1. 二进制样式的数字相加。

比如：计算2+3；2的二进制位10,3为11

先计算10和11相加不算进位的结果。方法：2^3. **^为异或**。结果为1

然后计算进位。方法：2&3. &为与操作，结果为10.因为涉及到进位，所以还需要加个操作—左移一位。即：(2&3)<<1 ,结果为100

把上述两个结果继续进行异或和与操作。直到进位为0即止。最后结果为101.即为5

**public** **int** Add(**int** a,**int** b) {

**while** (b!=0) {

**int** temp = a^b;

             b = (a&b)<<1;

             a = temp;

         }

**return** a;

     }

1. ASCLL码表

() 40,41

\* + , - . / 42-47

0-9 48-57

1. Z 65-90
2. z 97-122
3. 静态代码块

类中可以有静态代码块和非静态代码块，静态代码块在加载类的时候自动调用，而且只调一次。非静态代码块需要被调用，new一个对象的时候就调用一次。

执行顺序：静态代码块——>非静态代码块——>构造方法

1. 重写equals方法与hashcode方法
   1. 为什么要重写equals方法

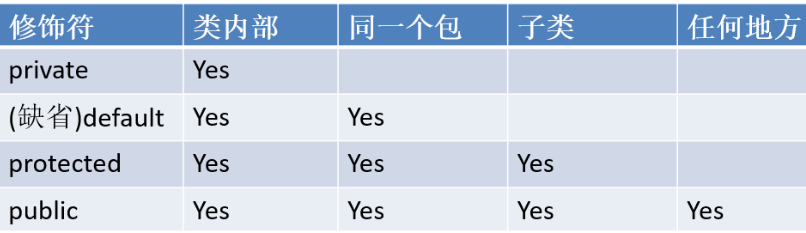
判断两个对象是否相等？默认的equals方法是对两个对象引用的比较，而业务中一般都是要对对象里面的值进行比较的。

* 1. 为什么要重写hashcode方法

不是所有的类都要重写hashcode的，只有用到hash结构时候才需要重载hashcode,默认的hashcode也是用引用来生成的。hash结构里是用hashcode来存储查找的，当两个对象相等时，hashcode必须是一样的。但是hashcode相等，对象不一定相等。

1. 堆排序
   1. 介绍：使用完全二叉树逻辑结构，数组存储结构。
      1. 流程：首先把无序的堆构建成堆，输出堆顶（数组第一个元素与最后一个交换），对输出之后的堆排序，重复操作。
      2. 从小到大排序用大根堆，反之亦然。
      3. 关键问题
         1. 将一个无序序列构造成一个堆
         2. 输出堆顶元素后，调整剩余元素成为一个新堆
2. 算法的稳定性：
   1. 不稳定：快排，希尔，选择排序，堆排序（快希选堆——>快些选队）
   2. 稳定：冒泡，插入，归并，基数





修饰class的只能是public和默认。默认的权限只能同一个包引用这个类。

1. JVM
   1. 运行时内存区：
      1. 方法区：是线程共享的内存区域，用来存储类加载的信息、常量、静态变量、即时编译器编译后的代码等。其中方法区中还有个经常会用到的区域叫做运行时常量池，主要用于存储一些常量，当创建一个常量时，首先会在运行时常量池查看是否有，有则直接使用，否则重新创建。
      2. 堆：堆是最大的一块内存区域，也是垃圾回收管理的主要区域，主要用于存放对象实例和数组。
      3. 程序计时器：线程私有的，每个线程都会分配一个线程计时器，用来表示当前线程执行的字节码的行号指示器。在多线程中，一个线程执行的时候释放锁，另一个线程执行完，再回来执行前面线程的时候，就是通过程序计时器来获取继续执行的位置。
      4. 虚拟机栈：虚拟机栈主要存储基本数据类型变量和引用类型变量。其中与堆的区别就是如：Obj obj=new Object()；等号左边则是在虚拟机栈上分配栈区存储引用类型变量的句柄Obj obj，等号右边则是存储对象实例，栈区的句柄是指向堆区的对象实例的，一般通过句柄访问堆区的对象实例。
      5. 本地方法栈：与虚拟机栈意义相似，区别在于虚拟机栈用于Java方法，而本地方法栈则是针对于Native方法服务。
   2. 知识点：
      1. 堆主要用来存放对象，栈主要用来执行程序。JVM为每一个线程都分配一个栈，对一个java程序来说，它的运行就是通过栈操作完成的，栈以帧为单位保存线程状态。当线程激活一个函数时，jvm就会在栈里压入一个帧，此方法运行期间，这个帧用来保存参数，局部变量，中间计算过程和其他数据，
   3. 垃圾回收
      1. 算法：
         1. 标记-清除
         2. 复制算法
         3. 标记-整理
         4. 分代收集
      2. JVM中共划分为三个代：年轻代、年老代和持久代
         1. 年轻代：存放所有新生成的对象
         2. 年老代：在年轻代中经历了N次垃圾回收仍然存活的对象
         3. 持久代：用于存放静态文件，如Java类、方法等。

Ps:

* + 1. New创建的对象和数组放在堆内存中。

堆中产生的数组和对象，可以在栈中定义一个特殊变量，变量值为对应数组或对象的首地址，这个引用变量在程序运行到作用域外就释放。

1. 锁
   1. 分为乐观锁和悲观锁，synchronized是悲观锁，偏向锁，自旋锁是乐观锁。
   2. Java锁有方法锁，对象锁和类锁。
      1. 方法锁就是对象锁，创建一个类,synchronized修饰普通方法,即为对象锁，不同对象访问同一个被synchronized修饰的方法的时候不会阻塞。

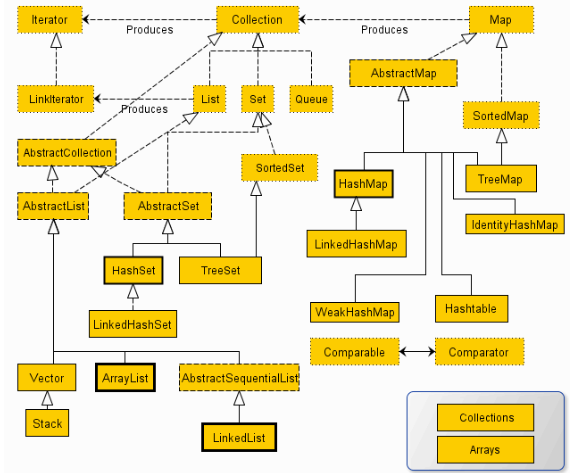
Synchronized(this){}

private synchronized void method1(){}

这两种都是对象锁

* + 1. 类锁是锁住整个类，不管多少个对象,共用一把锁,且只有一把,不管怎么调用,都会同步
       1. private static synchronized void method(){} 对象锁直接加static就变成了类锁
       2. Synchronized(SingleTon.class) 也是类锁

1. 集合
   1. 框架图



* 1. 扩容：
     1. arrayList初始默认为10，1.5倍加1扩容
     2. Vector 初始默认长度也为10，当扩容容量增量大于0时、新数组长度为原数组长度+扩容容量增量、否则新数组长度为原数组长度的2倍。之后讲新数组长度：newCapacity与所需的最小容量：minCapacity进行比较、较大者作为最终的新长度。
     3. HashMap初始默认16，增长因子为0.75，即数量超过现有长度的75%，则一倍扩容。
     4. HashTable初始默认11，增长因子也为0.75，一倍扩容再加1。
  2. 遍历
     1. HashMap

HashMap<Integer,Integer> map =new HashMap<Integer,Integer>();

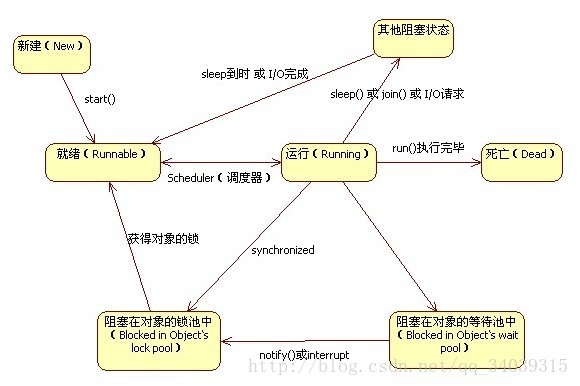
for(Entry<Integer, Integer> entry: map.entrySet()) {

entry.getKey();

entry.getValue();

}

1. 多线程



* 1. 进程和线程的区别
     1. 一个进程是一个独立的运行环境， 可以看做一个程序或者应用。而线程是进程中执行的一个任务。线程需要较少的资源来创建和驻留在进程中，并且可以共享进程中的资源。
  2. 多线程的好处
     1. 最大程度的利用cpu，提高程序的效率。
     2. 多线程可以共享堆内存，所以执行任务，多线程比多进程更好
  3. 如何创建一个线程
     1. 实现Runnable接口，然后将它传递给Thread的构造函数，创建一个Thread对象。之后重写run方法。
     2. 继承Thread类。之后重写run方法。
  4. 死锁
     1. 银行家算法：避免死锁

资源有序分配法：预防死锁

资源分配图化简法：检测死锁

撤销进程法：解决死锁

1. 内部类
   1. 内部类一般包括四类：成员内部类、局部内部类、匿名内部类和静态内部类
      1. 成员内部类
         1. 成员内部类可以无条件访问外部类的所有成员属性和成员方法（包括private成员和静态成员）。

class Circle {

    private double radius = 0;

    public static int count =1;

    public Circle(double radius) {

        this.radius = radius;

    }

    class Draw {     //内部类

        public void drawSahpe() {

            System.out.println(radius);  //外部类的private成员

            System.out.println(count);   //外部类的静态成员

        }

    }}

* + - 1. 当成员内部类拥有和外部类同名的成员变量或者方法时，会发生隐藏现象，即默认情况下访问的是成员内部类的成员，要访问外部类，需要使用外部类.this.成员变量（方法）
      2. 外部类中如果要访问成员内部类的成员，必须先创建一个成员内部类的对象，再通过指向这个对象的引用来访问
      3. 成员内部类是依附外部类而存在的，如果要创建成员内部类的对象，前提是必须存在一个外部类的对象

Outter outter = new Outter();

        Outter.Inner inner = outter.new Inner(); //inner是outter的成员内 部类

* + - 1. 外部类只能被pubic 和默认修饰

内部类可以像变量一样有四种修饰。

Private修饰：只能在外部类的内部访问

默认修饰：只能在同一个包下访问

Protected修饰：只能在同一个包下或者继承外部类的情况下访问

Public修饰：任何地方都能访问

* + 1. 局部内部类
       1. 定义在一个方法或者一个作用域里面的类，它和成员内部类的区别在于局部内部类的访问仅限于方法内或者该作用域内

class People{

    public People() {}

}

class Man{

    public Man(){}

    public People getWoman(){

        class Woman extends People{   //局部内部类

            int age =0;

        }

        return new Woman();

    }

}

* + - 1. 局部内部类就像是方法里面的一个局部变量一样，是不能有public、protected、private以及static修饰符的。
    1. 匿名内部类
       1. 匿名内部类是唯一一种没有构造器的类。正因为其没有构造器，所以匿名内部类的使用范围非常有限，大部分匿名内部类用于接口回调。

匿名内部类在编译的时候由系统自动起名为Outter$1.class。

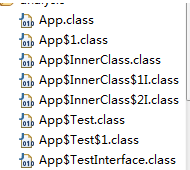
一般来说，匿名内部类用于继承其他类或是实现接口，并不需要增加额外 的方法，只是对继承方法的实现或是重写。

* + 1. 静态内部类
       1. 静态内部类也是定义在另一个类里面的类，只不过在成员内部类的前面多了一个关键字static。（因为局部内部类和匿名内部类前面是不能加修饰符的）静态内部类是不需要依赖于外部类的，并且它不能使用外部类的非static成员变量或者方法，这点很好理解，因为在没有外部类的对象的情况下，可以创建静态内部类的对象，如果允许访问外部类的非static成员就会产生矛盾，因为外部类的非static成员必须依附于具体的对象。
  1. 内部类命名规则
     1. 分两种，成员内部类和静态内部类为一种，局部内部类与匿名内部类为一种

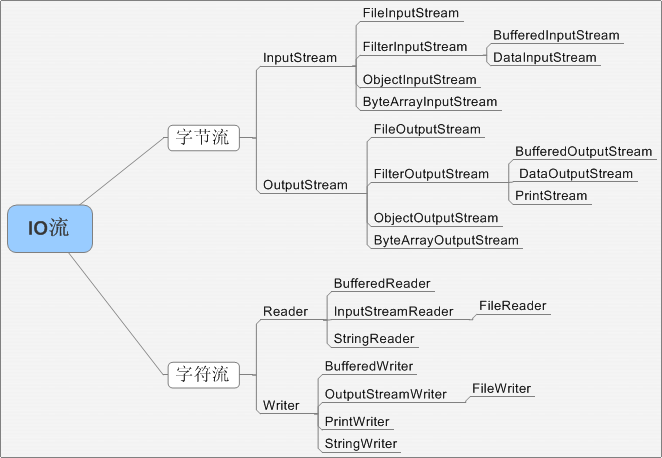
有几个类就会有几个文件，都会生成外部类名字命名的Outer.class。

$就是一层的意思，a$b的意思就是a类里面的b内部类。

* + - 1. 如果类有名字,那么就是xxx$xx.class,对于没有名字匿名内部类,用数字代替类名,因为数字在语言规范里是不正确的类名,用户代码里面的类名不可能就是一个数字.
      2. 第二个,对于方法体里面内部类,编译后的类名,xxx$xxx$数字+类名.class格式，一定要加数字,因为无法保证在另外一个方法里面不会出现同样的类名.
      3. 举例：



1. IO



1. 数据库
   1. 索引
      1. 包括普通索引（允许重复的值出现）、唯一索引（不能有重复的值）、主键索引、聚集索引
      2. 数据库结构为B+树
   2. 表之间的连接
      1. 包括内连接，外连接和交叉连接
      2. Select \* from a join b on a.x=b.x(内连接)
      3. Select \* from a left (right) join b on a.x=b.x (外连接)
      4. Select \* from a cross join b where....(交叉连接，不能用on，不用写连接要求)
   3. 范式
      1. 第一范式：无重复的列，列数据的不可分割。
      2. 第二范式：属性完全依赖于主键
      3. 第三范式：属性不传递依赖于主键
   4. Sql语句优化
      1. 尽量避免where语句使用!=或<>,还有避免判断字段为空，因为这样不会使用索引