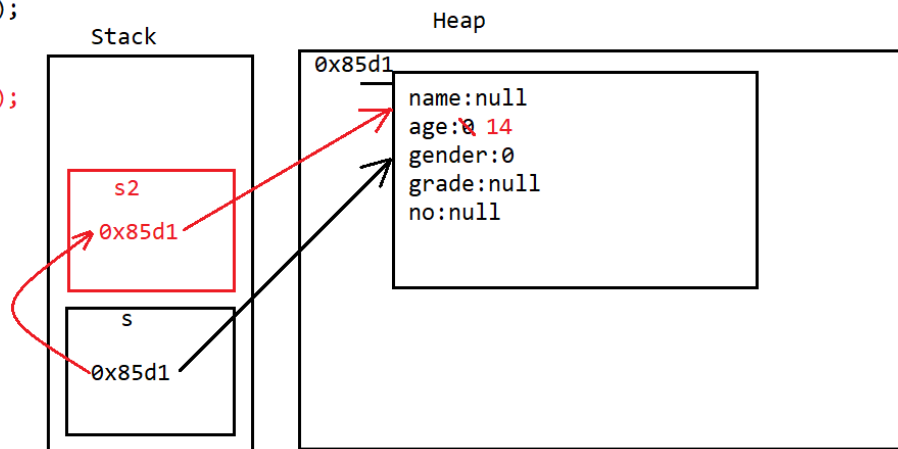
### 三、对象的内存存储

1. 对象在创建的时候是在栈内存中存储地址引用，在堆内存中存储实际对象
2. 对象在内存中存储的时候，各项属性都会进行赋值。byte/short/int类型赋值为0，long类型赋值为0L，float类型赋值为0.0f，double类型赋值为0.0，char类型赋值为'\u000'，boolean类型赋值为false，其他类型（引用类型）赋值为null
3. 对象在传值的时候传递的也是地址

```
Student s = new Student();  
s.name = "Amy";  
s.age = 15;
```



```
Student s = new Student();  
Student s2 = s;  
s2.age = 14;  
System.out.println(s.age);  
14
```



基本概念

一、成员变量和局部变量

区别	成员变量	局部变量
定义位置	定义在类中，成员变量又叫属性	定义在方法或者语句中
作用范围	作用在整个类中	作用在对应的方法或者语句中
内存位置	存储在堆内存中，并且在堆内存中被赋予了默认值	存储在栈内存中，而且没有默认值需要手动给定数据
生命周期	在对象创建的时候创建，在对象被销毁的时候销毁	在方法或者语句执行的时候创建，方法或者语句执行完成之后销毁

二、匿名对象

- 1. 所谓匿名对象是指没有名字的对象。
- 2. 注意事项
  - a. 因为匿名对象没有名字，所以只能在创建的时候使用一次
  - b. 匿名对象可以作为参数进行传递
  - c. 匿名对象如果没有作为参数传递，那么在栈内存中是没有引用的

三、构造方法

- 1. 所谓构造方法是指在类中存在的方法名与类名一致而没有返回值类型的方法
- 2. 作用：
  - a. 构造方法最主要的作用是用于创建对象
  - b. 可以在构造方法中对属性进行初始化，或者执行其他一些初始化的操作
- 3. 如果在类中没有手动给定构造方法，那么在编译的时候会自动添加一个默认的空参构造
- 4. 如果一个类中手动添加了构造方法，那么在编译的时候就不会再添加空参的构造方法
- 5. 构造方法虽然没有返回值类型但是可以有return语句，其作用是避免一些不合常理的数据被赋值给属性
- 6. 构造方法可以进行重载

四、this关键字

- 1. 因为在java中所有的非静态的方法和属性都是通过对象来调用的，而在本类中如果想调用本类中的方法和属性，并没有本类的对象，所以要用到this来代表当前类的对象，来调用属性和方法
- 2. this代表本类在活动的对象的引用，可以认为是一个虚拟对象，用于在类内调用本类中的非静态方法和非静态属性
- 3. this语句表示在本类的构造方法中调用本类其他形式的构造方法。需要注意的是，this语句必须放在构造方法的第一行

五、代码块

- 1. 构造代码块：
  - a. 所谓构造代码块是指定义在类内用{}包起来的代码，也称之为是初始化代码块
  - b. 无论调用哪个构造方法，构造代码块都会执行
  - c. 构造代码块是在创建对象的时候先于构造方法执行
- 2. 局部代码块：
  - a. 所谓局部代码块，是指定义在方法中用{}包起来的代码
  - b. 作用是限制变量的生命周期从而提高栈内存的利用率

六、权限修饰符

- 1. 使用范围

关键字	本类中	子类中	同包类中	其他类中
public	可以使用	可以使用	可以使用	可以使用
protected	可以使用	可以使用	可以使用	不可以使用
默认	可以使用	同包子类中可以使用	可以使用	不可以使用
private	可以使用	不可以使用	不可以使用	不可以使用

## 2. 注意事项:

- 限修饰符的范围是public > protected > 默认 > private
- 需要注意的是，默认的权限修饰符只能在本类中以及同包类中使用，同包子类本质上也是同包类
- protected在子类中使用指的是在对应的子类中使用，跨子类是不能使用的

# 面向对象特征

## 一、封装

1. 封装是面向对象方法的重要原则，就是把对象的属性和方法（或服务）结合为一个独立的整体，并尽可能隐藏对象的内部实现细节。封装是一种信息隐藏技术
2. 形式
  - a. 方法。方法其实是封装的一种常见的形式，通过将某段常用的逻辑进行提取从而形成一种新的形式
  - b. 属性的私有化。在类中定义了属性之后，如果允许在类外直接操作属性会导致用户直接给属性赋予一些不合常理的数据。将属性私有化，将属性用`private`修饰，然后提供对外的访问(`getXXX`)和设置(`setXXX`)的方法，在方法中进行限定，使属性值更加符合的场景要求
  - c. 内部类。内部类作为封装的一种形式，是为了让代码的结构更加的紧凑
3. 作用
  - a. 提高了复用性（降低了代码的冗余度）、安全性、使代码结构更加紧密
  - b. 使属性私有化-隐藏信息，实现细节
  - c. 使属性值更符合要求-可以对成员进行更精确的控制
  - d. 提高了代码的安全性-类内部的结构可以自由修改
  - e. 良好的封装能够减少耦合

## 二、继承

1. 如果一些类中的属性和方法是相同的，那么可以把这些类中相同的属性和方法提取到一个新的类中，然后利用`extends`关键字让原来的类和新的类产生联系，这种联系称之为继承。而这个时候原来的类称之为是子类，新的类称之为父类
2. 注意事项
  - a. Java中支持的是类和类之间的**单继承**。即一个子类只能有一个父类，但是一个父类可以有多个子类
  - b. 通过继承，子类可以继承父类全部的数据域，但是只有一部分数据域对子类可以见，所以子类也只能使用这一部分可见的数据域
3. 单继承与多继承的比较
  - a. 多继承在代码的复用性上要优于单继承，但是存在方法调用的混乱
  - b. 单继承也可以提高代码的复用性，可以避免方法调用的混乱，提高了方法调用的安全性
4. 特征
  - a. 继承关系是传递的
  - b. 继承简化了人们对事物的认识和描述，能清晰的体现相关类间的层次结构关系
  - c. 继承提高了代码的复用性
  - d. 继承通过增强一致性来减少模块间的接口和界面，大大增加了程序的易维护性
5. 优点
  - a. 提高复用性
  - b. 提高安全性，避免方法调用产生混乱
  - c. 统一结构
6. 方法的重写
  - a. 子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖

- b. 在父子类里存在了方法签名（方法名+参数列表）完全一致的的非静态方法，就构成了方法的重写（覆盖）
- c. 遵循规则：方法的重写需要遵循“两等两小一大”的原则：
  - i. 方法签名要求完全一致
  - ii. 如果父类里的返回值是基本数据类型/void/最终类，那么子类里重写的返回值类型必须一致（如果父类的方法是private的话，子类是无法被继承的，即使子类重新定义这个方法，也不算重写）
  - iii. 如果父类方法的返回值类型是一个引用类型，子类方法的返回值类型是父类方法的返回值类型的子类

```
class Pet{}  
class Dog extends Pet{}
```

```
class A {  
    public Pet m(){}  
}
```

```
class B extends A {  
    // public Pet m(){}  
    public Dog m(){}  
}
```

- iv. 子类方法的权限修饰符的范围要大于等于父类方法权限修饰符的范围。需要注意的是，对于8种基本数据类型之间是没有继承关系的，它们是同级关系

```
class A {  
    protected void m(){}  
}  
class B extends A {  
    // protected void m(){}  
    public void m(){}  
}
```

## 7. super关键字

- a. super表示在子类中对父类对象的引用，可以看作是一个虚拟对象
- b. 在子类中可以通过super调用父类中的非静态方法或者非静态属性
- c. 在子类的构造方法中，可以通过super关键字调用父类中的对应形式的构造方法。如果不指定，默认调用父类的无参构造
- d. 如果父类中只提供了含参的构造方法，那么子类的构造方法中必须手动定义super语句来调用父类中的含参构造
- e. super语句必须放在构造方法的第一行，所以super语句和this语句不能同时出现

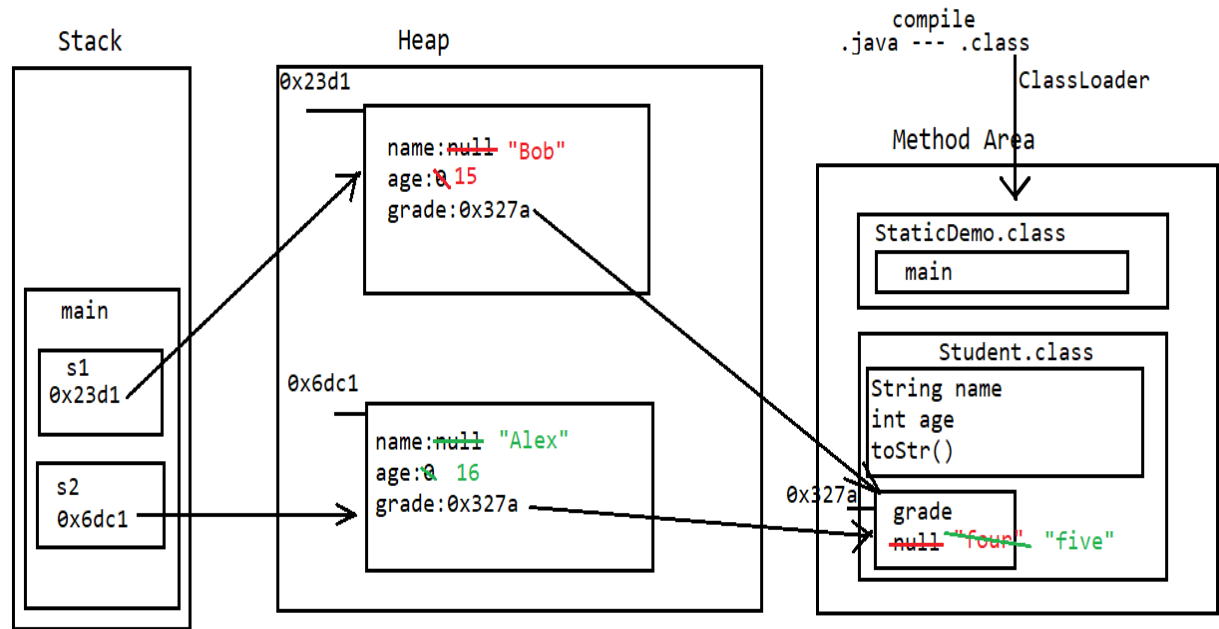
## 三、多态

- 1. 多态是指允许不同类的对象对同一消息做出响应。即同一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。多态主要针对对象的行为即方法，而不是对象的属性
- 2. 体现形式：
  - a. 编译时多态：在编译时期就要进行动态绑定的行为，主要体现为方法的重载
  - b. 运行时多态：
    - i. 在运行时期才能确定绑定的运行过程，主要体现形式是向上造型、方法的重写
    - ii. 需要注意的是运行时多态的前提的产生了继承关系或者实现关系
- 3. 向上造型：指的是用父类声明对象而用子类创建对象的行为方式。利用向上造型创建的对象所能使用的属性或者方法看的是声明类，而具体怎么执行所调用的方法看的是实现类
- 4. 作用：统一结构，实现解耦

# 基本概念（二）

## 一、static

- 1. static本身是一个修饰符，可以修饰变量、方法、内部类、代码块
- 2. 静态是从类的层面来看的，已经超越了对象
- 3. 静态变量：用static修饰变量称之为静态变量，又称为类变量
  - a. 静态变量随着类的加载而加载到了方法区，并且在方法区中自动赋予了一个默认值。静态变量先于对象而存在，所以静态变量可以通过类名来调用，也可以通过对象来调用，实际过程中一般是通过类名来调用静态属性。该类产生的所有的对象实际上存的是该静态变量在静态区中的地址，静态变量是被所有对象所共享的
  - b. 实际过程中，如果某些特征是所有属性共有的特征，例如一个班级的学生，那么这个时候班级这个属性就是所有学生共有的，那么此时这个属性就可以设置为静态的
  - c. 静态内存流程



- 4. 静态方法：static修饰方法就叫静态方法，也叫类方法
  - a. 在类加载的时候加载到了方法区，只是存储在方法区此时并没有执行，而是在方法被调用的时候到栈内存中执行。所有的静态元素不归属于某一个对象而是归属于类，所以所有的静态元素都是先于对象存在，因此静态方法也是先于对象而存在的，所以静态方法可以通过类名来调用，也可以通过对象来调用，实际过程中静态方法一般是通过类名来调用。例如： `Arrays.sort()`， `Arrays` 是一个类， `sort` 是一个方法，通过类名来调用
  - b. 静态变量不可以定义到静态方法中。静态方法在方法区中只存储不执行，当被调用的时候才在栈中执行。而静态变量是根据类的加载而加载，上来就要进行初始化 - 所有的静态元素都是定义在类中
  - c. 静态方法中不可以直接调用本类中的非静态方法。在Java中所有非静态方法和非静态属性都是通过对象来调用的，静态方法是优先于对象存在的，也就意味着静态方法执行的时候可以没有对象。也因此静态方法中不可以使用 `this` 和 `super`。 `super` 与 `this` 都是和对象是相关联的，而静态先于对象而存在
  - d. 静态方法可以重载。重载对修饰符没有要求的，对返回值也没有要求
  - e. 静态方法不可以重写（方法的覆盖）：静态方法可以存在方法签名完全一致的静态方法，这不是重写，称之为隐藏。但是也适用于重写的那套规则
  - f. 注意：如果父子类中存在方法签名相同的方法要么都是普通的方法，要么都是静态方法
- 5. 静态代码块
  - a. 在类中用static修饰用{}括起来的代码块



- b. 静态代码块针对的是类，所以也可以叫做类代码块
- c. 实际上静态代码块是随着类的加载而加载到方法区，在类创建对象或者执行方法之前执行一次，终其一生只执行一次
- d. 执行顺序：在这个类第一次被真正使用（第一次创建对象/调用方法）的时候执行一次。如果一个类包含多个静态代码块，则按照书写顺序执行。由于类只在第一次使用的时候加载，所以静态代码块也只执行一次。代码执行顺序：先父类后子类，先静态后动态。（先父子类的静态，后父子类的动态）静态优先，父类优先
- e. 代码是从上到下，从左到右依次编译执行：创建子类对象的时候需要先创建父类对象→加载父类→执行父类静态代码块→执行子类静态代码块→父类构造代码块→父类构造函数→子类构造代码块→子类构造函数

```

class A {
    // 静态代码块
    static {
        System.out.println("A 1");
    }
    {
        System.out.println("A 2");
    }
    public A(){
        System.out.println("A 3" );
    }
}

class B extends A {
    static {
        System.out.println("B 1");
    }
    {
        System.out.println("B 2");
    }
    public B(){
        System.out.println("B 3");
    }
}

new B();

```

试图B类，但是准备加载B类的时候发现B有父类A，所以先加载A类，执行A中的静态代码块，A类加载完成之后再加载B类，执行B中的静态代码块。试图创建B对象的时候，会先利用super语句创建一个A类对象，然后再创建B类对象

## 二、final

1. 常量：当final修饰数据（基本类型和引用类型）的时候，表示这个变量的值不可变，称之为常量。终其一生只能赋值一次。在Java中所说的常量往往是指静态常量。因为实质上只有静态常量才是独有的一个
  - a. 特点：
    - i. 常量在定义好之后不可改变，final固定的是栈内存中的数值
    - ii. 常量可以作为参数传递，传递之后是否还是一个常量要看接收的方法中是否定义为一个常量
    - iii. 对引用类型而言，final固定的是其在栈中的地址不可变。例如：数组在栈内存中存储的是地址，用final修饰，是不能改变数组的地址，但数组的值可以改变。对于对象而言，对象的引用不能改变，但是引用的属性值是可以进行改变的
    - iv. 成员常量只要是在对象创建完成之前（构造方法/函数执行结束之前）赋初始值即可
    - v. 静态成员常量（static final）只要在类加载完成之前给值即可，而且只能在静态代码块中赋值
2. 最终方法
  - a. final修饰方法的时候，这个方法就是最终方法
  - b. 特点：
    - i. 最终方法不可以被重写也不能被隐藏，可以重载，可以被继承
    - ii. 静态方法可以被final修饰
3. 最终类



- a. final修饰的类称之为最终类
- b. 特点
  - i. 最终类不可以被继承，也不能有匿名内部类形式。（匿名内部类后续详细说）
  - ii. 由于最终类不能被继承，因此重写也是不可以的

### 三、abstract

#### 1. 抽象类

- a. 将一些名称一致但是细节不同的行为提取到父类中定义为抽象方法，抽象方法所在的类就是抽象类，用abstract来修饰的类
- b. 抽象类中，不一定含有抽象方法，但是抽象方法所在的类一定是抽象类
- c. 抽象类不可以在Java中创建对象/实例化。即使没有抽象方法也无法创建对象，可以创建匿名内部类
- d. 抽象类被子类继承之后，必须重写其中的抽象方法，除非子类也是抽象类
- e. 抽象类中可以没有抽象方法
- f. 抽象类中可以定义一切的属性和方法
- g. 抽象类不能用final修饰。最终类不可以是抽象类

#### 2. 抽象方法：

- a. 如果所有的子类中存在了一些名称一致而细节不同的方法的时候，这个时候可以在父类中声明该行为，此时声明行为的时候不需要添加方法体，所以此时该方法就形成了抽象方法，使用abstract修饰
- b. 简单来讲：就是在父子类进行继承的时候，子类重写父类的方法但是父类的方法在后续并不打算使用，因此就会将父类的方法的方法体删除声明抽象方法，子类直接重写即可
- c. 抽象方法可以和抽象方法重载，也可以和实体方法重载
- d. 抽象方法没有方法体
- e. 抽象方法不可以被static、final、private修饰，因为final和private修饰符修饰的方法都不可以被重写；static修饰的方法，先于对象存在，没有具体对象没办法加载
- f. 抽象方法可以使用默认权限修饰，要求子类必须和父类同包
- g. 抽象方法可以被protected权限修饰，要求要么同包要么是子类

### 四、接口interface

- 1. 接口用interface来声明，其中所有方法都为抽象方法。但是从JDK1.8开始，接口中允许存在实体方法
- 2. 通过implements关键字让接口和类产生联系，这个过程叫实现
- 3. 利用接口的向上造型来创建对象，就是接口的多态
- 4. 接口中方法的abstract关键字可以省略
- 5. 类实现接口的时候必须实现这个接口中所有的方法
- 6. 由于接口中都是抽象方法，所以接口不能实例化
- 7. 接口中没有构造函数
- 8. 虽然接口在编译完成之后会产生class文件，但是接口不是类
- 9. 接口中可以定义属性，这个属性默认是一个静态常量即接口中的属性默认是用public static final来修饰
- 10. 接口中的抽象方法默认用public abstract修饰，而且只能是public修饰的，public可以省略不写。在接口的子类中实现接口的方法记得用public修饰

11. Java中类支持单继承，多实现。一个类只能继承一个类，但是一个类可以实现多个接口。一旦出现了多实现，那就必不可免的会导致方法调用混乱 - 类和类之间是单继承，类和接口之间是多实现，接口和接口之间是多继承
12. 注意：Java中接口之间是多继承，并且接口和类之间是多实现的关系，所以就形成了一张继承关系网，由于在网状结构中寻找一个根节点比较困难，为了提高效率，Java在编译的时候放弃检查接口和类之间是否有实现关系。当类进行强制转换的时候，JVM在编译的时候会对两个类进行检查，检查这两个类之间是否有继承关系。如果有继承关系，则编译的时候会通过，但是运行的时候不一定正确。如果没有继承关系，则在编译的时候直接报错。
13. 作用：统一结构。接口可以作为模板，配合多态实现解耦

## 五、内部类

1. 定义在类或者接口中的类就称之为内部类。内部类是封装的第三种形式
2. 内部类根据使用的位置和修饰符不同分为：方法内部类、成员内部类、静态内部类和匿名内部类
3. 特点：
  - a. 除了静态内部类，其余的内部类中都不允许定义静态属性和静态方法，但是可以定义静态常量
  - b. 除了静态内部类，其余的内部类都可以使用当前外部类的属性和方法，但是静态内部类只能使用外部类的静态成员。
4. 方法内部类
  - a. 定义在方法里的类叫做方法内部类，也叫局部内部类
  - b. 特点
    - i. 可以定义成员属性和成员方法
    - ii. 不可以定义静态属性和静态方法，但是可以定义静态常量
    - iii. 可以继承其他类和实现接口。注意：在这个继承和实现只是只能继承和实现外部类的接口和类，内部类和内部接口是不能被继承和实现的
    - iv. 可以使用外部类的属性和方法
    - v. 不可以使用当前方法里的局部变量，但是可以使用当前方法中的局部常量
  - c. 作用：方法内部类是为了私有本类方法中的参数
5. 成员内部类
  - a. 定义在类内方法外的类叫做成员内部类，也就是成员变量的位置，利用外部类对象来创建成员内部类对象
  - b. 特点
    - i. 成员内部类可以定义非静态变量和非静态方法
    - ii. 不可以定义静态变量和静态方法
    - iii. 可以定义静态常量
    - iv. 可以使用外部类中的一切属性和一切方法
    - v. 内部类的权限可以定义为私有的
    - vi. 可以继承类或者实现接口
6. 静态内部类
  - a. 用static修饰的成员内部类叫做静态内部类。可以直接利用外部类来创建静态内部类的对象
  - b. 特点
    - i. 可以定义成员属性和成员方法
    - ii. 可以定义静态属性和静态方法

iii. 不允许使用外部类里面的非静态属性和静态方法

## 7. 匿名内部类

- a. 没有名字的内部类叫做匿名内部类。包含成员匿名内部类，方法匿名内部类
- b. 注意：抽象类可以创建匿名内部类，实体类只要不是最终类就可以创建匿名内部类。当利用接口创建匿名内部类的时候，实际上是实现了对应的接口
- c. 如果匿名内部类定义在了方法或者语句内的时候，使用规则和方法是一致的。当定义在类内时，使用方法和成员内部类相同。本质上是继承了对应的类或者实现了对应的接口
- d. 可以使用匿名内部类的方式创建对象
- e. 特点
  - i. 匿名内部类本质上是继承了对应的类或者实现对应的接口
  - ii. 只要一个类可以被继承，那么这个类就可以出现匿名内部类的形式，当利用一个类来创建一个匿名内部类的时候，实际上这个匿名内部类是继承了这个类
  - iii. 匿名内部类有构造函数。但是不能进行手动的添加
  - iv. 当利用匿名内部类来创建对象的时候，要求这个匿名内部类必须实现父类中的所有抽象方法
  - v. 如果匿名内部类定义到了方法中，此时匿名内部类的使用规则和方法内部类一致
  - vi. 如果匿名内部类定义到了类中，此时匿名内部类的使用规则和成员内部类一致

## 六、包

1. 声明包用的package，包的产生是为了解决同名文件的问题
2. 注意：一个Java文件中只允许存在一个package语句，而且这个package语句必须放在整个Java文件的首行
3. 导入包用的是import。在导包的时候，\*表示通配符，用于导入指定包下的所有的类而不包括子包下的类。例如import java.util.\*; // 表示导入util包下的所有的类而不包括util子包下的类
4. java.lang包下的类在程序运行的时候会自动导入，所以java.lang包下的类在使用的时候不需要写导包语句
5. 同一个包下的类在使用的时候也不需要导包
6. 注意：包名在命名的时候尽量不以java, javax等开头

## 七、垃圾分代回收机制

1. 垃圾回收针对的是堆内存
2. 对象在堆内存中存储，对象在使用完成之后会在不定的某个时刻被垃圾回收器（GC - Garbage Collector）解析掉。现阶段回收过程无法手动控制。当调用构造方法的时候，创建好一个对象，因为Java中对每种数据类型都明确给定了大小，在创建对象的时候，会自动计算大小分配内存，所以在内存的回收和释放的时候也是由Java自己管理
3. 堆内存分为了新生代（年轻代）和老生代。新生代划分为伊甸园区和幸存区。一个对象新创建是放到了伊甸园区，如果这个对象在栈内存中没有引用，那么会在扫描的时候被解析，释放内存；在伊甸园区经过了一次扫描如果依然存活则标记到幸存区。幸存区的扫描频率要略低于伊甸园区。如果在幸存区中经过了多次扫描这个对象依然没有被解析，则标记到老生代。老生代的扫描频率要远远低于新生代。如果老生代的对象发生了回收，导致程序的卡顿甚至崩溃
4. 发生在新生代的回收称之为minor gc，即初代回收；发生在老生代的回收称之为full gc，即完全回收