# 面相对象

**创建对象实例用什么关键字，对象实例和对象引用有什么区别**

new的对象实例放在堆中，由对象引用指针指向对象实例（也可以为null）

**类的组成部分**

局部变量：定义时未初始化无默认值，需要赋值后才能使用，否在会报错

成员变量： 定义时未初始化会有默认

静态变量(类变量)：类加载时初始化

成员方法

静态方法(类方法)

代码块:实例化时执行一次

静态代码块：类加载时，执行一次

构造函数

**成员变量和局部变量的区别**

存储位置：成员变量在对象实例中(堆)，局部变量存在栈中(方法中定义的变量)

生命周期：成员变量随对象一起死亡，局部变量当方法调用完就死亡

初始值：成员变量有默认值，局部变量必须赋值后使用

**类初始化过程**

静态块(父子)， 父代码块/构造函数, 子代码块/构造函数

**构造方法能不能显式调用，可以被重载和重写吗**

构造方法不能显式调用，只有创建对象时系统调用

会有默认的无参构造（前提：没有自定义的构造方法）

构造方法可以重载，不能重写，无返回值

**内部类有哪些**

定义在类的内部

静态内部类：类中定义，用static修饰，定义在成员位置上

成员内部类：定义在成员位置上的类

局部内部类：定义在方法中的类

匿名内部类：没有名字的内部类

匿名内部类必须继承抽象类或实现接口，必须实现抽象类或接口的所有抽象方法

匿名内部类不能定义任何静态成员和静态方法

匿名内部类需要使用所在方法的形参使用时，必须声明为 final

**内部类优点**

内部类对象可以访问创建它的外部类对象的内容，包括私有数据

内部类不被同包的其他类所见，有更好的封装性

内部类可以多重继承，弥补单继承的缺陷

匿名内部类可以方便的定义回调

**内部类的应用场景**

解决非面向对象的语句块

局部内部类访问局部变量时，为什么变量要加上final

因为生命周期不同， 局部变量在栈中，当方法执行结束后，非final的局部变量被销毁。局部内部类对局部变量的引用依然存在，如果局部内部类要调用局部变量时，就会出错。加了final，确保局部内部类使用的变量与外层的局部变量区分开，解决了这个问题

**static关键字修饰变量，方法，代码块，类时有什么不同的作用**

static修饰的变量：静态变量, 所有实例共享，通过类名.变量名来访问

static修饰的方法：静态方法，不能重写，通过类名.方法名来访问，不能使用非静态方法/变量

static修饰的代码块：静态代码块，在类加载时执行一次，多个静态代码块按序执行

static修饰的类：静态内部类

public static void main(String[] args){}:表示主程序

**final关键字修饰变量、对象、方法、类分别有什么作用？**

final修饰变量：常量，创建时初始化，常量值不能改变

final 修饰对象：引用地址不可变

final 修饰方法：方法不能被重写

final修饰类：最终类，不能被继承

**如何实现多态**

继承，重写，父类引用指向子类对象

编译看左，运行看右（只能调用父类有的方法（编译看左），运行的结果是子类中对应的方法（运行看右））

**多态原理：静态分派和动态分派的区别**

Parent p=new Son();

静态分派：根据变量的静态类型来确定执行哪个方法，Parent就是变量的静态类，发生在编译期间

应用场景(重载)

public void sayHello(Paraent p) {System.out.println("hello,p!");}

sayHello(p); //输出hello,p!

动态分派：根据变量的动态类型来确定执行哪个方法，Son就是变量的动态类型,发生在运行期间

例子：p.sayHello(); //hello,p

son.sayHello(); //hello,son

**java中是单继承还是多继承**

类单继承类，多实现接口

接口可以多继承接口

**super和this表示什么**

super: 当前类的父类对象

super() 构造方法不能被继承，子类默认调用父类无参构造，若无无参构造，子类要用super()调用父类有参构造(放第一行)

this：表示当前类的自身，成员变量和局部变量重名时，表示当前的成员变量

this(): 调用主构造方法，需要在第一行，和super()不能同时使用

**abstract关键字修饰类、方法分别有什么作用**

修饰类：表示该类不能被实例化，必须被继承，可以有普通变量和方法，

修饰方法：一定在抽象类中，表示继承该类后，该方法要被实现

**接口(interface)的作用和特点**

弥补单继承的缺点

变量都是 public static final

方法都是抽象方法

**为什么要实现Serializable接口**

将java对象序列化(转成二进制字节流)，便于数据存储和传输，只有实现了该接口才能被序列化和反序列化

场景：持久化，将数据存入数据库；网络，进程之间传输

**Java面向对象的三大特性**

封装：封装属性和方法，对外暴露调用的接口；不必了解类内部具体细节，减少耦合，对成员控制更精准

继承：子类可以继承父类的所有成员，但是对private变量的无访问权限，private方法不会被继承

多态: 就是用父类的引用指向子类的对象

     格式：父类 变量名=new 子类();

List<Integer> list = new ArrayList<>();

多态调用方法时，先查看父类中是否有该方法，如果没有则报错，如果有，再去调用子类的同名方法

**访问修饰符和作用范围**

private 当前类  
default（默认） 当前类 + 同包  
protected 当前类 + 同包 + 子类  
public 当前类 + 同包 + 子类 + 其他包

**重载和重写的区别，构造方法能被重写吗**

重载：相同的方法名，参数个数/类型不同

重写: 子类重写父类的方法(static方法不能被重写)，方法名，形参，返回类型都不能改变

子类返回类型小于等于父类方法返回类型，  
子类抛出异常小于等于父类方法抛出异常，  
子类访问权限大于等于父类方法访问权限。

构造器不能被重写

**抽象类和接口的区别**

抽象类：可以有变量，抽象/非抽象方法,静态方法和静态代码块，单继承类，方法(默认 default)

接口：变量(默认public static final， 需要初始化)， 方法(默认public abstract)，多继承接口

**接口中可以有具体方法吗**

Jdk8后接口中可以有静态方法和defaultl修饰的默认方法