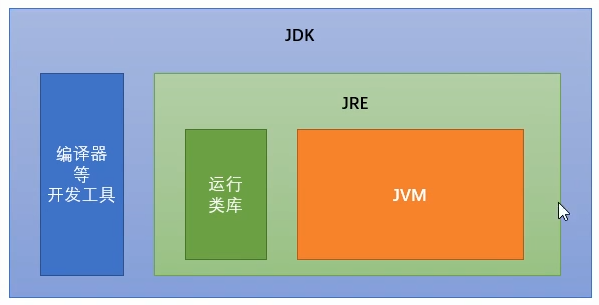
# Java学习

## 一 JRE和JDK

* JRE:是java程序的运行环境
* JDK：是java程序的开发工具包，包含JRE和开发人员使用的工具。

我们想要运行一个已有的Java程序，那么只需要安装JRE即可。

我们想要开发一个全新的Java程序，那么必须安装JDK。



## 二 java基础知识

### 关键字的特点

* 完全小写的字母。
* 在开发工具中有特殊颜色。

### 标识符

是指在程序中，我们自己定义的内容。比如类的名字、方法的名字和变量的名字等等，都是表示符。

* 命名规则：
* 标识符包含字母、数字、下划线。
* 标识符不能以数字开头
* 标识符不能是关键字
* 命名规范：
* 类名规范：首字母大写，后面每个单词首字母大写
* 变量名规范：首字母小写，后面每个单词首字母大写。
* 方法名规范：同变量名。

### 方法

* 方法定义的先后顺序无所谓
* 方法定义不能嵌套，即不能在一个方法内定义另一个方法。
* 方法定义好之后，不会自动执行。如果想要执行，一定要在main方 法中调用方法。
* 如果方法中有返回值，那么必须写上“return返回值”
* rerurn后面的返回值类型必须和方法的返回值类型一致。
* Void方法为无返回值类型，return可以省略不写。
* 一个方法中可以有多个retrun语句，但必须保证同时只有一个会被 执行。
* 方法定义格式：

public static void 方法名称（）{

方法体

}

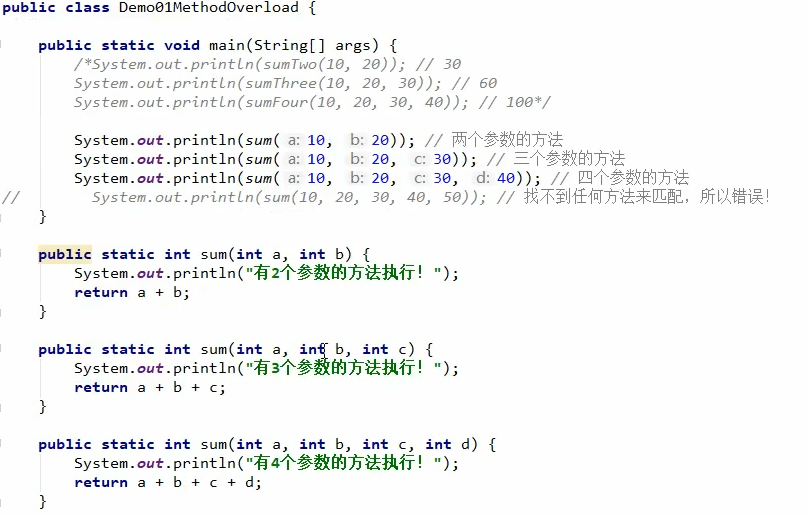
* 方法调用的三种格式：
  + 单独调用：方法名称（参数）；
  + 打印调用：System.out.println(方法名称（参数）);
  + 赋值调用：数据类型 变量名称 = 方法名称（参数）；

注意：此前学习的方法，返回值类型固定写为void,这种方法只能单独调用，不能进行打印调用或者赋值调用。因为void为无返回值类型。

### 方法重载（Overload）

什么是方法重载：方法重载是多个方法名称一样，参数列表不一样。

优点：只需要记住唯一一个方法名，就可以实现类似的多个功能。



方法重载与下列因素相关：

* 参数个数不同
* 参数类型不同
* 参数的多个顺序不同

### 数组

数组是一种容器，可以同时存放多种数据值。

* 动态初始化数组格式：

数据类型[ ] 数组名称 = new 数组类型[数组长度];

* 解析含义:
  + 左侧数据类型：数组中保存的数据，全都是统一的什么类型。
  + 左侧中括号：代表我是一个数组。
  + 左侧数组名称：给数组去一个名字。
  + 右侧的new：代表创建数组的动作。
  + 右侧数据类型：和左侧数据类型一致。
  + 右侧中括号的长度：也就是说数组，要保存多少个数据，是一个int 类型数字。
* 静态初始化基本赋值格式：

数据类型[ ] 数组名称 = new 数组类型[ ] {元素1，元素2，...};

* 静态初始化省略赋值格式：

数据类型[ ] 数组名称 = {元素1，元素2，...};

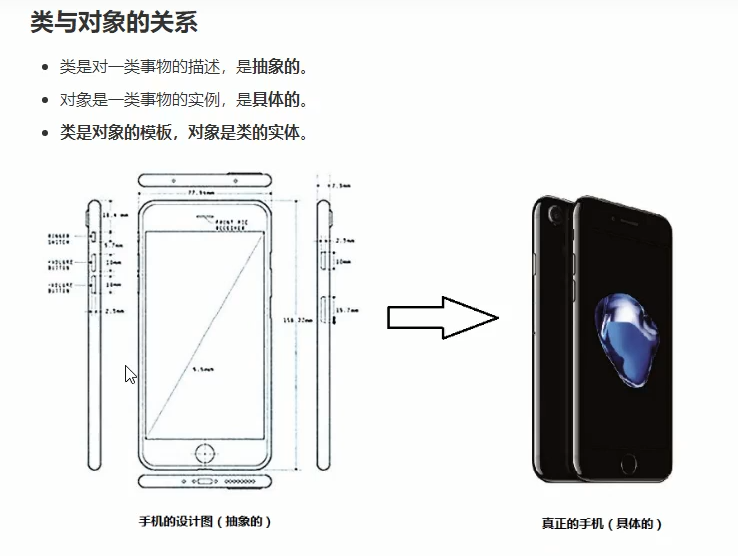
* 求数组的长度

Int[ ] array = { 1,3,4,5}

Int len = array.length;//求数组的长度

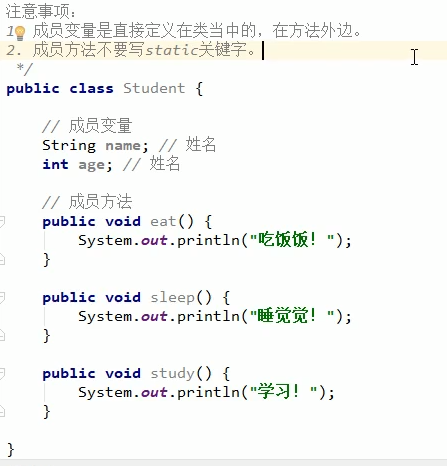
### 类和对象

* 1. 类：是一组相关**属性**的**行为**集合。可以看成一类事物的模板，使用事物的属性特征和行为特征来描述该类事物。
  2. 属性：就是该事物的状态信息（是什么）。（名字，体重，年龄，颜色）
  3. 行为：就是该事物能够做什么（做什么）。（走，跑，叫）
  4. 对象：是一类事物的具体体现。对象是类的一个实例，必然具备该类事物的属性和行为。（一只小猫）



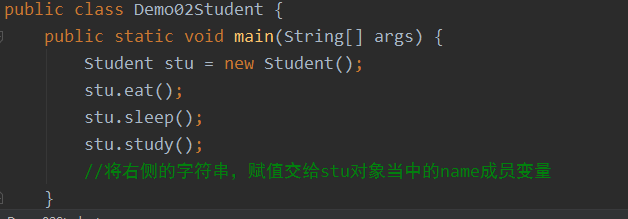
### 类对应的Java的类中

* 属性（成员变量）
  + String name;//姓名
  + Int age;//年龄
* 行为（成员方法）
  + Public void eat( ) { }//吃饭 注意！！成员方法中Public 后面没有static
  + Public void sleep( ) { } //睡觉



### 创建对象

通常一个类不能单独使用，需要根据类创建一个对象，才能使用  
 1.导包：也就时指出需要使用的类，在什么位置。  
 import 包名称.类名称；  
 import com.text.day01.demo03.Student;  
 对于和当前类属于同一个包的情况，可以省略导包语句不写。  
 2.创建 格式  
 类名称 对象名 = new 类名称();  
 //根据Student类,创建一个名为stu的对象  
 Student stu = new Student();  
 3.使用  
 使用成员变量：对象名.成员变量名  
 使用成员方法：对象名.成员方法名(参数)  
 （想用谁，就用对象名点谁）



当一个对象作为参数，传递到方法中时，实际上传递进去的是对象的参数的地址值。

### 局部变量和成员变量

* 1. 定义位置不一样

局部变量：在方法内部

成员变量：在方法外部，直接写在类当中

* 1. 作用范围不一样

局部变量：只有方法当中才能使用，出了方法就不能在使用

成员变量：整个类全都可以使用

* 1. 默认值不一样

局部变量：没有默认值，如果要使用，必须手动进行赋值。

成员变量：如果没有进行赋值，会有默认，规则和数组一样。

* 1. 内存位置不一样

局部变量：位于栈内存（硬盘）。

成员变量：位于堆内存（运行内存）。

* 1. 生命周期不一样

局部变量：随着方法进栈而产生，随着方法出栈而消失（执行到执行结束）。

成员变量：随时创建对象而产生的，随之对象被垃圾回收而消失（从创建到销毁）。

### 面向对象包含三大特征

封装、继承、多态。

### 封装

封装就是将一些细节性的东西隐藏起来，对于外界不可见。

封装性在Java中的体现：

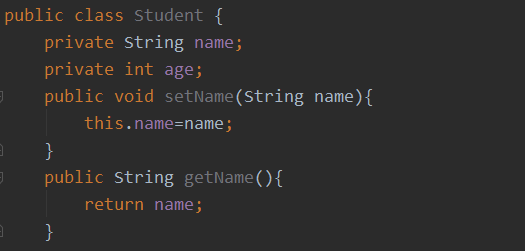
1. 方法就是一种封装
2. 关键字private也是一种封装

### Private关键字

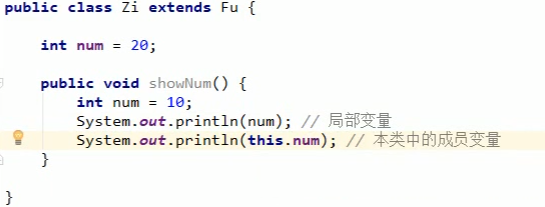
1. 在使用private（私有的）进行修饰（privete int age;），那么本类当中任然可以随 意访问，但是！超出本类范围之外就不能再直接访问。
2. 间接访问private 成员变量，就是定义一对setXxx/getXxx方法。
3. Set方法用于保存数据，必须有参数、无返回值。Set类型和set的成员变量类型 一致。
4. Get方法用于获取数据，没有参数、有返回值。返回值和成员变量类型一致。
5. 对于一般的成员变量是定义一对setXxx/getXxx方法。而对于boolean类型是定 义setXxx/isMale方法。

### this关键字

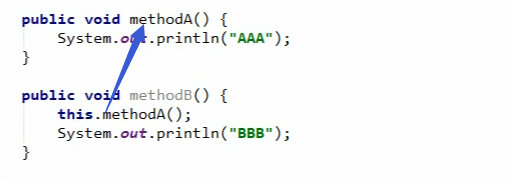
* 当方法的局部变量和类的成员变量重名的时候，根据“就近原则”，优先使用局部变量。如果需要访问本类中被覆盖的的成员变量，需要使用格式：this.成员变量名。
* 通过谁调用的方法，谁就是this位于行。唯一的。



* this关键字可以用来访问本类内容（继承关系中的super关键字来访问父类内容）：
  + 在本类的成员方法中，访问本类成员变量。（this.成员变量）



* + 在本类的成员方法中，访问本类的另一个成员方法。（this.成员方法）



* + 在本类的构造方法中，访问本类的另一个构造方法。（this.构造方法）

### 构造方法

构造方法是专门用来创建对象的方法，当我们通过关键字new来创建对象时，其实就是在调用构造方法。

1. 构造方法格式：

public 类名（参数类型 参数名称，....）{

方法体

}

2. 注意事项：

1. 构造方法名称必须和所在类名称完全一致，就连大小写也完全一样。
2. 构造方法不要写返回值类型，就连void都不写。例：public Stuedt( ){ }
3. 构造方法不能return一个具体返回值。
4. 如果没有任何构造方法，那么编译器将会赠送一个构造方法，没有参数、方法体什么都不做。public Stuedt( ){ }
5. 一旦编写了一个构造方法，那么编译器将不再赠送。
6. 构造方法也可以重载。

### 定义一个标准的类

一个标准的类通常有四个组成部分：

1. 所有的变量成员都要使用private关键字修饰。
2. 为每一个成员变量编写一个Getter/Setter方法。

（快捷键：鼠标右键→Generate→Getter and Setter→全选→OK）

1. 编写一个无参数的构造方法。

（快捷键：鼠标右键→Generate→Constructor→Select None）

1. 编写一个全参数的构造方法。

（快捷键：鼠标右键→Generate→Constructor→全选→OK）

这样的标准类也叫做Java Bean

### Scanner类 键盘输入

Scanner类的功能：可以实现从键盘输入数据，到程序当中。

引用类型的一般使用步骤:

1. 导包

格式： import 包路径.类名称；

如果需要使用的目标类，和当前类位于同一包下，则可以省略导包语句不写。

只有java.long包下的内容不需要导包，String就在其中。

1. 创建

类名称 对象名 = new 类名称（）；

1. 使用

对象名.成员方法名（）

Scanner的使用步骤

1. 导包
   1. import java.util.Scanner;
2. 创建

//固定写法，万年不变

//System.in代表从键盘上进行输入

* 1. Scanner sc = new Scanner(System.in);

1. 使用
   1. 获取键盘输入的一个int数字：int num = sc.nextInt（）;
   2. 获取键盘输入的一个字符串：String num = sc.next（）;

### 匿名对象

匿名对象就是只有右边的对象，没有左边的名字和赋值运算符。

格式：new 类名称()；

注意事项：匿名对象只能使用唯一的一次，下次再用不得不在创建一个新的对象。

使用建议：如果确定有一个对象只需要使用唯一的一次，就可以使用匿名对象。

### Random类 产生随机数

Randoml类用来产生随机数字，使用起来也是三个步骤同：

1. 导包

import java.util.Random;

1. 创建

Random random = new Random();//小括号留空就好

1. 使用

(1)获取一个随机的int数字(范围是int所有范围，有正负两中)： int r = random.nextInt()

(2)获取一个随机的int数字（参数代表范围，左闭右开区间）：int r = random.nextInt(10)

实际上代表的含义是[0,10)，也就是产生的随机在0~9之间

### ArrayList集合类

* 数组的长度不可发生变化，但是ArrayList集合的长度是可以随意变化的。
* 对于ArrayList来说有一个尖括号<E>代表泛型。
  + 泛型：也就是装载集合中的所有元素，全都是统一的什么类型。
  + 注意：泛型只能是引用类型不能是基本类型 (int,float...)。
* 固定格式ArrayList<E> 集合名称 = new ArrayLis<E>();
* 从JDK 1.7+开始，右侧尖括号内部可以省略尖括号中的E不写

即：ArrayList<E> 集合名称 = new ArrayLis<>();

* + 例如定义一个字符串类型集合：

ArrayList<String> list = new ArrayList<>();

* + 创建一个ArrayList集合，集合名字为list，里面装的全是String字符串类型的数据。
  + 备注：从JDK 1.7+开始，右侧尖括号内部可以不写内容，但<>本身还是要写的。
* 注意事项：
  + 对于ArrayList集合来说，直接打印集合名list得到的不是地址值，而是内容。如果内容是空，得到的是空的中括号:[]
* 可以通过list.add();向集合中存数据。
  + 例如:list.add(“库里”);//添加进去的元素和泛型一致
* ArrayLsti中常用的方法
  + public boolean add(E):向集合中添加元素，参数的类型和泛型一致。add方法是带有返回值的,返回值代表添加是否成功，成功返回true。(其中E代表泛型)

备注：对于ArrayList集合来说，add添加动作一定是成功的，返回 值可以不写。但是对于其他集合来说，add添加动作不一定成功。

例如 ：list.add(“库里”);

* + public E get(int index);从集合中获取元素，参数是索引编号，返回值是对应位置的元素。

例如：String name=list.get(1);获取集合中的1号元素给name

* + public E remove(int index);从集合中删除元素，参数是索引编号，返回值是被删除的元素。

例如：String name= list.remove(1);删除集合中一号元素，返回 被删除的元素给name。

* + public int size();获取集合尺寸长度，返回值是集合中包含的元素个数。int size = list.size();获取集合的长度给size。
* 如果希望向集合ArrayList当中存储基本类型，必须使用基本类型对应的“包装类”。

基本类型 包装类（引用类型 包装类都位于java.lang包下）

byte Byte

short Short

long Long

float Float

double Double

int Integer [特殊]

char Character [特殊]

boolean Boolean

从jdk1.5+开始，支持自动装箱、自动拆箱。

自动装箱：基本类型 → 包装类型

自动拆箱：包装类型 → 基本类型

### Static关键字

* 如果一个成员变量使用了static关键字，那么这个对象不再属于对象自己，而是属于所在的类。多个对象共享同一份数据。
* 如果static 用于修饰一个方法，这个方法叫做静态方法。静态方法不属于对象，而是属于类的。



* 如果没有static关键字，那么必须首先创建对象，然后通过对象才能使用它
* 对于static修饰的静态方法，那么不需要创建对象，可以直接通过类名称来调用。
* 无论成员变量，还是成员方法。如果有static，都推荐使用类名称进行调用。
  + - 静态变量：类名称.静态变量
    - 静态方法：类名称.静态方法（）
* 对于本类中的静态方法，可以省略类名称



注意事项：

1. 静态不能访问非静态

原因：因为内存中是【先】有的静态内容，【后】有非静态内容。

1. 静态方法中不能使用this

原因：this代表当前对象，通过谁调用的方法，谁就是当前对象。

### Arrays方法

* Java.util.Arrays是一个与数组相关的工具类，里面提供的大量的静态方法，用来实现数组的常见操作。
  + public static String toString(数组)：将参数数组变成字符串（按照默认格式[元素1,元素2,元素3,....]）

int[] intArray={20,10,30,6}

String intStr = Arrays.toString(intArray)

System.out.println(intStr)//输出结果为[20,10,30,6}

* + public static void sort(数组)：按照默认升序（从小到大）对数组进行排序。

Arrays.sort(intArray)

System.out.println(Arrays.toString(intArray));

//输出结果为[6,10,20.30]

备注：

1. 如果是数值，sort默认升序从小到大
2. 如果是字符串，sort默认按照字母升序

* 将字符串转换成数组：

String str = “wang”;

Char[] char1=Str.toCharArray();

输出char1的值就是一个数组{“w”,”a”,”n”,”g”}

System.out.println(Arrays.toString(char1));

结果为[“w”,”a”,”n”,”g”]

### 继承（extends）

* 特点：
  + 子类可以拥有父类的“内容”，同时也可以拥有自己的“内容”。
  + Java语言是单继承的（一个类的直接父类只能有唯一一个）。
  + Java语言可以多级继承（我有一个父亲，父亲还有一个父亲）。
  + 一个子类的直接父亲是唯一的，但是一个父类可以拥有多个子类。
* 继承关键字extends
  + public class Zi extends Fu{}//子类（Zi）继承父类（Fu）
* 在父子类的继承关系中，创建子类对象，访问成员方法的规则：
  + 创建对象是谁，就优先用谁，如果没有则网上找（若子类和父类中的方法重名，优先使用子类中的方法）

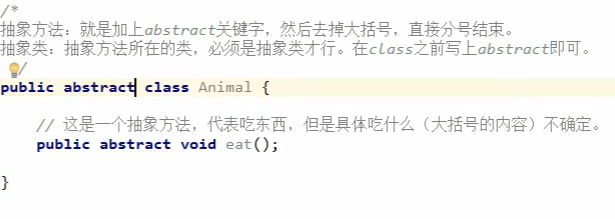
注意事项：无论是成员变量还是成员方法，都优先使用子类中的。如果子类没有，在使用父类中的变量或者方法。

* 重写（Override）：发生在继承关系中
  + 重写（Override）：方法名称一样，参数列表一样。也叫覆盖、复写
  + 重载（Overload）：方法名称一样，参数列表不一样。
* 继承关系中的，父子类构造方法特点：
  + 先调用父类构造方法，后执行子类的构造方法。
* super关键字用法有三种：
  + 在子类的成员方法中，访问父类的成员变量。（super.变量名）
  + 在子类的成员方法中，访问父类的成员方法。（super.成员方法）
  + 在子类的构造方法中，访问父类的构造方法。（super（））

### 抽象类

* 抽象方法的定义：在返回值前加上abstract关键字，然后去掉大括号，直接加分号结束。

注意：抽象方法所在的类，必须是抽象类才行。在class前加abstract即可。



* 如何使用抽象类和抽象方法：
  + 不能直接创建new抽象类对象。
  + 必须用一个子类来继承抽象父类。
  + 子类必须覆盖重写抽象父类中的所有的抽象方法。

覆盖重写（实现）：子类去掉抽象方法的abstract关键字，然后补上大括号。

* + 创建子类对象进行使用。

### 接口

* 接口就是多个类的公共规范。
* 接口是一种引用类型，最重要的内容就是其中的：抽象方法。
* 如何定义一个接口的格式：
  + public interface 接口名称 {

接口内容

}

* 接口中包含以下内容：
  + 常量（成员变量）
    - 接口当中也可以定义“成员变量”，但必须使用public static final三个关键字进行修饰。从效果上看，这其实就是接口是【常量】不可改变因为使用了final进行了修饰。
    - 格式：public static final 数据类型 常量名称 = 数据值；

在接口常量中，可以省略public static final 。例如：int NUM= 10；

其中NUM是一个常量，不可改变。

常量名称完全大写（推荐命名）

* 接口当中的常量必须进行赋值，不能不赋值。
* 在实现类中如何使用，格式：接口名称.常量名；
  + **抽象方法（重点）**
    - **格式：public abstract 返回值类型 方法名（参数列表）；**

**注意：**

1. **接口中的抽象方法，修饰符必须是两个固定的关键字：public abstract**
2. **这两个关键字修饰符，可以选择性地省略。可以省略其中一个，也可以两个都省略。**

**（public 返回值类型 方法名（参数列表）；返回值 方法名（参数列表）；）都是正确格式**

* + - **接口使用步骤：**

1. **接口不能直接使用（例如：接口类名称 名称 = new 接口类名称（）；）这样是错误的，必须有个“实现类”来实现该接口。**

**格式：public class 实现类名称 implements 接口名称{...}。**

**2. 接口实现类必须覆盖重写（实现）接口中所有的抽象方法。**

**实现：去掉abstract关键字，加上方法体大括号。**

**3. 创建实现类的对象，进行使用。**

* + 默认方法
    - 格式：public default 返回值类型 方法名称（参数列表）{

方法体

}

1. 接口的默认方法，可以通过接口实现类对象，直接调用。
2. 接口的默认方法，可以不用被实现类覆盖重写。也可以被覆盖重写。
   * 静态方法
     + 格式：public static 返回值类型 方法名称（参数列表）{

方法体

}

1. 不能通过接口实现类的对象来调用接口当中的静态方法。
2. 正确用法：通过接口名称，直接调用其中的静态方法。

格式：接口名称.静态方法名（参数）；

* + 私有方法

注意：

1.一个类的直接父类是唯一的，但是一个类可以同时实现多个接口

2.格式：public class CImp implements A,B{

//覆盖重写接口A、接口B中的所有抽象方法

}

3.如果实现类所实现的多个接口当中，存在重复的抽象方法，那么只需要覆盖一次即可。

4.如果实现类没有覆盖重写所有接口当中的所有抽象方法，那么实现类就必须是一个抽象类。

5.一个类如果直接父类中的方法，和接口当中的默认方法产生的冲突，悠闲用父类中的方法。

### 多态

* 多态就是一个对象拥有多种形态
* 代码中体现多态性，其实就是父类引用指向子类对象。
  + 格式：父类名称 对象名 = new 子类名称（）；

接口名称 对象名 = new 实现类名称（）；

* 在多态代码中，成员变量访问规则是：
  + 看new的是谁，就优先用谁，没有则向上找。
  + 编译看左边，运行看左边。
* 成员方法

编译看左边，运行看右边。

若在父类中没有的方法，而在子类中有，是不能通过多态进行访问子类中特有的方法。

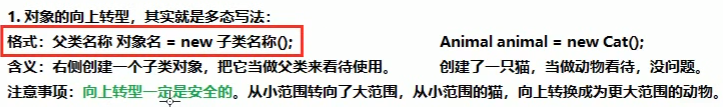
若父类中的方法，在子类中被重写，则多态访问的是子类中重新的方法。

若在子类中没有的方法，而在父类中有，可以通过多态进行访问父类中特有的方法。

* 使用多态的优点：无论右边new的时候换成那个子类对象，等号左边调用的方法都不会变化。

### 对象的向上转型和对象的向下转型

* 对象的向上转型



缺点 ：对象一旦向上转型为父类，那么就无法调用子类原本特有的内容。

解决方案：用对象的向下转型

* 对象的向下转型
  + 对象的向下转型，其实是一个【还原】动作

格式 子类名称 对象名 = （子类名称）父类对象；

* + 含义 将父类对象，【还原】成本类的子类对象；

Animal animal = new Cat();//本来是猫，向上转型成为动物

Cat cat = (cat) animal;//本来是猫，已经被当成动物，在还原成为原来猫

* + 注意事项：

1. 必须保证对象本来创建的时候，就是猫类，才能向下转型成为猫类
2. 如果对象创建的时候本来不是猫类，现在非要向下转型成为猫类，就会报错。

### instanceof关键字

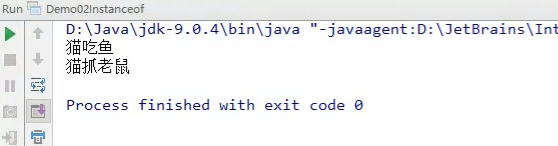
在对象的向下转型中要进行一个判断，如何才能知道一个父类引用的对象，本来是什么子类。这里可以用instanceof关键字

格式：对象 instanceof 类型。

这将得到一个boolean值结果，也就是判断前面的对象能不能当做后面类型实例。



上图程序将执行第二个if语句，也就是完成猫类的向下转型，其结果为：



1. final 关键字代表最终、不可改变的。

常见四种方法：

* 可以用来修饰一个类
  + 格式：public final class 类名称{。。。}

含义：用final修饰的当前这个类不能有任何子类。

注意：一个类如果是final，那么其中的所有的成员方 法都无法覆盖重写（因为没有子类，所有不能覆盖重写）

* 可以用来修饰一个方法
  + 用final关键字来修饰一个方法的时候，这个方法就是最终方法，也就是不能被覆盖重写。
  + 格式：final 返回值类型 方法名称（参数列表）{方法体}

注意：用final来修饰的方法，该方法在子类中不能被覆盖重写

对于类、方法来说，abstract关键字和final关键字不能一起使用， 因为矛盾。

* 还可以用来修饰一个局部变量
  + 一旦用final关键字来修饰局部变量，那么这个变量的值就不能在更改（变成常量）。（一次赋值，终生不变）
* 还可以用来修饰一个成员变量
  + 用final关键字来修饰成员变量，那么这个变量的值就不能在更改

1.由于成员变量具有默认值，所以用了final之后必须手动赋值，不会再给默认值。

2.对于final的成员变量，要么使用直接赋值，要么通过构造方法赋值。

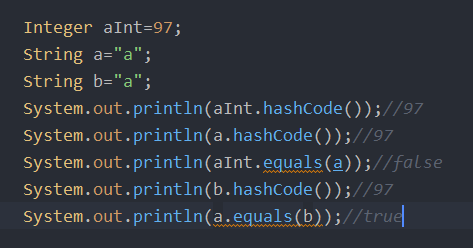
3.必须保证类当中所有重载的构造方法，都最终会对final的成员变量进行赋值。

### 28. hashcoud()和equals()

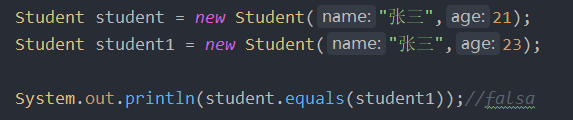
举例： 比较两个对象是否相等（对象的成员变量相同，认为两个对象是相同的）

结论1：两个对象的hashcoud()内容比较相等，但是equals()去比较不一定相等，也就是说hashCode()不是绝对可靠。

结论2：两个对象内容通过equals()比较相等，那么hashCoud()肯定相等。



在创建对象时使用equals进行比较的是创建对象的内存地址，（equals不进行重写，默认）



虽然两个对象的内容相等，但是创建对象的内存地址不相等

## 三 java升级

### Java中四种权限修饰符

级别由高到低：public > protected > (default) > private

### 内部类

* 分类：
  + 成员内部类
    - 定义格式：修饰符 class 外部类名称{

修饰符 class 内部类名称{

/....

}

}

* + 如何使用成员内部类？有两种方式
    - 间接方式：在外部类的方法中，使用内部类；然后main只是调用外部的方法。
    - 直接方式：公式，外部类名称.内部类名称 对象名 = new 外部类名称（）.new 内部类名称（）；

注意：内部类用外，部类可以随意访问；外用内，一定需要借助内部对象。

* + 内部类访问外部类的成员变量，如果变量重名。

公式：外部类名称.this.成员变量名

* + 局部内部类（包含匿名内部类）
    - 如果一个类是定义在一个方法的内部，那么这就是一个局部内部类。只有当前所属的方法才能使用它，出了这个方法外面就不能再用了。
    - 定义格式：修饰符 class 外部类名称{

修饰符 返回值类型 外部类方法名称（参数列表）{

class 局部类名称{

//...

}

}

}

注意：局部内部类修饰符什么都不写！！

* + 匿名内部类
    - 定义格式：接口名称 对象名 = new 接口名（）{

//覆盖重写所有的抽象方法

} ；

* + - 对格式进行解析

1. new代表创建对象的动作
2. 接口名称就是匿名内部类需要实现那个接口
3. {.....} 这是匿名内部类的内容。

注意： 1. 匿名内部类在创建对象时，只能使用唯一的一次。

### 类作为成员变量

## 四Object类 常用API

### Object类

* 概述：java.lang.Object（不必导入直接就可以用）类是Java语言中的根类，即所有类的父类。它中描述的所有方法子类都可以使用。在对象实例化的时候，最终找到的父类就是Object。（如果一个类没有指定父类，那么默认则继承Object类）

例如：public class text /\* extend Object\*/{.....}

* toString() 用于返回字符串
  + 直接打印对象的名字其实就是调用对象的toString方法
  + 看一个类是否重写了toString，直接打印这个类的对象即可，如果没有重写toString方法那么打印的是对象的地址值
* equals方法
* 调用成员方法equals并指定参数为另一个对象，则可以判断这两个对象是否相同。这里的“相同”有默认和自定义两种方式。

格式：boolean b1= s1.equals(s2);（s1与s2不能为空，否则报错）

* equals返回值类型为布尔类型（返回值为True或者False）
* Objects类中的equals方法：对两个对象进行比较，可以防止空指针异常。

格式:boolean b2 = Objects.equals(s1,s2);

### Date类

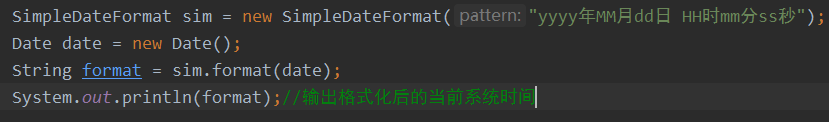
Java.util.Date类表示日期和时间的类，精确到毫秒（千分之一秒，1000毫秒=1秒）

* DateFormat类对日期进行格式化，我们通过这个类可以帮我们完成日期和文本之间的转换，也就是可以在Date对象与String对象之间进行来回转换。
  + DateFormat类中的成员方法：

String format（Date date）按照指定的模式，把Date日期，格式化为符号模式的字符串。

Date parse（String source）把符号模式的字符串，解析为Date日期。

* + DateFormat类是一个抽象类无法之间创建对象使用，可以使用该类的子类SimpleDateFormat.
  + SimpleDateFormat类的构造方法：
    - SimpleDateFormat（String pattern）
    - 例如：SimpleDateFormat（String pattern）
    - SimpleDateFormat（“yyyy年MM月dd日 HH时mm分ss秒”）



格式 pattern严格区分大小写

y 年

M 月

d 日期

H 时

m 分

s 秒

* + 使用DateFormat类中的方法parse，把文本解析为日期
    - 使用步骤：

1.创建SimpleDateFormat对象，构造方法中传递指定模式

2.调用SimpleDateFormat对象中的构造方法parse，把符号构造方法中的字符串解析为Date日期

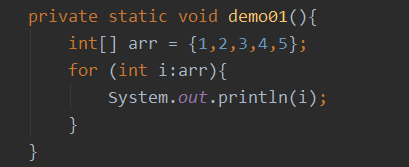
### 3.增强for循环

* 增强for循环：底层使用的也是迭代器，使用for循环的格式，简化了迭代器的书写，是jdk1.5之后出现的新接口。
* 增强for循环：主要是用来遍历集合和数组
  + 格式：for（集合/数组的数据类型 变量名:集合/数组名）{

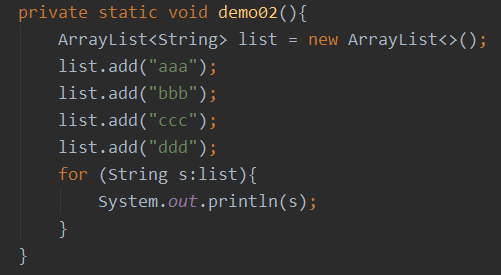
输出变量名；

}

使用增强for循环遍历数组



使用增强for循环遍历集合



注意事项：新for循环必须有被遍历的目标。目标只能是Collection或者是数组。新式for仅仅作为遍历操作出现。

### 4.Iterator迭代器

Java.util.Iterator即Collection集合元素的通用获取方式。在取出之前先要判断集合中有没有元素，如果有，就把这个元素取出，继续在判断，如果还有就在取出来。一直把集合中的元素全部取出来。这种取出方式专业术语称为迭代。

两个常用方法：

Boolean hasNext（）判断集合中还有没有下一个元素，有则返回true， 没有则返回false。

E next() 取出集合的下一个元素

Iterator迭代器是一个接口，我们无法直接使用，需要使用Iterator接口的实现类对象，获取实现类的方式比较特使。Collection接口中有一个方法叫做Iterator（）；这个方法返回的就是迭代器的实现类对象。

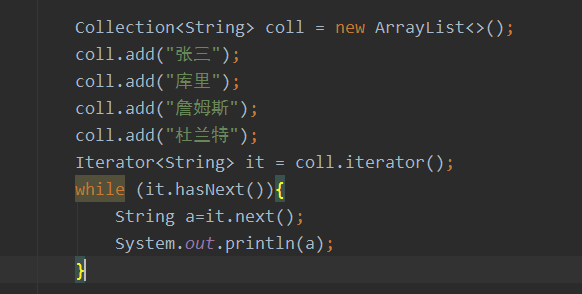
迭代器的使用步骤：

使用集合中的方法iteration（）获取迭代器的实现类 对象，使用Iteration接口接收（多态）。

使用Iterator接口的方法hasNext判断还有没有下一个元素。

使用Iterator 接口中的方法next取出集合的下一个元素。

Iterator<E>接口也是有泛型的，迭代器的泛型跟着集合走，集合是什么泛型，迭代器就是什么泛型。



### 5.泛型概述

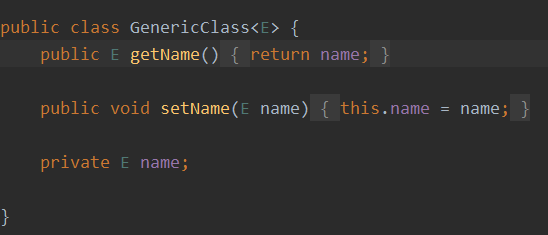
在使用集合的过程中数据的存储类型就是泛型，泛型就是一种未知的数据类型，当我们不知道使用什么数据类型的时候，可以使用泛型。泛型也可以看出是一个变量，用来接收数据类型。

当创建集合对象的时候，就会确定泛型的数据类型，例如集合的固定格式ArrayList<E> 集合名称 = new ArrayLis<>();（其中E可以为任意大写字母），泛型的默认类型是Object类型。

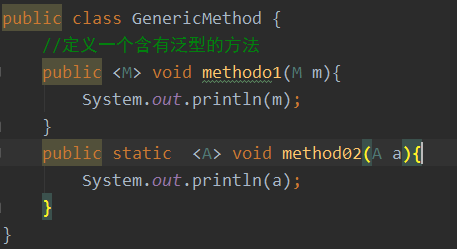
当集合的格式定义为ArrayList<String> 集合名称 = new ArrayLis<>();时会将String类型赋值给泛型E;

定义一个含有泛型的类，模拟ArrayList集合：

* + - 泛型可以是一个未知类型的数据类型，当我们不确定什么类型的时候，可以使用泛型。
    - 泛型可以接收任意数据类型，可以使用Integer，String，Student...
    - 创建对象的时候确定泛型的数据类型。

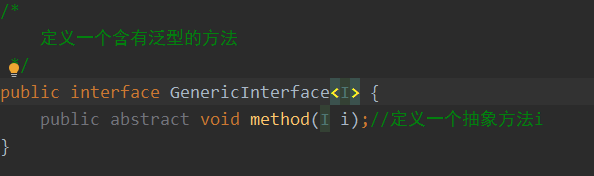


使用一个含有泛型的方法

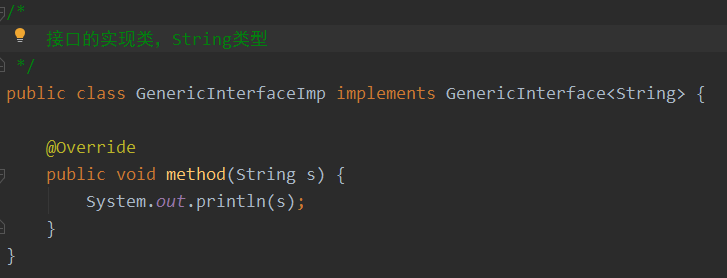


使用含有泛型的接口

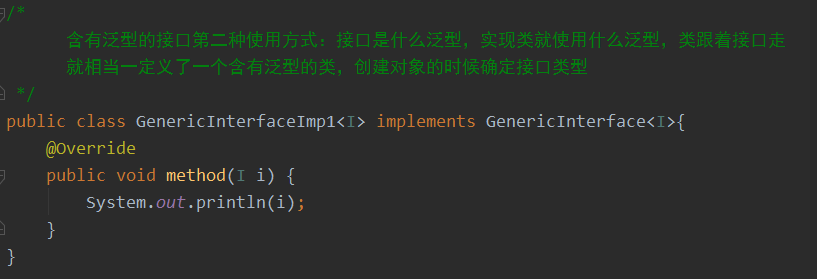
定义一个接口



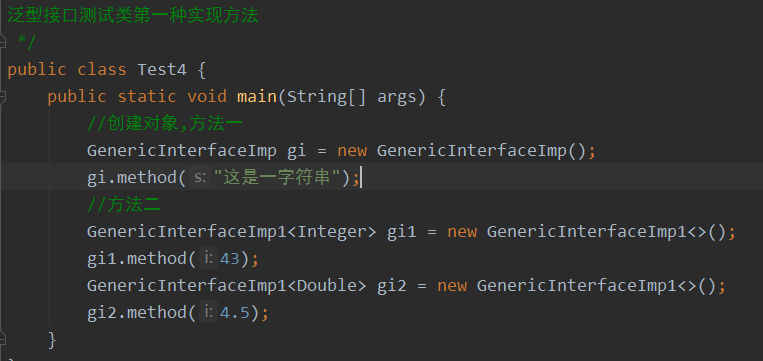
定义实现类方法一（在实现类中确定泛型类型）



定义实现类方法二（在创建对象时确定泛型的类型）



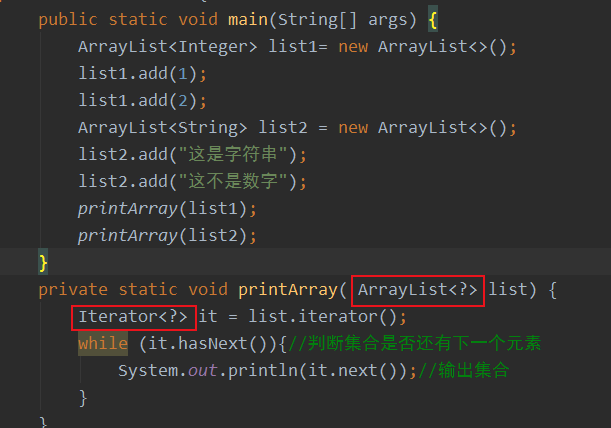
定义测试类

 泛型通配符：

定义一个方法，能遍历所有类型的ArrayList集合。

这时候我们不知道ArrayList集合使用什么类型数据，可以泛型的统配符？来接收数据类型（通配符是不能在定义或者创建对象时使用的）

泛型格式：用？来替代尖括号中的泛型。



## 五 数据结构

### 数据结构—链表

特点：

查询慢：链表中地址不是连续的，每次查询元素，都必须从头开始查询。

增删快：链表结构，增加/删除一个元素，对链表整体结构没有影响。

## 六 List集合及其子类

### List集合

Java.util.list接口继承自Collection接口，习惯行地会将实现了List接口的对象成为List集合。在List集合中允许出现重复的元素，所有的元素是以一种线性方式进行存储的，在程序中可以同索引来访问集合中指定的元素。另外，List集合还有一个特点就是元素有序，即元素的存入顺序和取出顺序一只一致。

List接口有三大特点：

1. 有序的集合，存储元素和取出元素的顺序是一致的（存123，取123）
2. 有索引，包含一些带索引的方法
3. 允许存储重复的元素

List接口中带索引的方法（特有的方法）：

1. public void add（int index, E element）：将指定元素，添加到该集合 的制定位置上。
2. public E get(int index):返回集合中指定位置的元素。
3. public E remove(int index)：移除列表中指定位置的元素，返回的是被 移除的元素。
4. public E set(int index,E element):用指定元素替换集合中指定位置元 素，返回值是更新前的元素。

注意：

操作索引的时候，一定要防止索引越界异常。



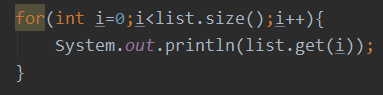
在创建List对象时注意：

1.由于在java.util里面List是一个接口，所以不能直接初始化，所以会编译错误（错误格式：List<String> list = new List<>();）

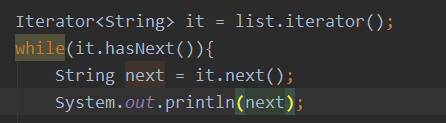
2.你可以List l =new ArrayList();这样是可以的，因为 ArrayList是实现List 接口我们一般 使用List都是new ArrayList();

使用List集合遍历有3种方法：

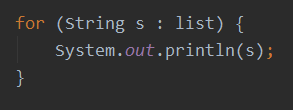
1. 使用普通的for循环遍历数组



1. 使用迭代器遍历数组



1. 使用增强for循环遍历数组



直接输出集合的名字，输出的是集合中的元素

### ArrayList集合

Java.util.ArrayList集合数据存储结构是数据结构。元素增删慢，查找快，由于日常开发中使用最多的功能为查询数据，遍历数据，所以ArrayList是最常用的集合。

特性及使用前面有记录

### LinkedList集合

java.utlil.LinkedList集合 implement List接口

LinkedList集合特点：

1. 底层是一个链表结构：查询慢，增删快。
2. 里面包含大量首位元素的方法

注意：使用LinkedList集合特有方法，不能使用多态

LinkenList特有方法：（List方法中所有的特性该方法都有）

1. public void addFirst(E e):将指定元素插入此列表的开头。
2. Public void addLast(E e):将指定元素添加到此列表的结尾。此方法等效于add方法
3. Public void push(E e):将元素推入此列表所表示的堆栈中,此方法等效于addFirst(E e)
4. public E getFirst(E e):返回此列表的第一个元素。
5. public E getLast(E e):返回此列表的最后一个元素。
6. public E removeFirst(E e):移除并返回此列表的第一个元素
7. public E removeLast(E e):移除并返回此列表的第一个元素
8. public E pop(E e):从此列表的堆栈处弹出一个元素
9. public boolean isEmpty():如果列表不包含元素，则返回true。

## 七Set接口

java.util.Set接口和java.util.List 接口一样，同样继承Collection接口。与List接口不同的是，Set接口中的元素无序，并且都会一某种规则保证存入的元素不重复。

Set接口特点：

1. 不允许存储重复元素。
2. 没有索引，没有带索引的方法，不能使用普通的for循环遍历。只能使用迭代器或者增强for循环遍历set集合。

Set集合有多个子类，这里我们介绍其中的java.util.HashSet、java.util.LinkedHashSet这两个集合。

### HashSet类

Java.util.HashSet类 implements Set接口

1. HashSet接口特点：
2. 不允许存储重复元素。
3. 没有索引，没有带索引的方法，不能使用普通的for循环遍历。
4. 是一个无序的集合，存储集合和取出元素顺序有可能不一致
5. 底层是一个哈希表结构（查询速度非常快）
6. 哈希值：是一个十进制整数，由系统随机给出（就是对象的地址值，是一个逻辑地址，是模拟出来的地址，不是数据实际存放的物理地址）在Object类有一个方法，可以获取对象的哈希值。
7. 如何获取对象的哈希值

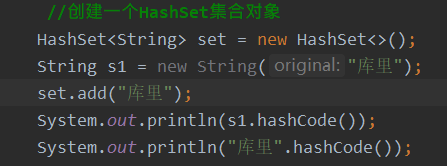
在Object类有一个方法，可以获取对象的哈希值

使用Int hashCode() 方法返回该对象的哈希值，格式：

对象名.hashCode（）



一下两种方式获取的哈希值一样（一种是创建字符串集合另一种是直接添加）



1. 哈希表特点：查询速度快

Jdk1.8版本之后:

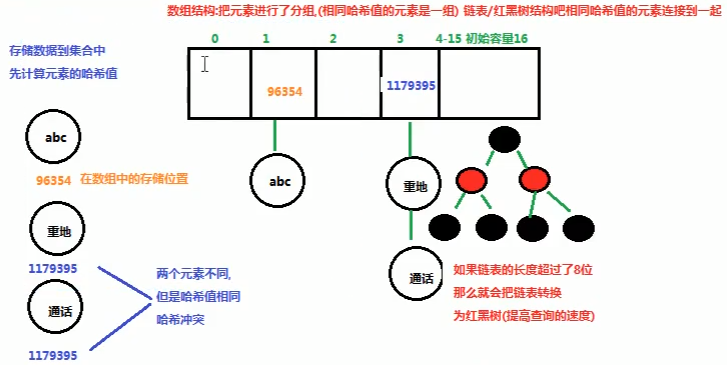
哈希表=数组+链表（横着是数组结构，竖着是链表或者红黑树）

哈希表=数组+红黑树（提高查询速度）

数组结构：把元素进行了分组（相同哈希值的元素是一组）

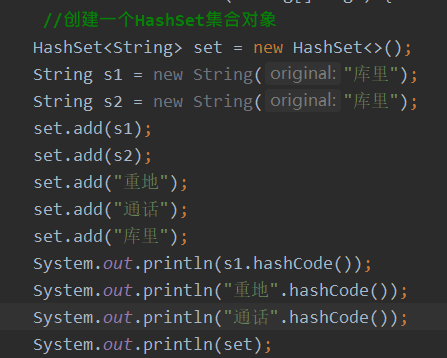
链表/红黑树结构：吧相同哈希值的元素连接到一起

如果链表的长度超过8位，那么就吧链表转化成红黑树（提高查询速度）



1. Set集合不允许存储重复元素原理：

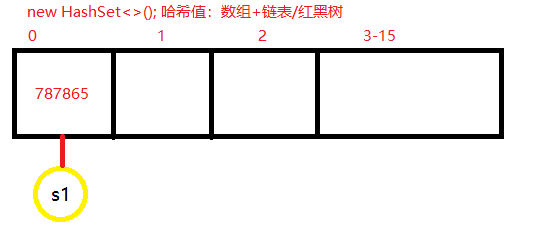
首先创建HashSet集合对象



Set集合在调用add方法的时候，add方法会调用元素的hashCode方法和equals方法判断元素是否重复

1. set.add(s1)

add方法会调用s1的hashCode方法，计算字符串”库里”的哈希值，哈希值是787865，在集合中查找有没有787865这个哈希值的元素，在数组中发现没有，就会吧s1的哈希值存入集合中，s1的值就挂在哈希值对应集合的下面。



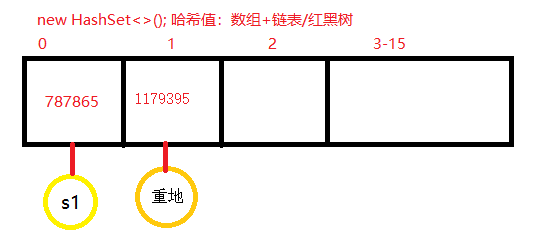
1. set.add(s2)

add方法会调用s2的hashCode方法，计算字符串”库里”的哈希值，哈希值是787865，在集合中查找有没有787865这个哈希值的元素，发现有（哈希冲突）。

s2会调用equals方法和哈希值相同的元素进行比较。s2.equals(s1)返回true，两个相同哈希值元素相同，不会吧s2存储到集合中。

1. set.add(重地)

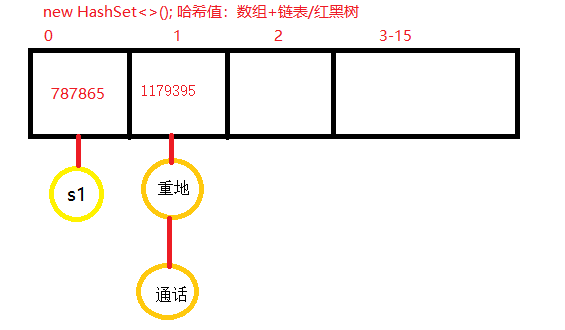
add方法会调用”重地”的hashCode方法，计算字符串”重地”的哈希值，哈希值是1179395，在集合中查找有没有1179395这个哈希值的元素，在数组中发现没有，就会吧”重地”的哈希值存入集合中，”格林”就挂在哈希值对应集合的下面。



1. set.add(通话)

add方法会调用”通话”的hashCode方法，计算字符串”通话”的哈希值，哈希值是1179395，在集合中查找有没有1179395这个哈希值的元素，在数组中发现有（哈希冲突），

“通话”会调用equals方法和哈希值相同的元素进行比较。”通话”.equals(重地)返回false，两个相同哈希值的元素不同，就会吧通话存入集合中。就挂在哈希值对应集合的下面。



### LinkedHashSet类

LinkedHashSet是HashSet的一个子类

Java.util.LinkedHashSet集合 extends HashSet集合

LinkedHashSet集合特点

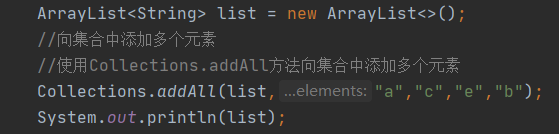
底层是一个哈希表（数组+链表/红黑树）+链表：多了一个链表用来记录元素的存储顺序，保证元素有序（输入顺序和存储顺序一致）。而HashSet存储的元素是无须的（输入顺序和存储顺序不一致）。

## 八 Collection

### java.utils.Collections是集合工具类，用来对集合进行操作。

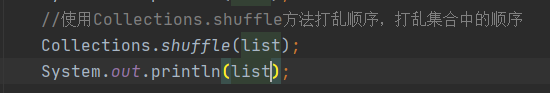
### 部分方法如下：

1. Collections.addll(集合名,集合元素1,集合元素2,…..) 往集合中添加一些元素



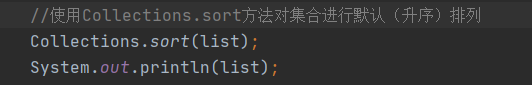


1. Collections.shuffle(集合名称) 打乱顺序：随机打乱集合顺序。





1. Collections.sort(集合名) 将集合中元素按照默认规则排序（默认升序）





注意：

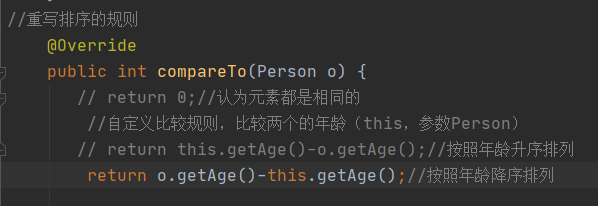
sort（list<T> list）方法的使用前提是被排序的集合存储的元素，使用自定义方法，必须实现Comparable,重写接口中的方法compareTo方法

Comparable接口的排序规则：

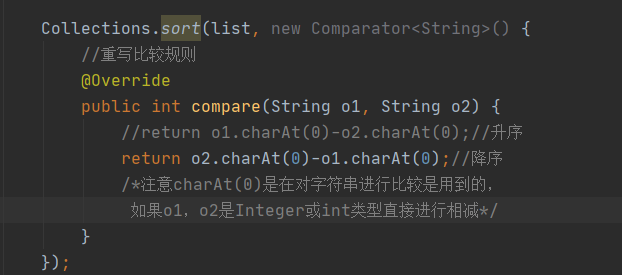
自己（this）-参数；升序

参数-自己（this）;降序





1. 自定义排序格式：Collections.sort(集合名,new Comparator<T>)



## 九 Map集合

Java.util.Map<k,v>集合

### 1. Map集合特点：

1. Map集合是一个双列集合，一个元素包含两个值（key，value） 。
2. Map集合中的元素，key和value的数据类型可以相同也可以不同。
3. Map集合中的元素，key是不可以重复的，value是可以重复的。
4. Map集合中的元素，key和value是一一对应的。

Java.util.HashMap<k,v>集合 implement Map<k,y>接口

### HashMap集合特点：

1. HashMap集合底层是哈希表，查询速度快。
2. HashMap集合是一个无序集合，存储元素和取出元素顺序有可能不一致。

Java.util.LinkedHashMap<k,y>集合 extends HashMap<k,y>集合.

### 3. LinkedHashMap集合特点；

1. LinkedHashMap底层是一个哈希表+链表（保证迭代顺序）。

2. LInkedHashMap集合是一个有序集合，存储元素和取出元素的顺序是一致的。

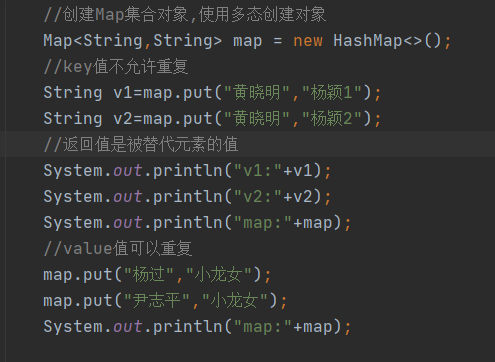
### 4. put方法（添加）

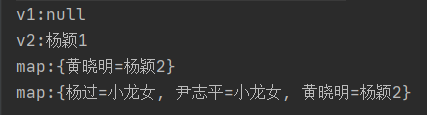
格式：对象名.put(K key，V value)把指定的键与指定的值添加到Map集合中

返回值：v

存储键值对的时候，key不重复，返回值V是null

存储键值对的时候，key重复，会使用新的value替换Map中重复的value，返会被替换的value值。





### 5. remove方法（删除）

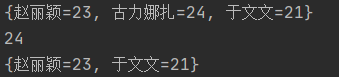
格式：对象名.remove(Object key):把指定的键 所对应的键值对元素 在Map集合中删除，返回被删除元素的值。

返回值：V

Key存在，v返回被删除的值

Key不存在，v返回null。





### 6. get方法

用途：通过指定键，获取Map集合中的值

格式：对象名.get(Object key)

返回值 V；

### 7. containskey方法

用途：判断集合中是否包含指定的键

格式：对象名.containskey(Object key);

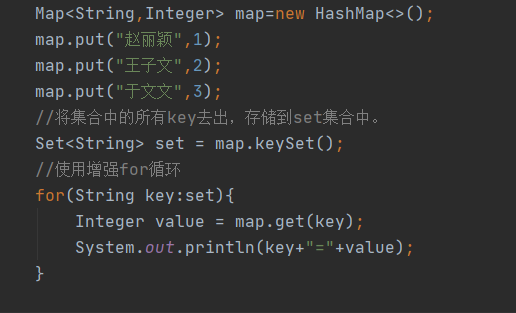
返回值是布尔类型 存在返回true,不存在返回false。

### 8. Map集合遍历键找值方法

（1）Map集合中的第一种方式：KeySet（）方法，通过键找值。

实现步骤：

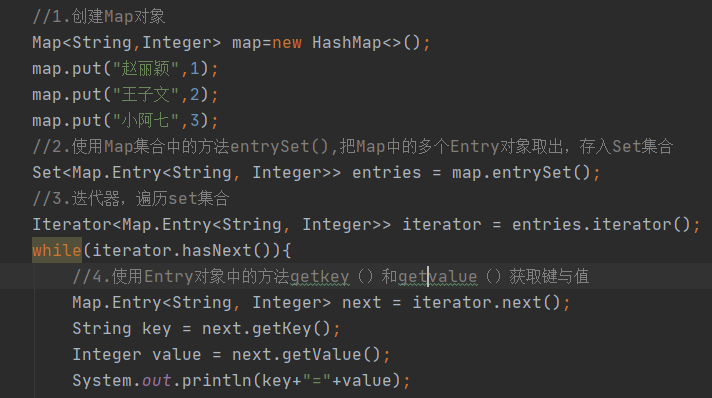
1. 使用Map集合中的方法keySet()把Map集合中的所有的key取出来，存储的set集合中。
2. 遍历set集合，获取Map集合中的每一个key。
3. 通过Map集合中的get（key）方法，通过key找到value。



（2）Map集合中的第二种方式：使用Entry对象遍历

实现步骤：

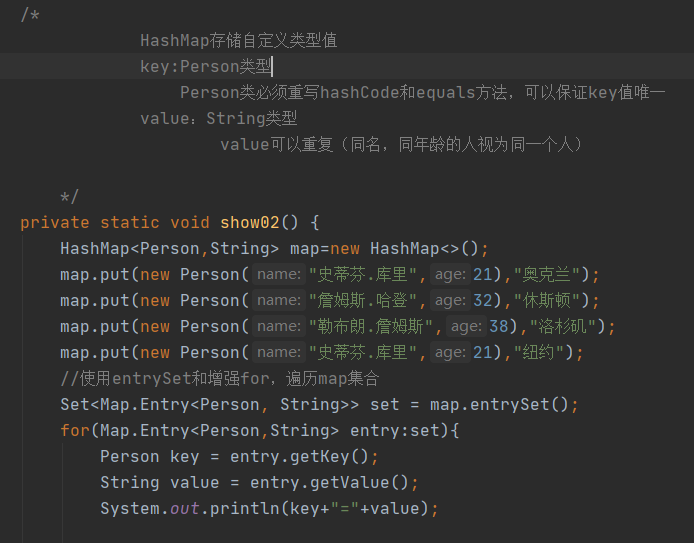
1. 使用Map集合中的方法entrySet（），把Map集合中的多个Entry对象取出来，存储到set集合中。
2. 遍历Set集合，获取每一个Entry对象。
3. 使用Entry对象中的方法getkey()和getvalue()获取键与值。

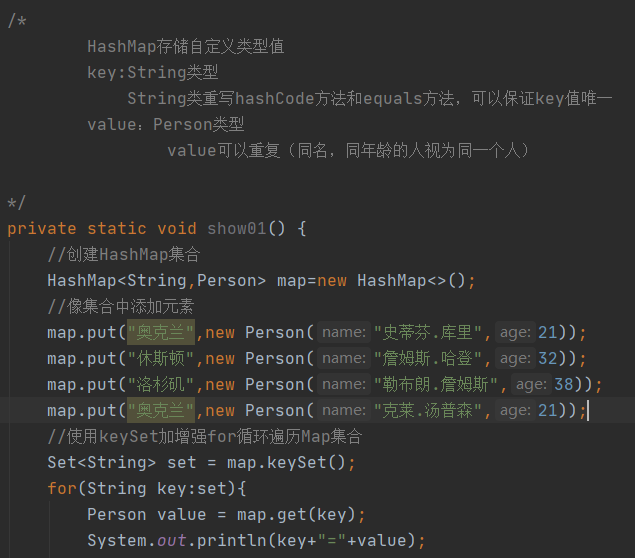


### 9. 使用HashMap存储自定义的键值

Map集合保证key是唯一的：

作为key元素，必须重写hashCode方法和equals方法，保证key唯一





## 十 异常

1. Java.lang.Throwable:类是java语言中所有错误或者异常的超类。
2. 在Throwable中有两个直接子类。
   1. Exception异常类：

Exception类及其子类：编译期异常，进行编译java程序出现的问题。

RuntimException：运行期异常，java程序运行过程中出现的问题。

* 1. Error错误：代表程序中产生的错误。

## 十一 异常处理

Java 异常处理的五个关键字：try ,catch, finally, throw, throws

### throw关键字

作用：

可以使用throw关键字在指定的方法中抛出指定异常

使用格式：

throw new xxxException(“异常的原因”);

注意：

1. throw关键字后面new的对象必须是Exception或者Exception的子类对象。
2. throw关键字抛出指定的异常对象，我们就必须处理这个异常对象

1). throw关键字后面创建的是（RuntimException）运行时异常或者是RuntimException的子类对象，我们可以不处理，默认交给JVM处理（打印异常对象，中断程序）。

2). throw关键字后面创建的是（Exception）编译异常（写代码的时候报错），我们就必须处理这个异常，要么throw，要么try...cath.

### 2. 声明异常关键字throws

2.1 作用：

当方法内部抛出异常对象的时候，那么我们就必须处理中异常对象可以使用throws关键字处理异常对象，会吧异常对象抛出给方法的调用者处理，最终交给JVM处理（JVM处理是中断处理）

2.2使用格式：在方法声明是使用

修饰符 返回值类型 方法名（参数列表）throws AAAException BBBException {

throws new AAAException (“产生原因”);

throws new BBBException (“产生原因”);

.....

}

注意事项：

1. throws 关键字必须写在方法声明处
2. throwa 关键字后面声明的异常必须是Exception或者是Exctpion的子类
3. 方法内部如果抛出了多个异常对象，那么throws 后面必须也声明多个异常。如果抛出的多个对象有子父类关系，那么直接声明父类异常。
4. 调用的一个声明抛出异常的方法，我们就必须的处理声明异常，要么继续使用throws 声明抛出，交给方法的调用者，最终交给JVM，要么try....cath自己处理异常。
5. 如果抛出的多个异常对象有子父类关系，那么直接声明父类异常即可。

### 3. 捕获异常try...catch

3.1 try...catch 异常处理的第二中方式，自己处理异常。

3.2 格式：

try{

可能产生异常的代码

}catch（定义一个异常变量，用来接收try中抛出的异常对象）{

\\try中抛出什么异常，catch中就定义什么类型变量

\\类型必须时Exception类或其子类

异常处理逻辑，异常对象之后，怎么处理异常对象

一般在工作中，会把异常的信息记录到一个日志中

}

....catch(异常类名 变量名){

}

注意：

1. try中可能会抛出多个异常对象，那么就可以使用多个catch来处理这些异常对象。（try...catch中try只能有一个，而catch只能有一个。）
2. 如果try中产生了异常，那么就会执行catch中的异常处理逻辑，执行完毕后，继续执行try...catch之后的代码。
3. 如果try中没有产生异常，那么就不会执行catch中异常的处理逻辑，执行完try中的代码，继续执行try...cath之后的代码。
4. try中抛出什么异常，catch就定义什么变量，用来接收这个异常。

3.3 throws和try...catch的区别：

使用throws关键字，程序一旦发生异常，如果没有被处理，程序就会发生正常终止。所以当声明throws时，就必须在程序中对异常进行处理。

try...catch，catch代码对异常进行处理完毕后，程序仍会向下执行，而不会异常终止。

## 十二 多线程

### 主线程：执行主方法（main）的线程,

单线程程序：java程序中只有一个线程，执行从main方法开始，从上到下依次执行。

### 创建多线程程序的第一种方式：创建Thread类的子类

Java.lang.Thread类，是描述线程的类，我们想要使用多线程程序，就必须继承Thread类。

实现步骤：

1. 创建一个Thread类的子类（继承Thread）
2. 在Thread的子类中重写Thread类中的run方法，设置线程任务。
3. 创建Thread的子类对象。
4. 调用Thread类中的方法start方法，开启新的线程，执行run方法。

void start()使用该线程开启执行；java虚拟机调用该线程的run方法。（结果是两个线程并发的运行；当前程序（mian线程）返回给start方法和另一个线程（创建的新线程，执行其run方法）。多次启动一个线程是非法的。特别是当线程已经结束执行后，不能在重写启动）。

Java程序属于抢占式调度，那个线程的优先级高，那个线程优先执行；同一个优先级，随机选择一个执行。

### Thread类

3.1常用方法：

* 获取线程的名称：

1. 使用Thread类中的方法getName（）

String getName() 返回该线程的名称。

1. 可以先获取到当前正在执行的线程，使用线程中的方法getName （）获取线程的名称.

* Public static void sleep(long millis):是当前正在执行的线程以指定的毫秒数暂停（暂时停止执行）。毫秒数结束之后，线程继续执行。

例如：Thread.sleep(1000);程序休眠1s后执行；

### 创建多线程程序的第二种方式：实现Runnable接口

在使用多线程时一般使用第二种方式

Java.long.Runnable

Runnable 接口应该有那些打算通过某一线程执行其实例的类来实现。类必须定义一个称为run的无参方法。

4.1实现步骤：

1.创建一Runnable接口的实现类

2.在实现类中重写Runnable接口的run方法，是指线程任务。

3.创建一个Runnnabel接口的实现类对象

4.创建Thread类对象，构造方法中传递Runnable接口的实现类对象

5.调用Thread类中的start方法，开启新线程执行run方法。

实现Runnable接口创建多线程程序的好处：

1. 避免了单继承的局限性

一个类只能继承一个类（一个人只能有一个亲爹），类继承Thread类就不能继承其他的类。实现了Runnable接口，还可以继承其他的类，实现其他的接口。

1. 增强了程序的扩展性，降低的程序的耦合性（接除耦合性）

实现Runnable接口的方式，把设置线程任务和开启新线程进行的分离，实现类中重写了run方法；用来设置线程任务。

创建Thread类对象，调用start方法：用来开启新线程

### 使用匿名内部类实现线程的创建

匿名：没有名字

内部类：写在其他类内部的类

匿名内部类：简化代码

把子类继承父类，重写父类的方法，创建子类对象合成一步完成

把实现类实现类接口，重写接口中的方法，创建实现类对象合成一步完成。

匿名内部类的最终产物：子类/实现类对象，而这个类没有名字。

格式：

new 父类/接口（）{

重写父类/接口中的方法

}；

### 线程中的安全

单线程程序是不会出现线程安全问题

多线程程序，没有访问共享数据不会产生安全问题

多线程访问了共享的数据，会产生线程安全问题

如何避免线程中出现安全问题：

我们可以让一个线程在访问共享数据源的时候，无论是否失去了cpu的执行权；都要让其他线程等待，等待当前线程执行完毕后。其他线程才能占有cpu的使用权。

### 线程同步

使用线程同步可以解决线程中的安全问题

为了保证每个线程都能正常执行原子操作，java引入了线程同步机制。有三种方式完成同步操作：

1. 同步代码块
2. 同步方法
3. 锁机制

7.1 同步代码块

同步代码块是解决线程安全的一种方案，

格式：

Synchronized（锁对象）{

可能会出现的线程安全问题的代码（访问了共享数据的代码）

}

步骤：

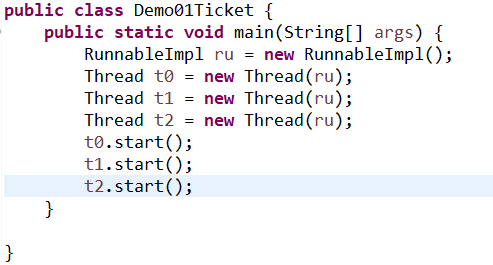
1. 创建一个锁对象
2. 创建一个同步代码块
3. 将访问数据源的代码放入，同步代码块中

注意：

1. 通过代码块中的锁对象，可以使用任意对象
2. 但是必须保证多个线程使用的锁对象是同一个
3. 锁对象作用：

把同步代码块锁住，只让一个线程在同步代码块中执行





同步技术原理：

使用了一个锁对象，这个锁叫做同步锁，也叫对象锁，也叫对象监视器

3个线程一起抢夺cup的执行权，谁抢到了谁执行run方法。如果t0抢到了cpu的执行权，执行run方法，遇到synchronized代码块，

这时t0会检查synchronized代码块是否有锁对象，发现有，就会获取到锁对象，进入到同步中执行。

T1会检查synchronized代码块是否有锁对象，发现没有，t1就会进入阻塞状态，会一直等待t0线程归还锁对象。一直到t0线程执行完同步中的代码，会把锁对象归还给同步代码块，这时候t1才能获取锁对象进入同步中。

总结：同步中的线程，没有执行完毕不会释放锁，同步外的线程没有锁进不去同步，进入阻塞状态等待获取cup执行。

7.2同步方法

解决线程安全问题的第二种方法：使用同步方法

使用步骤：

1. 把访问共数据源的代码抽取出来，放到一个方法中
2. 在方法中添加一个synchronized修饰符

格式：定义方法的格式

修饰符 synchronized 返回值类型 方法名（参数列表）{

可能会出现的线程安全问题的代码（访问了共享数据的代码）

}



7.3Lock锁

解决线程安全的第三种方案：Lock锁

java.utli.concurrent.locks.Lock接口

Lock实现提供了比使用synchronized方法和语句可获得的更广泛的锁操作。

Lock接口中的方法：

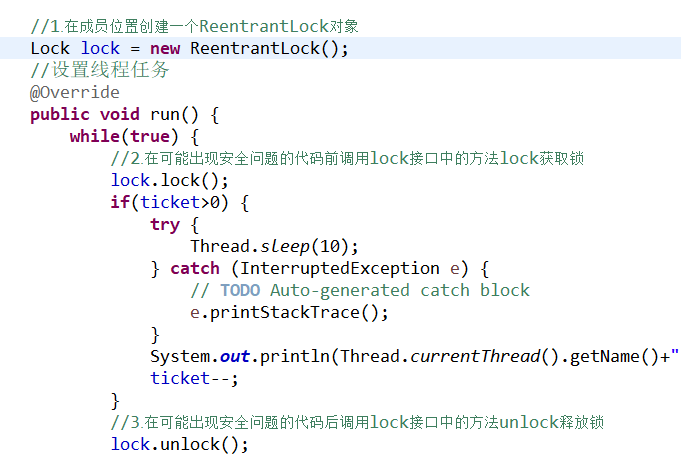
void lock()获取锁

void unlock()释放锁

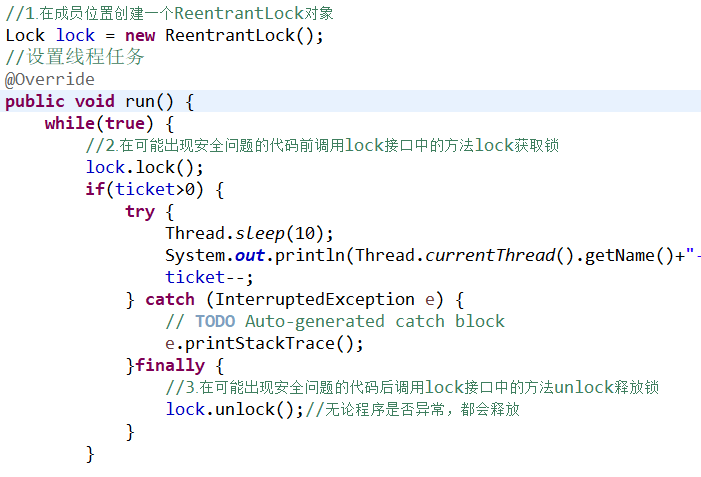
使用步骤：

1. 在成员位置创建一个ReentrantLock对象
2. 在可能出现安全问题的代码前调用lock接口中的方法lock获取锁
3. 在可能出现安全问题的代码后调用lock接口中的方法unlock释放锁

方法一

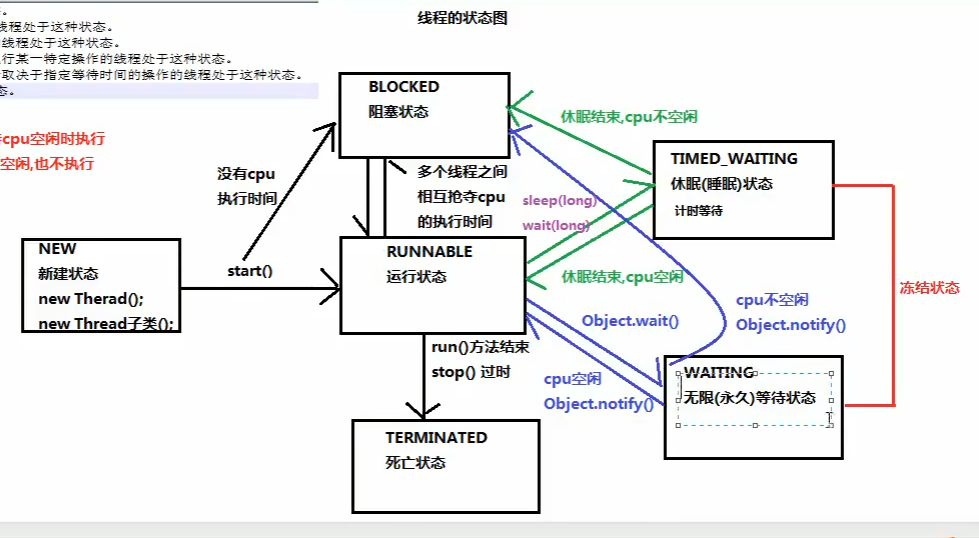


方法二（性能更高）



### 8.线程的状态

新建、运行、阻塞、休眠、无线等待、死亡。



### 9.等待唤醒

1. 等待唤醒案例：线程之间的通信

1. 创建一个顾客线程（消费者）：告知老板要的包子的种类和数量，调用wait方法，放弃cpu的执行，进入waiting状态（无线等待）。

2. 创建一个老板进程（生产者）：用了5秒钟做包子，做好之后，调用notify方法，唤醒顾客吃包子。

注意：

1. 顾客和老板进程必须使用同步代码块包裹起来，保证等待和唤醒只能有一个执行。
2. 同步使用的锁必须保证是唯一的
3. 只有锁对象才能调用wait和notify方法，非锁不能调用。

创建锁对象

Object obj = new Object（）;

锁对象名.wat(); 使线程进入等待状态。

锁对象名.notify(); 唤醒等待的进行。

### Object类中wait带参数方法和notify

进入到计数等待有两种方式

1. 使用sleep(long m )方法，在毫秒值结束之后，线程睡醒进入到Runnable/Blocked状态。
2. 使用wait(long m)方法，wait方法如果在毫秒结束之后，还没有被notify唤醒来，线程睡醒进入到Runnable/Blocked状态。

唤醒的方法

1. notify()唤醒在对象监视器上等待的单个线程。
2. notifyAll()唤醒在对象监视器上等待的所以线程。

## 十三 等待唤醒机制

### 1．线程之间的通信

概念：

多个线程在处理同一个资源，但是处理的动作（线程的任务）却不相同。

为什么要处理线程之间的通信

多个线程并发执行时，在默认情况下CPU是随机切换线程的，当我们需要多个线程共同完成一件任务，并且我们希望他们有规律的执行，那么多线程之间需要一些协调通信，以此来帮助我们达到多线程共同操作一份数据。

## 十四 反射

十五 动态代理

### 什么是代理模式？

代理模式是常用的java设计模式，他的特征是代理类与委托类有同样的接口，代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后处理消息等。简单的说就是，我们在访问实际对象时，是通过代理对象来访问的。

### 什么是动态代理？

代理类在程序运行时创建的代理方式被成为动态代理。

* 创建一个InvocationHandler对象

//创建一个与代理对象相关联的InvocationHandler

InvocationHandler stuHandler = new MyInvocationHandler<Person>(stu);

* 使用Proxy类的getProxyClass静态方法生成一个动态代理类stuProxyClass

Class<?> stuProxyClass = Proxy.getProxyClass(Person.class.getClassLoader(), new Class<?>[] {Person.class});

* 获得stuProxyClass 中一个带InvocationHandler参数的构造器constructor

Constructor<?> constructor = PersonProxy.getConstructor(InvocationHandler.class);

* 通过构造器constructor来创建一个动态实例stuProxy

Person stuProxy = (Person) cons.newInstance(stuHandler);

就此，一个动态代理对象就创建完毕，当然，上面四个步骤可以通过Proxy类的newProxyInstances方法来简化：

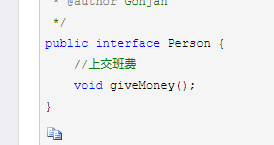
//创建一个与代理对象相关联的InvocationHandler

InvocationHandler stuHandler = new MyInvocationHandler<Person>(stu);

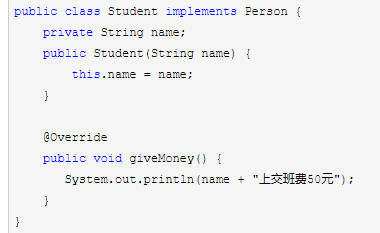
//创建一个代理对象stuProxy，代理对象的每个执行方法都会替换执行Invocation中的invoke方法

Person stuProxy= (Person) Proxy.newProxyInstance(Person.class.getClassLoader(), new Class<?>[]{Person.class}, stuHandler);

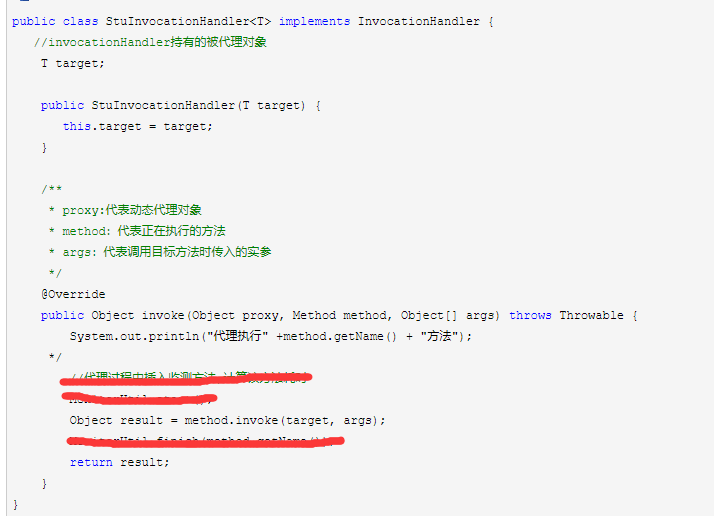
1.首先，我们创建一个Person接口。这个接口就是学生（被代理类），和班长（代理类）的公共接口，他们都有上交班费的行为。这样，学生上交班费就可以让班长来代理执行。



2. 创建需要被代理的实际类，Student类实现Person接口。Student可以具体实施上交班费的动作。



3. 创建StuInvocationHandler类，实现InvocationHandler接口，这个类中持有一个被代理对象的实例target。InvocationHandler中有一个invoke方法，所有执行代理对象的方法都会被替换成执行invoke方法。再invoke方法中执行被代理对象target的相应方法。



4.测试类



newProxyInstance，方法有三个参数：

参数1：用哪个类加载器去加载代理对象

参数2：动态代理类需要实现的接口

参数3：动态代理方法在执行时，会调用h里面的invoke方法去执行

注意其中：