**定义一个数组**

;//不能指定数组大小，Name只是一个引用变量（指针）

type[] Name = new type[]{element1,element2,...};//type也可换成type的字类

Nmae = {element1,element2,...};

type[] Name = new type[4]{element1,element2,...};

使用Name[0],Name[1],

**类的定义**

[public/final] class 类名

{

1：成员有多种，最常见为构造器，属性，方法

2：成员之间顺序无影响，各种成员之间可以互相调用，

但是static修饰的成员不能访问没有static修饰的成员

3：构造器

[修饰符] 构造器名(类名)(参数列表)

{

语句;

}

修饰符：可省略，public/protected/private其一

构造器名必须为类名，无返回值类型

默认构造器无参数

如程序员定义个一个以上构造器，则系统默认构造器不存在

通过new来调用构造器，返回该类实例的引用

Person P = new Person();

现在可以通过P这个实例访问其包含的方法和属性

P.属性 P.方法

static修饰的属性和方法也可以通过:

类.属性 类.方法

因为构造器主要用于被其他方法调用，所以通常设置为public访问权限

如果构造器A中的内容只是构造器B中的一部分，则可以在构造器B中第一条语句用this调用构造器A

4：属性(Field,字段)

[修饰符] 类型 属性名 [=默认值];

修饰符：可省略,public/protected/private/static/final,

public/protected/private只出现其一

5：方法

[修饰符] 返回值类型 方法名(参数列表)

{

语句;

}

修饰符：可省略，public/protected/private/static/final/abstract,

public/protected/private只出现其一

final/abstract只出现其一

返回值类型：若无则用void

java中的方法不能独立存在，只能在类中定义

java允许同一个类中定义多个同名方法,只要形参列表不同就可，这叫做方法的重载

**方法重载overload**只要求1：方法名相同 2：形参列表不同，至于方法返回值类型，修饰符等与重载无关

6：static可修饰属性和方法，叫类属性和类方法

表明修饰的是属于类共有的，不属于单个实例的

static修饰的成员不能访问没有static修饰的成员

}

**对象,引用,指针**

类 是一种引用数据类型

数组是一种引用数据类型

实际上**java中的引用类型就是c中的指针**

Person P = new Person();

Person P1 = P;//P P1指向同一个对象

当对象没有引用变量指向时，会自动释放

所以如果想释放某个对象的内存，将指向其的引用变了置NULL就可

**this引用**

每个实例中都有一个this引用，类似：

Person this;

且this引用指向对象本身，相当于在构造器结束前将对象引用给了this

其最主要的作用是让类中的方法可以访问本类中的另外一个方法或属性，

因为调用一个方法或属性(非类方法/属性)必须要有实例，所以相当于方法的参数列表中有个隐含参数[类名 this],属性实际为this.属性

注意,static修饰的方法中无这个隐含的参数,static修饰的变量无this.前缀，因为没有实例时可以调用static方法/属性，如果有this参数或前缀会报错，所以static修饰的方法/属性访问不了非static的属性和方法

**java的三大特性之封装**

主要思路是不允许直接操作熟悉，而是通过方法来操作属性，也就是隐藏属性，暴露方法

怎么隐藏？就要用到修饰符

public/protected/private叫做访问修饰符，其实还有一个默认的default

访问控制严格从小到大

**public：**

修饰的属性/方法可以被全局类访问

**protected:**

修饰的属性/方法可以被同一个包内的类访问+可被全局子类访问

**default：**

修饰的属性/方法可以被同一个包内的类访问

**private**

修饰的属性/方法可以被类的内部访问，外部不能访问，这适合对外隐藏属性

**java中的包**

实际就是作用域，两个包中可以包含同名的类，同一个包内不能有重名类

实际上会将包名加到类名前面去

指定类所处于的包中，只需要在源文件最前面加上，这个源文件所有类都属于这个包

package packageName;

位于包中的每个类的完整类名为包名+类名的组合,如文件Hello.java

package lee.sub;

public class Hello{

}

javac –d . Hello.java

会在当前目录生成 ./lee/sub/Hello.class

进入lee，可执行java lee.sub.Hello //注意执行java Hello是错误的

另外一个包内的类想创建一个Hallo类实例，需要加上包名

lee.sub.Hello p = new lee.sub.Hello();

鉴于此，我们应当将java源文件放在与包名一致的目录结构中

一个源文件中只能指定一个包，也就是一条package语句

由于使用不同包内的类需要加上包名比较繁琐，所以引入import关键字，import关键字相当于c中的include，将包中定义的类部分或全部包含到当前源文件中来，import出现在package之后，这些包含进来的类也会成为当前包中的类

java默认在所有源文件中导入java.lang包下的所有类，都是常用类

导入指定类的单个静态属性如下

import static 包名.类名.属性名

导入指定类的全部静态属性如下

import static 包名.类名.\*

**java的三大特性之继承**

java继承具有单继承特点，只有最多一个直接父类

用extends关键字实现继承，格式如下

[修饰符] class subclass extends SuperClass

{…}

子类获取父类全部的属性和方法，除父类构造器

如果一个类没有显式的直接父类，则默认继承java.lang.Object类，因此java.lang.Object类是所有类的父类，要么是直接父类，要么是间接父类

子类继承父类后，一方面增加新的属性和方法，一方面重写继承的方法，重写并不影响父类的方法，只是子类增加了与父类同名的方法，这叫做**方法重写**或**方法覆盖（Override）**,注意，方法覆盖需要遵循

1：方法名，参数列表，是否是static,相同

2：子类方法返回值比父类返回值类型要更小或相等

3：子类方法的访问权限要比父类方法更大或相等

子类覆盖父类方法/属性后，无法直接访问父类方法/属性，可以使用**super.方法**/**super****.属性**访问父类方法/属性

如果父类的方法是private访问权限的，子类定义一个相同名称，参数，是否static的方法，也不是覆盖，只是子类新增的方法而已

**super引用**

java在创建类时，系统会隐式创建其直接父类，直接父类又会创建其父类，直到创建java.lang.Object对象

**每个类中都隐含定义了一个直接父类的引用****super**,子类创建实例时，首先创建直接父类的实例，并将直接父类对象引用给了super

有时需要在子类构造器中执行父类构造器的初始化代码(并不是创建一个父类对象)，这是可在子类构造器第一行代码中使用super.构造器来实现

**java的三大特性之多态**

类BassClass,其子类SubClass

定义一个BassClass的引用变量 BassClass p;

这p的编译类型为BassClass引用

因为SubClass为BassClass的子类，所以可以将SubClass的对象引用给p

p = new SubClass();

则p的运行时类型为SubClass

p只能调用编译时类型的方法和属性，比如BassClass中有方法A ，SubClass中新增了方法B

则p.A正确，p.B报错

如果方法A在SubClass中被覆盖override ,则p.A调用的是SubClass中的A方法，这便是多态

简言之也就是用父类引用调用其子类的方法

属性不具有多态性，直接使用编译时类型，如BassClass中有bass=1,SubClass中有bass=2,则p.bass是1

**继承与组合**

继承与组合其实功能相似，只是继承破坏了父类的封装，组合则没有

一般只有在子类需要增加属性且独有的方法(新增或重写)，才会选择采用继承，否则采用封装

组合就是将父类作为子类的一个属性，在子类实现中有类似如下的语句

prviate BassClass bass;

**基本数据类型的包装类**

因为所有的类的基类都有Object类，但是java提供的8种基本数据类型没有，

为了解决基本类型没有继承Object类的问题，java提供了基本数据类型的包装类

byte包装类Byte,short包装类Short,int包装类Integer,long包装类Long,char包装类Character,float包装类Float,double包装类Double,boolean包装类Boolean

可执行类似如下操作

boolean bl=true;

Boolean blObj = new Boolean(bl);

Boolean blObj = new Boolean(“true”);

Float fl = new Float(“4.5”);

除了Character，其他都可以通过传入一个字符串来创建包装类对象，但是必须是合理的值

Long lObj = new Long(“ddd”);//编译不报错，运行报错

获取包装类中包装的基本类型变量，可以使用xxxValue()方法

boolean = bObject.booleanValue();

float f = f1.floatValue();

因为这样比较繁琐，所以后续的JDK提供了自动装箱和拆箱

直接可以把一个基本类型变量赋给对应的包装类对象引用

Integer inObj = 5;

直接把一个包装类对象引用赋给对应的基本类型变量

int i = intObj;

**单例(Singleton)类**

全局范围内只允许一个对象存在，实现这样的想法，则需要如下支持

1：构造器用private访问权限修饰

2：提供一个static public方法，其中会创建类对象

3：提供一个static 类对象引用，保存创建的对象引用

实现类似如下

class Singleton{

private Singleton(){};

public static Singleton getInstance(){

if(instance == null){instance = new Singleton();}

return instance;

}

private static Singleton instance;

}

**final修饰符**

可用于修饰类，变量，方法，表明修饰的类，变量，方法不可改变

final 变量：表明变量获得初始值后不可改变

变量为成员变量

final修饰的成员变量可在如下几个地方获取初始值

如果修饰的成员变量是类属性：静态初始化块，定义时

如果修饰的成员变量是实例属性：非静态初始化块，定义时，构造器

变量为局部变量

final修饰的局部变量可在如下几个地方获取初始值

定义时，定义未赋值后面只能赋一次值

注意：当final修引用类型的变量时，因为引用变量是地址，只能保证地址不会改变，对于地址的内容也就是对象是可以改变的，比如属性可以重新赋值

final 方法：表明方法不可被子类重写Override

可重载overload

final类：表明修饰的类不可有子类

其是最终类

**抽象类**

为了实现多态，也就是父类的引用调用其子类的方法，有时为简洁引入抽象方法

也就是父类中只包含方法签名，无方法实现，也就是没有方法体，这**相当于C++中的纯虚函数**

C++中包含纯虚函数的类叫抽象类，由于纯虚函数无方法体，所以抽象类不能实例对象，只能被继承

抽象方法和抽象类必须使用abstract修饰符修饰，抽象方法的类只能定义成抽象类，抽象类可以没有抽象方法

**抽象类不能被创建实例**

定义抽象方法只需要在普通方法上增加abstract修饰符并去掉方法体

如public abstract void test();

定义抽象类只需要在类上增加abstract修饰符即可，抽象类不能实例对象

**接口：更彻底的抽象**

接口中只有抽象方法，可有常量定义

**[修饰符] interface 接口名 extends 父接口1，父接口2**

**{…}**

接口可以有多个直接父接口，但是不能继承类

接口中不包含构造器，初始化块等，只可以有

**常量属性**(系统自动加上static和final修饰符)

**抽象方法**(系统自动加上public abstract修饰符)

**内部类(**系统自动加上public abstract修饰符)

**枚举定义(**系统自动加上public abstract修饰符)

使用接口

一个类可以实现一个或多个接口，实现使用implements关键字

**[修饰符] class 类名 extends 父类 implements 接口1，接口2**

**{…}**

implements 必须放于extends 后

接口不能显示继承任何类，单接口类型的引用变了可以直接赋给Object类型的引用变量

**内部类**

可以将类放于另外一个类的内部定义，这叫做内部类，包含内部类的类叫外部类

没有static修饰的叫非静态内部类

非静态内部类成员可以直接访问外部类的私有数据，因为非静态内部类是外部类的成员，但外部类不能访问非静态内部类的属性和方法,因为外部类创建对象时，非静态内部类对象根本不存在

如果外部类想访问非静态内部类的成员，则要显示创建非静态内部类对象

匿名内部类使用与只需要一次使用的类

static修饰符叫静态内部类

这个内部类是外部类的，不是外部类实例的

静态内部类可以包含静态和非静态成员，静态内部类相当于外部类的一个静态成员，不能访问外部类的非静态成员

使用内部类

内部类的访问修饰符决定其访问权限

default则内部类可被与外部类同一包内的类访问

protected则内部类可被与外部类同一包内的类访问+子类访问

public则内部类可被全局访问

注意在外部类外使用内部类时要将类名加全

外部类.内部类

因为非静态内部类的对象必须寄存在外部类对象中，所以创建非静态内部类对象前先创建外部类对象，并用new调用内部类

外部类对象引用.new 内部类构造器

在外部类中使用静态内部类，不需要创建外部类对象

new 外部类名.内部类名

**局部内部类**

若把一个类定义放于方法中，则这个内部类是一个局部内部类，不属于外部类的成员，局部内部类只在方法范围内有效，无需使用访问控制符和static等修饰

**匿名内部类**

创建那种只使用一次的类，创建匿名内部类时立即创建一个该类实例，创建格式如下

new **父构造器(实参列表)** | **接口** ()

{

实现接口或重写父类方法/属性

}

创建匿名内部类时，必须实现接口或抽象父类中所有的抽象方法，因为要创建实例了

如果匿名内部类要访问外部类的局部变量，需要使用final修饰符修饰外部类局部变量，否则报错

匿名内部类必须继承一个父类或实现一个接口,且最多只能继承一个父类或实现一个接口

匿名内部类没有类名，所以无构造器，但是可以有初始化块来实现初始化

interface Product

{…}

创建匿名内部类 test( **new Product(){**

**实现接口或重写父类方法/属性**

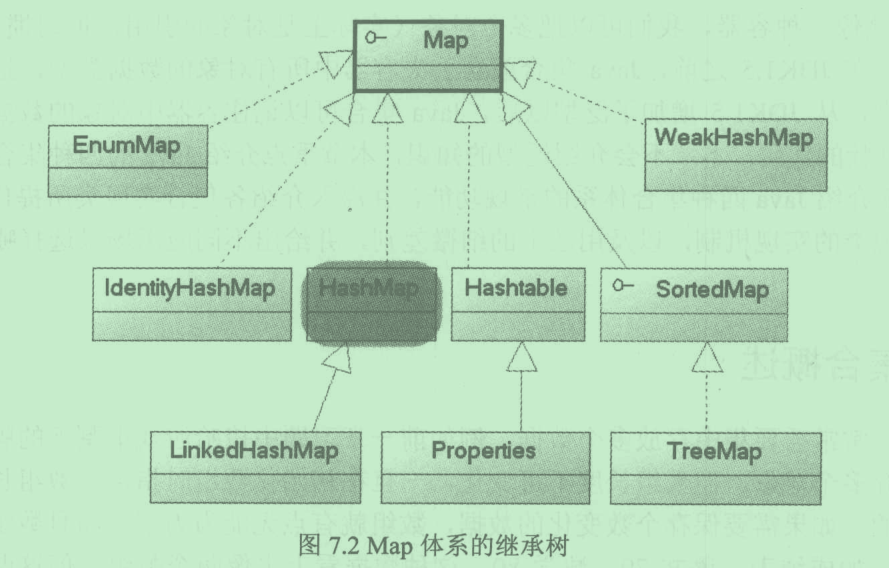
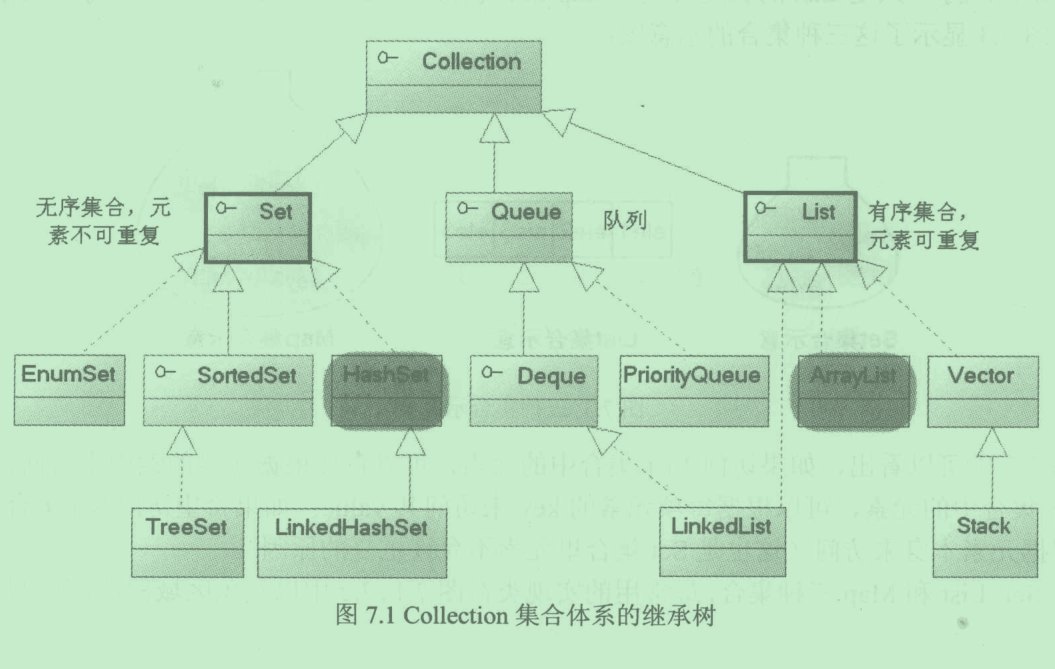
**}** ); //test(Product p)

**java集合**

为了保存数量可变且具有映射关系的数据，java提供集合类，也叫容器类，都位于java.util包下

集合只能保存对象的引用变量，不能保存基本数据类型

java集合类主要由两个接口派生：Collection 和 Map



所有Map实现类用于保存具有映射关系的数据，每项数据都是key-value对，key表示集合中数据的标识，所以不可重复，根据key来取值value

**java中的集合**

可以分为三类

继承Set接口：无序，不可重复，访问元素只能根据元素本身值访问,常用的实现类**HashSet类**

继承List接口：类似数组，只是长度可变，访问元素可以根据元素索引访问,常用的实现类**ArrayList类**

继承Map接口：每项数据又key-value两个值组成，访问元素根据key访问其value,常用的实现类**HashMap类**

**Collection接口**

Collection接口是List,Set,Queue接口的父接口，其中的给出了能对元素操作的抽象方法

**boolean add(Object o);**向集合添加一个元素，成功返回true

**boolean addAll(Collection c);**将集合c中的所有元素添加到指定集合里，成功返回true

**void clear();**清楚集合里所有元素，集合长度变为0

**boolean contains(Object o);**返回集合是否包含指定元素

**boolean containsAll(Collection c);**返回集合是否包含集合c中所有元素

**boolean isEmpty();**返回集合是否为空，空返回true

**Iterator iterator();**返回一个Iterator对象，用于遍历集合里的元素,也叫**迭代器**

**boolean remove(Object o);**删除集合指定元素o

**boolean removeAll(Collection c);**从集合删除集合c中包含的所有元素，相当于该集合减去c集合

**boolean retainAll(Collection c);**从集合删除c中不包含的元素，相当于得到与c集合的交集

**int size();**返回集合众元素个数

**Object[] toArray();**将集合转换成一个数组

Iterator向程序提供遍历Collection集合元素的统一接口，主要有如下方法

boolean hasNext();

Object next();集合里下一个元素

void remove;删除集合里上一次next方法返回的元素

**Set接口**

Set集合的元素无序，不可重复

如果试图加入两个相同的元素，add会返回false且新元素不能被加入

其实现类 HashSet类

当向HashSet对象添加一个元素时，HashSet对象会调用**该元素对象的方法hashCode()** 来得到该元素的hashCode值，然后根据hashCode值将元素存入相应的位置

**List接口**

List集合代表一个有序集合，类似数组，但长度可变

相对于父接口Collection新增的抽象方法

**void add(int index,Object element);**将元素element插入在List集合的index处

**boolean addAll(int index,Collection c);**将集合c所包含的所有元素都插入到List集合的index处

**Object get(int index);**返回集合index处的元素

**int indexOf(Object o);**返回对象o的索引

**int lastIndexOf(Object o);**返回对象o最后一次出现的索引

**Object remove(int index);**删除index处的元素

**Object set(int index,Object element);**替换index处的对象

**List subList(int formIndex,int toIndex);**返回formIndex和toIndex之间所有元素组成的子集合

**ListIterator listIterator();**返回一个**迭代器对象**

ListIterator继承于Iterator，且新增如下方法

boolean hasPrevious();

Object previous;

void add();

**其实现类 ArrayList和Vector**

这两个实现类都有一个capaciry属性，表示集合的长度，相当于数组的大小，若开始就知道这两个实现类需要存储多少元素，可以直接指定capacity大小

这两个类都提供如下方法来操作capacity属性

void ensureCapacity(int minCapacity);将capacity增加minCapacity

void trimToSize();调整capacity大小，可节省存储空间

Vector还提供了一个Stack子类，它用于模拟栈这种数据结构，Stack类提供如下几个方法

Object peek();返回栈的第一个元素，并不pop出栈

Object pop();返回栈的第一个元素，pop出栈

void push(Object item);push进栈

**Queue接口**

用于模拟队列这种数据结构，先进先出，通常不允许随机访问队列中的元素

**Map接口**

Map集合用于保存具有映射关系的数据，key-value，相当于有两组值，一组全是key,一组全是value

一个key对应一个value，通过key来寻找value，不允许存在key相同的对象，key,vlaue都可以是任何引用类型

所有的key组成一个Set集合，Map集合提供的keySet()就是返回Map中所有key组成的Set集合

Map接口提供如下抽象方法

void clear();删除该Map对象中所有key-value对

boolean containsKey(Object key);查询是否包含指定key

boolean containsValue(Object value);查询是否包含value

Set entrySet();返回Map中的所有key-value对所组成的Set集合，每个集合元素都是Map.Entry对象

boolean isEmpty();查询Map集合是否为空

Object put(Object key,Object value);添加一个key-value对，若有key相同，则覆盖之

void putAll(Map m);将指定Map集合的key-value复制到本Map集合中

Object remove(Object key);删除指定key所对应的key-value对

int size();返回key-value对数

Collection values();返回value组成的Collection集合

Map接口包含Entry内部类，该类封装了key-value对，Entry提供如下方法

Object getKey();返回该Entry对象包含的key值

Object getValue();返回该Entry对象包含的value值

Object setValue(V value);设置该Entry里的value值

**我们可以将Map集合理解为一个特殊的Set集合，只是该Set集合众的元素是Entry对象**

Map接口的实现类

HashMap和Hashtable

HashMap中key不能重复，所以最多有一个key为null的key-value对

**泛型**

由于java集合中一旦把一个对象丢进集合，集合就会忘记对象的类型，取出对象后腰强制类型转换，比较麻烦，所以java在JDK1.5增加泛型支持

类似于C++中的模版

public class Apple**<T>**

{

private **T** info;

public Apple(**T** info){ this.info = info;}

public **T**  getInfo(){…}

}

Apple**<String>** a1 = new Apple**<String>**(“apple”);

Apple**<Double>** a2 = new Apple**<Double>**(5.5);

上述定义了一个带泛型声明的Apple<T>类

创建带泛型声明的接口，父类后，可以为该接口创建实现类，或者派生出子类，此时需将**类型参数指定具体类型**

public class A extends Apple<T>{…}//错误

public class A extends Apple<String>{…}//正确

**线程**

java使用Thread类代表线程，所有线程对象都必须是Thread类或其子类的实例

每个线程的任务是完成一定的任务，实际就是一段程序，java中用run方法封装这段程序流

**通过继承Thread类创建线程类**

创建并启动一个线程步骤如下：

1：定义一个Thread类的子类，并重写run方法

2：创建Thread子类的实例，也就是线程对象

3：用线程对象的start方法启动该线程对象

public class FirstThread extends Thread

{

public void run(){…}

}

new FirstThread().start();//启动第一条线程

new FirstThread().start();//启动第二条线程

线程对象有两个方法

Thread.currentThread();//静态方法，返回当前正在执行的线程对象

getName();返回调用该方法的线程名字

可以通过setName(String name)设置线程的名字，一般主线程名字为main,用户启动的线程名依次为Thread-0 Thread-1 …

**实现Runnable接口创建线程类**

创建并启动一个线程步骤如下：

1：定义Runnable接口的实现类，重写run方法

2：创建Runnable实现类的实例，并以此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象

3：调用线程对象的start方法来启动线程

SecondThread st = new SecondThread();//创建Runnable实现类的对象

new Thread(st).start();

Runable对象只是作为Thread对象的target，主要是Runnable实现类中的run方法，实际线程仍然为Thread实例

两种方式的对比

采用Runnable方式下，多个线程共享同一个target对象，适合多线程处理一份资源的情况

**线程的生命周期**

线程生命周期有新建，就绪，运行，阻塞，死亡五种状态

当调用start()方法后，该线程处于就绪状态，至于何时运行，看系统调度

当run方法执行完成，线程正常结束，或直接调用线程的stop()方法

**控制线程**

Thread提供让一个线程等待另外一个线程完成的方法：joib()

当某个程序执行流中调用了线程A.join方法，则调用处阻塞，直到线程A完成为止

**改变线程的优先级**

每个线程都有执行时都有一定的优先级，优先级高的执行的机会较多

每个线程的默认优先级与创建它的父线程相同，main具有普通优先级6

Thread提供了setPriority(int newPriority)和getPriority()方法来设置和返回指定线程的优先级，newPriority的值范围是1-10之间，java提供三个宏表示三个优先级

MAX\_PRIORITY 值为10

MIN\_PRIORITY 值为1

NORM\_PRIORITY 值为5

**线程同步**

两个线程可能修改同一个临界资源，此时需要线程同步

java多线程引入了同步监视器来解决这个问题，使用同步监视器的通用方法就是同步代码块，语法如下

synchronized(obj) //**obj就是同步监视器**

{

//同步代码块，也就是临界区

}

任何时刻只有一个线程可以获得同步监视器的锁定，同步代码块执行结束后，该 线程释放对该同步监视器的锁得

虽然java允许任何对象作为同步监视器，但建议用临界资源充当同步监视器

JDK1.5之后，java提供了**同步锁Lock对象**来实现线程间的同步

**线程池**

系统启动一个线程是需要开销的，可以使用线程池来提高性能

线程池是系统启动时就创建的大量的空闲线程，程序将一个Runnable对象传给线程池，线程池就会启动一个线程来执行该Runnable对象的run方法，run方法执行完毕后，线程不会死亡而是成为空闲状态

线程池可以控制系统中并发线程的数量，其中的最大线程数参数可以控制系统中并发的现场数目不超过此数

JDK1.5中提供一个Executors工厂类来产生线程池，其中包含如下几个静态方法

newCachedThreadPool();//创建一个有缓存的线程池，系统根据需要创建线程，并缓存于线程池中

newFixedThreadPool(int nThreads);创建一个可重用，具有固定线程数的线程池

newSingleThreadExecutor();创建一个只有单线程的线程池

newScheduledThreadPool(int corePoolSize);创建指定数目的线程池

newSingThreadScheduledExecutor();创建只有一个线程的线程池

以上五个方法前三个返回一个线程池对象ExecutorService,它可以执行Runnable对象或Callable对象所代表的线程

后两个方法返回一个ScheduledExecutorService线程池对象，其为ExecutorService的子类，可以指定延迟后执行线程任务

ExecutorService代表尽快执行线程的线程池对象，程序将一个Runnable或Callable对象给该线程池即可，该线程池会尽快执行该任务

ScheduledExecutorService代表可以指定延迟或周期性执行线程任务的线程池

可调用线程池的shutdown()f方法将线程池关闭

使用线程池执行线程的步骤

1：调用Executors类的静态方法创建一个ExecutorService对象，其表示一个线程池

2：创建Runnable或Callable实现类的实例

3：调用线程池对象ExecutorService的submit方法来提交Runnable或Callable实例

4：当想关闭线程池时，调用ExecutorService对象的shutdown方法关闭线程池

**线程相关的类之ThreadLocal**

java为线程安全提供了一些工具类，如ThreadLocal类，它代表一个**线程的局部变量**，通过把数据放在ThreadLocal中就可以让每条线程创建一个该变量的副本，从而 避免并发访问的线程安全问题

ThreadLocal类

通过ThreadLocal类可以简化 多线程编程的并发访问，JDK1.5后为该类增加了泛型支持ThreadLocal<T>

ThreadLocal是将临界的资源复制出多份，每个线程都有一个，每个线程修改资源并不会影响到其他线程的资源，这这主要用到临界的资源不需要全局一致的情况下，比如对于同一个账户，两个线程同时执行取现操作，用ThreadLocal后可以得到对于每个线程账户对象的名字属性可以不同，但对于账户余额这种全局一致的属性则不能用ThreadLocal

class Account

{

prviate ThreadLocal<String> name = **new ThreadLocal<String>();**

public Account(String str){**name.set(str);**}

public void setName(String str){**name.set(srt);**}

}

class Test extends Thread

{

private Account account;

public Test(Account account,String name){ super(name); account = account; }

public void run(){ **account.setName(getName());** }//account对象中德name值设置为线程的名字

}

Account at = new Account(“初识账户名”);**//账户建立时创建这个线程局部变量，有几个线程用到Account这个临界资源，则这个局部变量对象name就复制几分，每个线程独有自己的name，每个线程都可以去改变Account中的name值，获取name值用name.get(),设置name值用name.set(xxx);**