Java 从入门到精通

Java中的“引用”倒是和C/C++的指针更像一些，和C++的“引用”很不一样。C++的“引用”是从一而终的，而Java的“引用”可以改变被引用的对象

java去除指针概念，就用引用罗..

而C++则不然。C++的引用就语义上说是“别名”，而并不是指针的另一种用法：

chapter7 接口和内部类

接口内的变量自动设置为public static final

接口无修饰符缺省为package内可见，或者为public

public interface Animal{

int var=5;

void func1();

} // Java 无 分号，而C++有

class ClassName implements Interface1, Interface2 {

func1(){

}

func2(){

}

}

接口也可以继承

public interface Interface1 extends Interface2 {

funcOfInterface1(){

}

}

the class implenents the Interface1 should implement all functions for both Interface1 and Interface2.

技巧：把一些固定的常量组值放在接口中定义，然后在类中实现该接口。

interface WeekDays {

int Monday=1;

int Tuesday=2;

...

}

class Time implements WeekDays {

void print(){

System.out.println("Monday is"+Monday);

}

}

接口和抽象类的比较

差异：

* 一个类可以实现多个接口，但只能继承一个抽象类
* 抽象类中可以有非抽象方法，而接口中的方法都是抽象的
* 抽象类的方法可以是public，protected or private, 而接口的方法全部为public。
* 抽象类有构造函数，而接口没有

相似：

* 两者都不可实例化
* 都可以用来定义变量，用来存放子类（for抽象类）或者实现类（for接口）的对象，用于实现多态。

内部类是定义在其他类内部的类，内部类所在的类叫宿主类。

主要作用是把相关的类放在一起，并用于隐藏内部类的可见性。

class Host {

......

void useInner(){

Inner in = new Inner();

in.xxxx();

}

class Inner {

......

void xxxx(){}

}

}

内部类分静态内部类和非静态内部类

非静态内部类可以调用宿主类的所有成员和方法

静态内部类不能直接访问宿主类的成员和方法，必须实例化宿主类后通过宿主类的对象来访问。

内部类还包括 局部内部类和匿名内部类

对象的克隆

直接对对象赋值并不能达到克隆的目的，那样只是改变了对象变量的引用。

Class a = new Class；

Class b = a; // this only redirect b point to memory object of a

为了在Java中实现真正的克隆，我们要对我们希望支持克隆的类实现Cloneable接口：

class A implements Cloneable {

...

public Object clone(){

A a = null;

try {

a = (A)super.clone();

}

catch(CloneNotSupportException e) {

e.printStackTrace();

}

}

}

Then call as following:

A a = new A;

A b = a.clone();

注意如果一个类的成员也是一个类，而又希望支持类克隆，则要对逐级类都实现克隆才保险。

Chapter 异常处理

所有异常类都是内置类Throwable的子类，其有两个子类，一个是Error，一个是Exception。

Exception还分两个子类，一个是RuntimeException，另一个是IOException

RuntimeException：描述编程产生的错误，如空指针NullPointerException，其是RuntimeException的子类

try catch语句中如果有多个catch语句块，那么捕获父类异常的catch语句要放在后面，否则编译都会不过。

try catch 中只能有一个finally语句块，放在catch快之后，不管是否发生异常都会执行

在方法中抛出异常，留给方法调用者自己处理异常

public void func() throws Exception1, Exception2, Exception3 {

try {

}

catch ( Exception e) {

throw e;

}

}

创建自己的异常

class MyException extends Exception {

}

chapter 10 线程

Java 中有两种方法创建线程:

一种是对 Thread 类进行派生并覆盖 run 方法;

另一种是通过实现 runnable 接口创建

class ThreadDemo1 extends Thread{

ThreadDemo1(){}

ThreadDemo1(String szName) {

super(szName);

}

public void run() { ...... }  
 public static void main(String argv[ ]){

ThreadDemo1 td = new ThreadDemo1();

td.start();

}

}  
  
class ThreadDemo3 implements Runnable{

public void run() {

…

}

public static void main(String argv[ ]){

Runnable rb = new ThreadDemo3();

Thread td = new Thread(rb);

td.start();

}

}  
  
线程周期概念一个线程有 4 种状态,任何一个线程都处于这 4 种状态中的一种状态。

1. 创建(new)状态:调用 new 方法产生一个线程对象后、调用 start 方法前所处的状态。线程对象虽然已经创建,但还没有调用 start 方法启动,因此无法执行。当线程处于创建状态时,线程对象可以调用 start 方法进入启动状态,也可以调用 stop 方法进入停止状态。

2. 可运行(runnable)状态:当线程对象执行 start()方法后,线程就转到可运行状态。进入此状态只是说明线程对象具有了可以运行的条件,但线程并不一定处于运行状态。因为在单处理器系统中运行多线程程序时,一个时间点只有一个线程运行,系统通过调度机制实现宏观意义上的运行线程共享处理器。因此一个线程是否在运行,除了线程必须处于 Runnable 状态之外,还取决于优先级和调度。

3. 不可运行(non Runnable)状态:线程处于不可运行状态是由于线程被挂起或者发生阻塞,例如对一个线程调用 wait()函数后,它就可能进入阻塞状态;调用线程的notify 或 notifyAll 方法后它才能再次回到可执行状态。

4. 退出(done)状态:一个线程可以从任何一个状态中调用 stop 方法进入退出状态。线程一旦进入退出状态就不存在了,不能再返回到其他的状态。除此之外,如果线程执行完 run 方法,也会自动进入退出状态。

Java 中锁机制的实现方法是共享代码之前加入 synchronized 关键字。

线程通信

Java 中线程之间的通信是通过Object 类中的 wait()、notify()、notifyAll()等几种方法实现的。Java 中每个对象内部不仅有一个对象锁之外,还有一个线程等待队列,这个队列用于存放所有等待对象锁的线程。wait 方法调用的前提条件是当前线程获取了这个对象的锁,也就是说 wait 方法必须放在同步块或同步方法中。

Overload和Override的区别。Overloaded的方法是否可以改变返回值的类型?方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded的方法是可以改变返回值的类型。

[EJB到底是什么，真的那么神秘吗？？](http://blog.csdn.net/jojo52013145/article/details/5783677)

<http://blog.csdn.net/jojo52013145/article/details/5783677>