# 内部类

## ***内部类概述***

**就是在一个类中再定义一个类**

**内部类访问特点**

**内部类可以直接访问外部类的成员,包括私有.**

**外部类想要访问内部类的成员,必须先创建对象.**

## *成员内部类*

**内部类分类**

**成员内部类:在类的成员位置**

**局部内部类:在类的局部位置**

**成员内部类在外界如何创建对象,然后调用**

**创建格式: 外部类名.内部类名 对象名 = new 外部类名.内部类名();**

**注意:**

**1.内部类没有主方法(main方法);**

**2.内部类的内部还可以有内部类**

**3.一般内部类用pirvate修饰,所以其实不能用(外部类.内部类)访问,而是在外部类中创建方法method方法,在method方法中创建内部类的对象,调用内部类的成员.然后再通过外部类调用method方法,在method方法里面创建了内部类的对象,调用了内部类的成员的这种方式间接的访问了内部类的成员**

## *局部内部类*

**是在方法定义中的类,在外部类的局部位置(类中且方法中).外界无法直接访问,需要在方法内部创建对象并使用,然后外部类调用该方法,才能访问到.**

**局部内部类既可以访问外部类的成员位置,也可以访问方法中的局部变量.**

## *匿名内部类*

**前提:存在一个类或者接口;**

**这个类可以是具体类也可以是抽象类,一般多为抽象类.**

**格式:**

**new 类名/接口名(){**

**重写方法;**

**};**

**本质:是一个继承了该类,或者实现了该接口的子类匿名对象.**

# Math类

## *Math类概述*

**public final class Math extends Object{}**

**Math包含执行基本的数字数学运算的方法**

**Math类没有构造方法,使用math类的成员是使用类名调用,他的成员也全部都是静态修饰的**

## *Math常用方法*

**Math.abs(int a); 返回一个int数的绝对值**

**Math.ceil(double a); 返回>=a的一个最小int值**

**Math.floor(double a); 返回<=a的一个最大int值**

**Math.round(float a); 返回四舍五入的一个int值**

**Math.max(int a, int b); 返回两个数的较大值**

**Math.min(int a, int b); 返回两个数的较小值**

**Math.pow(double a, double b); 返回a的b次幂值**

**Math.random(); 返回一个[0.0,1.0)的double值**

# System类

## *System类概述*

**public final class Math extends Object{}**

**System包含了几个有用的类字段和方法,它不能被实例化.**

## *System的常用方法*

**System.exit(); 终止当前运行的Java虚拟机,非零为异常终止**

**System.currentTimeMillis();返回当前时间,以毫秒为单位**

**以1970年1月1日为标准**

# Object类

## *Object类概述*

**public class Object{}**

**Object类是类层次结构的根类,每个类都有Object作为超类,所有类直接或者间接的继承自该类.所有对象(包括数组)都实现了这个类的方法.**

## *Object的toString()方法*

**返回对象的字符串表达形式(地址值).**

**查看println源码,去看底层构架里面有object的toString方法,但是一般都会去重写toString方法,例如在学生类中重写.在打印对象名时就会从地址值改成变量属性...**

## *Object的equals()方法*

**比较对象是否相同,默认比较的时地址,也需要重写比较内容,可以自动生成,但是要理解代码作用和步骤**

# Arrays类

**冒泡排序:从小到大排列数组元素,双for循环**

## *Arrays类概述*

**public class Arrays extends Object{}**

**Arrays类包含操作数组的各种方法**

**java.util包下,需要导包**

## *Arrays常用方法*

**Arrays.toString(); 传入一个数组,按指定格式返回数组内容的字符串形式**

**Arrays.sort(); 按照数字顺序排列指定数组**

**Array是一个工具类**

**工具类的设计思想是**

**构造方法用private 修饰 防止外界创建对象**

**成员用public static 修饰 需要使用类名来访问成员**

# 基本类型包装类

**将基本数据类型封装成对象类的好处在于可以在对象中定义更多的功能方法来操作数据.**

## *基本数据类型的包装类*

**int→Integer char→Character**

**其余6个首字母大写**

**常用的基本操作**

**1.用于基本数据类型与字符串之间的转换;**

**2.**

## *Integer类*

### *Integer类概述*

**public class Integer extengs Number Implements Comparable<Integer>**

**Integer类包装了一个对象中的一个int值,他提供了几种将int转化为String和String转化为int的方法,以及其他处理int的方法**

**构造方法来创建Integer对象在进行操作已经过时**

**Integer i = new Integer();**

**现在使用类名调用静态方法来给对象赋值**

**Integer I = Integer.valveOf(int a); Integer I = Integer.valveOf(String s);**

**使用静态方法获取对象;**

**Int和String的相互转换**

**Int → String : String.valveOf(int i) 括号里面还可以是其他数据类型**

**String → int : Integer.parseInt(String s) 将字符串解析为int类型数**

## *自动装箱和拆箱*

**基本类型包装类**

**装箱:把基本数据类型转换为对应的包装类类型;**

**拆箱:把包装类类型转换为对应的基本数据类型;**

**Integer i = 100; //自动装箱**

**i += 200;**

**// i = i + 200; i + 200 自动拆箱**

**// i = i + 200; 自动装箱**

**注意:在使用包装类类型的时候,如果做操作,最好先判断是否为null**

**推荐是,只要是对象,在使用前都要进行不为null的判断.**

# 日期类

## *Date类*

### *Date类的概述和构造方法*

**继承和实现体系**

**public class Date extends Object implements Serializable, Cloneable, Comparable<Date>**

**Date类代表了一个特定的时间,以毫秒为精度**

**Date的构造方法**

**无参构造:**

**Date(); 分配了一个Date对象,并初始化.用对象代表他被分配的时间,精确到毫秒**

**Date(long date); 分配了一个Date对象,并将其初始化为表示从基准时间起指定的毫秒数**

**传入的参数在1970.01.01的基础上增加时间,传入的是毫秒值;**

**特别的:**

**打印Date对象得到的不是地址值,而是当前时间,说明重写了toString方法**

### *Date类的常用方法*

**d.getTime(); 获取的是日期对象从1970.01.01 00:00:00 到现在的毫秒值**

**d.setTime(); 设置时间**

## *SimpleDateFormat类*

### SimpleDateFormat类的概述

**继承体系:**

**java.lang.Object**

**java.test.Format**

**java.test.DateFormat**

**java.test.SimpleDateFormat**

**实现体系:**

**Implements** **Serializable, Cloneable**

### *SimpleDateFormat格式化和解析日期*

**特别的:该类一般在创建对象的时候都是带参构造,参数是给定的输出格式或者说是解析格式**

**若使用无参构造创建对象则与要再次调用applyLocalizedPattern​(String pattern)或者applyPattern​(String pattern)这两个方法给定具体格式,两个方法返回都是void**

**格式化 从Date到String**

**public final String format(Date date){}**

**将日期格式化成一个字符串,有默认格式,也可以按照需求自定义格式,这里查看API文档查看具体年月日对应字母, 如年→y**

**解析 从String到Date**

**public Date parse(String source){}**

**从给定的字符串开始,解析文本,生成日期**

**需要注意的是,解析是要按照固定格式的,就好比String使用split方法截取字符串返回字符串数组.**

## *Calendar类*

### *Calendar类概述*

**日历类**

**Calengar类为某一时刻和一组日历字符串之间的转换提供了一些方法,并为操作日历字符串提供了一些方法**

**Calendar类提供了一个构造方法getInstance(); 用于获取Calendar对象,其日历字段已使用当前日期和时间初始化**

**Calendar rightNow = Calendar.getInstance();**

**java.util包下**

### *继承和实现体系*

**public abstract class Calendar extends Object implements Serializable, Cloneable, Comparable<Calendar>**

**子类: GregorianCalendar**

**Calendar类是一个抽象类**

### *Calendar的构造方法*

**Calendar类是一个抽象类,使用父类引用指向子类对象的多态方式来实现对象的创建,直接打印对象名得到的重写后的日历情况,不是地址**

### *Calendar的常用方法:*

**public int get(int field)**

**根据给定的field值获取指定的信息,比如Calendar.YEAR对应是年,Calendar.MONTH对应月,不过这里的月从0开始,所以要+1.**

**如:c.get(Calendar.YEAR) 得到当前年,需要变量接收**

**public abstract void add(int field, int amount)**

**根据日历规则,前面参数代表是日历时间量,后面参数代表对该时间量数据的操作**

**如:c.add(Calendar.YEAR, -2) 代表当前年-2,不需要变量接受**

**public final void set(int year, int month, int date)**

**设置当前日历的年月日,设置完成后需要get接收并输出**

**如:**

**c.set(2013, 9, 25);**

**int year2 = c.get(Calendar.YEAR);**

**int month2 = c.get(Calendar.MONTH);**

**int day2 = c.get(Calendar.DAY\_OF\_MONTH);**

**System.out.println(year2 + "年" + month2 + "月" + day2 + "日");**

**总结:Calendar类给出了日历上时间等参数的量,可以供我们操作,通过对象调用对应你需要操作的方法,将时间等参数按照指定格式以及需求完成.可以个之前的Date类,SimpleDateFormat类进行一起操作.**

**可操作的需求,时间差,时常,给出时间,时间表等**

**案例:二月天**

**获取任意一年的二月有多少天**