1. 面向对象和面向过程

面向过程：把问题分解成一个一个步骤，每个步骤用函数实现，依次调用即可。

就是说，在进行面向过程编程的时候，不需要考虑那么多，上来先定义一个函数，然后使用各种诸如if-else、for-each等方式进行代码执行。

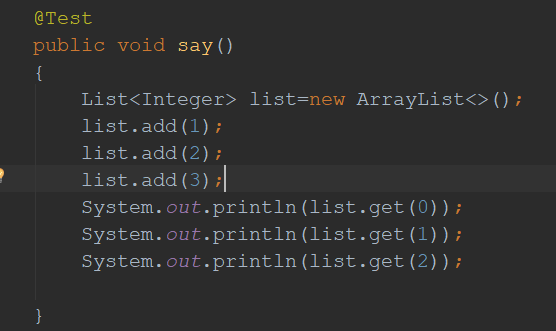
最典型的用法就是实现一个简单的算法，比如实现冒泡排序。

面向对象：将问题分解成一个一个步骤，对每个步骤进行相应的抽象，形成对象，通过不同对象之间的调用，组合解决问题。

就是说，在进行面向对象进行编程的时候，要把属性、行为等封装成对象，然后基于这些对象及对象的能力进行业务逻辑的实现。

比如:想要造一辆车，上来要先把车的各种属性定义出来，然后抽象成一个Car类。

1. 面向对象的基本特征：如果分为3种为封装，继承，多态。4种为抽象，继承，封装，多态。抽象可以整合到封装里面。
2. 包装类和其基本自动类型自动装箱解箱的实现原理



List只能放Integer类型的对象，把int基本数据类型放到list中，编译器会把int基本数据类型转换成Integer类型的对象，这个编译器的过程就是装箱的过程。从list中取出数据的时候，拿的也是Integer类型的对象，编译器把Integer类型的对象转换为int基本数据类型，编译器的这个过程就是拆箱的过程。

由于装箱和拆箱都是编译器自动完成，不需要在代码中显式的说明，故称为自动装箱和自动拆箱。

基本数据类型对应的包装类分别为：1，byte，Byte。2，short，Short。3，char，Character。4，int，Integer。5，long，Long。6，float，Float。7，double，Double。8，boolean，Boolean。

如何实现自动装箱和自动拆箱？

自动拆箱和自动装箱由编译器自动完成。当编译器检测到需要把基本数据类型转换为包装类型的时候，编译器自动调用包装类的valueOf（）方法实现自动装箱。当编译前检测到需要把包装类型转换为基本数据类型的时候，编译器自动调用包装类型的xxValue（）方法。

// Byte

public static Byte valueOf(byte b) {

final int offset = 128;

return ByteCache.cache[(int)b + offset];

}

// Short

public static Short valueOf(short s) {

final int offset = 128;

int sAsInt = s;

if (sAsInt >= -128 && sAsInt <= 127) { // must cache

return ShortCache.cache[sAsInt + offset];

}

return new Short(s);

}

// Character

public static Character valueOf(char c) {

if (c <= 127) { // must cache

return CharacterCache.cache[(int)c];

}

return new Character(c);

}

// Ingeger

public static Integer valueOf(int i) {

if (i >= IntegerCache.low && i <= IntegerCache.high)

return IntegerCache.cache[i + (-IntegerCache.low)];

return new Integer(i);

}

// Long

public static Long valueOf(long l) {

final int offset = 128;

if (l >= -128 && l <= 127) { // will cache

return LongCache.cache[(int)l + offset];

}

return new Long(l);

}

// Double

public static Double valueOf(double d) {

return new Double(d);

}

// Float

public static Float valueOf(float f) {

return new Float(f);

}

// Boolean

public static Boolean valueOf(boolean b) {

return (b ? TRUE : FALSE);

}

从装箱的源代码可知，当基本类型为byte,short,char,int和long类型时，首先根据赋值的数值大小判断是否在缓存区，否则直接new一个对应的包装类型对象；注意只有double和float会直接new一个包装类。（因为基本数据类型byte，char和int用的较多，所以Java在包装类初始化的时候在内存中缓存一定范围的值，避免生成大量的对象。缓存范围为[-128,127]，正好是short基本类型的范围）。

八大基本数据类型自动拆箱的原理代码：

public xxx xxxValue() {

return value;

}

自动装箱和自动拆箱有什么作用？有什么弊端？

DK1.5之前，编译器还没有实现自动装箱和自动拆箱，需要显式的编写装箱和拆箱的过程;如下面的代码：

intList.add(Integer.valueOf(100));

intList.add(Integer.valueOf(150));

intList.add(Integer.valueOf(200));

int \_a = intList.get(0).intValue();

int \_b = intList.get(1).intValue();

int \_c = intList.get(2).intValue();

可以看出代码很繁杂，JDK1.5之后的valueOf()和intValue()由编译器自动完成，可以简化代码。

自动装箱和自动拆箱给我们带来便利的同时，也带来了弊端，主要体现以下几点：

包装类之间的比较；

public static void method2() {

Integer a = 100;

int \_a = 100;

Integer \_a1 = new Integer(100);

System.out.println("a == \_a : " + (a == \_a)); // true

System.out.println("a == \_a1 : " + (a == \_a1)); // false

System.out.println("\_a == \_a1 : " + (\_a == \_a1)); //true

Integer b = 200;

int \_b = 200;

Integer \_b1 = new Integer(200);

System.out.println("b == \_b : " + (b == \_b)); // false

System.out.println("b == \_b1 : " + (b == \_b1)); // false

System.out.println("\_b == \_b1 : " + (\_b == \_b1)); //false

}

==可以用于对象与对象之间的比较，比较的是对象的内存地址是否相等，即是否是同一个对象，比较的过程没有自动拆箱；也可以用于基本数据类型之间的比较，比较的是字面值；所以，对于对象之间的比较应该用equals()方法。

稍不留神，容易创建额外对象

Integer result = 0;

for (int i = 127; i < 1127; i++) {

result += i;

}

1

2

3

4

上面的result += i 实际会先进行自动拆箱操作，然后进行自动装箱操作。即

result = result.intValue() + i;

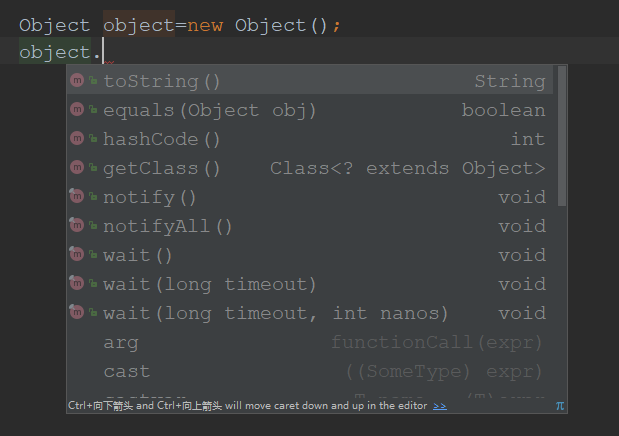
result = result.valueOf(result);

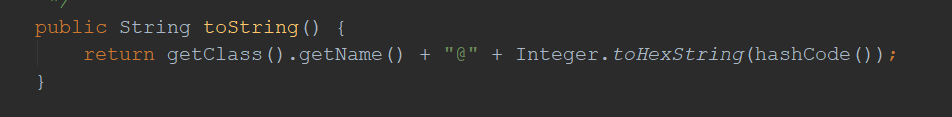
1

2

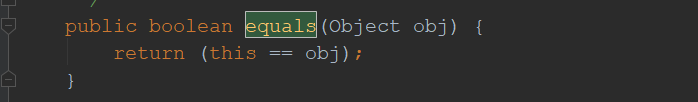
自动装箱实际会创建1000各无用的Integer对象，会给内存造成极大的浪费；所以编写代码中避免使用不必要的自动装箱。

1. Object有哪些方法



toString方法,一般子类都有覆盖，此方法使用较多。

equals方法，该方法是非常重要的一个方法。一般equals和==是不一样的，但是在Object中两者是一样的。子类一般都要重写这个方法。



hashCode该方法用于哈希查找，可以减少在查找中使用equals的次数，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hash- Code()==obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

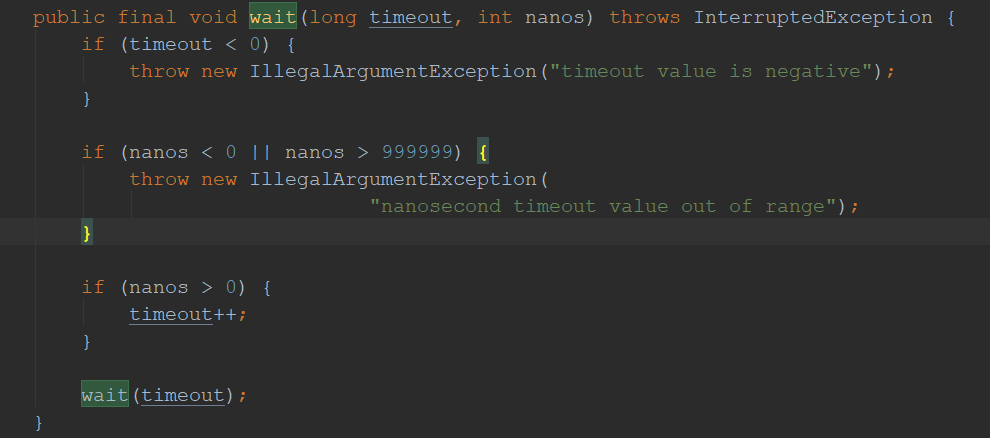
getClass,final方法，获得运行时类型。

notify,唤醒在该对象上等待的某个线程

notifyAll,唤醒在该对象上等待的全部线程

Wait，等待线程等待该对象的锁

Wait(long)，long是超时时间

Wait(long,int)，后面的参数应该是可以更好的控制时间来使用的

1. Thread的状态及转化

线程共包括以下 5 种状态:

1. 新建状态(New): 线程对象被创建后，就进入了新建状态。例如，Thread thread = new Thread()。

2. 就绪状态(Runnable): 也被称为“可执行状态”。线程对象被创建后，其它线程调用了该对象的start()方法，从而来启动该线程。例如，thread.start()。处于就绪状态的线程，随时可能被CPU调度执行。

3. 运行状态(Running): 线程获取CPU权限进行执行。需要注意的是，线程只能从就绪状态进入到运行状态。

4. 阻塞状态(Blocked): 阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

(01) 等待阻塞 -- 通过调用线程的wait()方法，让线程等待某工作的完成。

(02) 同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态。

(03) 其他阻塞 -- 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5. 死亡状态(Dead): 线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

一张图表示状态转换

1. 输出结果

**public** **class** SystemUtil{

**public** **static** **boolean** isAdmin(String userId){

**return** userId.toLowerCase()=="admin";

    }

**public** **static** **void** main(String[] args){

        System.out.println(isAdmin("Admin"));

    }

}

结果False,equals和==的区别 ==操作比较的是两个变量的值是否相等，对于引用型变量表示的是两个变量在堆中存储的地址是否相同，即栈中的内容是否相同。

在源码中 toLowerCase 是重新 new String()

所以为 == 是比较对象是否是同一个对象，所以为 false 。

1. 输出结果为

String str ="";

System.out.print(str.split(",").length);

结果1，String split 这个方法默认返回一个数组， \* 如果没有找到分隔符， \* 会把整个字符串当成一个长度为1的字符串数组 \* 返回到结果， 所以此处结果就是1。

下列选项中是正确的方法声明的是？（）

protected abstract void f1();

public final void f1() {}

static final void fq(){}

private void f1() {}

ABCD  
A：抽象方法只可以被public 和 protected修饰；  
B：final可以修饰类、方法、变量，分别表示：该类不可继承、该方法不能重写、该变量是常量  
C：static final 可以表达在一起来修饰方法，表示是该方法是静态的不可重写的方法  
D：private 修饰方法（这太常见的）表示私有方法，本类可以访问，外界不能访问

下列说法正确的有（ ）

环境变量可在编译source code时指定

在编译程序时，所能指定的环境变量不包括class path

javac一次可同时编译数个Java源文件

javac.exe能指定编译结果要置于哪个目录（directory）

ACD

|  |
| --- |
| byte b1=1,b2=2,b3,b6,b8;  final byte b4=4,b5=6,b7;  b3=(b1+b2);  /\*语句1\*/  b6=b4+b5;    /\*语句2\*/  b8=(b1+b4);  /\*语句3\*/  b7=(b2+b5);  /\*语句4\*/  System.out.println(b3+b6); |

下列代码片段中，存在编译错误的语句是()

BCD

本题答案应为：B、C、D

------------知识点------------

Java表达式转型规则由低到高转换：

1、所有的byte,short,char型的值将被提升为int型；

2、如果有一个操作数是long型，计算结果是long型；

3、如果有一个操作数是float型，计算结果是float型；

4、如果有一个操作数是double型，计算结果是double型；

5、被fianl修饰的变量不会自动改变类型，当2个final修饰相操作时，结果会根据左边变量的类型而转化。

--------------解析--------------

语句1错误：b3=(b1+b2);自动转为int，所以正确写法为b3=(byte)(b1+b2);或者将b3定义为int；

语句2正确：b6=b4+b5;b4、b5为final类型，不会自动提升，所以和的类型视左边变量类型而定，即b6可以是任意数值类型；

语句3错误：b8=(b1+b4);虽然b4不会自动提升，但b1仍会自动提升，所以结果需要强转，b8=(byte)(b1+b4);

语句4错误：b7=(b2+b5); 同上。同时注意b7是final修饰，即只可赋值一次，便不可再改变。

1. Java的集合类

1、 Collection

包含了集的基本操作和属性的高度抽象的接口。

Collection包含了List和Set两大分支。

(1) List是一个有序的队列，实现类有4个：LinkedList, ArrayList, Vector, Stack。

(2) Set是一个不允许有重复元素的集合。实现类有3个：TreeSet、HastSet、LinkHastSet。

2、 Map

一个映射接口，即key-value键值对。

（1）AbstractMap是个抽象类，它实现了Map接口中的大部分API。实现类有6个：TreeMap、HashMap、LinkHashMap、IdentityHashMap、WeakHashMap、HashTable

（2）SortedMap 是继承于Map的接口。内容是排序的键值对，通过比较器(Comparator)

3、Iterator。

它是遍历集合的工具，即我们通常通过Iterator迭代器来遍历集合。

1. 经常用的集合的数据结构

ArrayList是容量可以改变的非线程安全集合，使用数组进行存储，集合扩容时会创建更大的数组空间，把原有数据复制到新的空间内。支持元素的快速访问，但是插入和删除较慢。

LinkedList 的本质是双向链表。与 ArrayList 相比，LinkedList 的插入和删除速度更快，但是随机访问速度则很慢。

除了继承 AbstractList 抽象类外，LinkedList 还实现了另一个接口 Deque，即 double-ended queue，这个接口同时具有队列和栈的性质。LinkedList 的优点在于可以将零散的内存单元通过附加引用的方式关联起来，形成按链路顺序查找的线性结构，内存利用率较高。

HashMap数据结构有链表和数组两种实现对数据的存储(以后可以加到源码的时候学习。)

1. HashMap的数据结构：JDK7数组加链表实现加红黑树实现的，put的时候是放到数组里面的，那么就一定有一个下标来表示放到哪个数组，直接hash然后与-1逻辑运算。(还没有结束)

HashMap的源码阅读

为什么会产⽣线程安全问题？

线程不安全：

是指不提供加锁机制保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据。

假设售票系统有1000张票，A和B同时来买票，如果是线程不安全，那么可能售票系统可能出现1000-1去同时执行的情况，最终结果是A和B都买完后剩下999张票，而不是998张。

1、引起线程安全问题的原因：

线程安全问题都是由全局变量及静态变量引起的。

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。

2、解决多线程并发访问资源安全问题的方法：

（1）synchronized

synchronized关键字，就是用来控制线程同步的，保证我们的线程在多线程环境下，不被多个线程同时执行，确保我们数据的完整性，使用方法一般是加在方法上。

（2）Lock

Lock是在Java1.6被引入进来的，Lock的引入让锁有了可操作性，是指我们在需要的时候去手动的获取锁和释放锁，甚至我们还可以中断获取以及超时获取的同步特性，从使用上synchronized使用起来比Lock更方便。

1. AQS是啥（源码待分析）

AbstractQueuedSynchronizer（AQS）

类如其名，抽象的队列式的同步器，AQS定义了一套多线程访问共享资源的同步器框架，许多同步类实现都依赖于它，如常用的ReentrantLock/Semaphore/CountDownLatch...。

它是Java并发用来构建锁和其他同步组件的基础框架

AQS是一个抽象类，主是是以继承的方式使用。AQS本身是没有实现任何同步接口的，它仅仅只是定义了同步状态的获取和释放的方法来供自定义的同步组件的使用。从图中可以看出，在java的同步组件中，AQS的子类（Sync等）一般是同步组件的静态内部类，即通过组合的方式使用。

抽象的队列式的同步器，AQS定义了一套多线程访问共享资源的同步器框架，许多同步类实现都依赖于它，如常用的ReentrantLock/Semaphore/CountDownLatch

1. CAS

JUC是java.util.concurrent包的简称，JUC有2大核心，CAS和AQS，CAS是java.util.concurrent.atomic包的基础，即AtomicInteger和AtomicLong等是用CAS实现的